TEST AND VERIFICATION PLAN

EVLA Correlator Chip Functional Test Cases

TVP document: **A25082N0002**

Revision: 1.0

Rick Smegal

January 30, 2006

Aardvark Resources

Document No. EVLA-CV-R01 Rev. 4.2

Victoria, B.C., Canada

Table of Contents

1 Preamble....5
1.1 Revision history....5
1.2 Document organization....6
1.3 Document objectives and scope....6
1.4 Limitations....6
2 Overview: Correlator chip testing....7
2.1 Testing, validation and verification....7
2.2 Overview of the validation process....8
2.3 Post simulation processing....9
2.4 Correlator chip verification....10
3 Correlator chip validation programme....11
3.1 Nomenclature and symbol definition....11
3.2 Model test vectors....12
3.3 Tests by chip operation....13
3.3.1 Summary....13
3.3.2 CMAM and correlation....20
3.3.3 accumulator overflow....22
3.3.4 normal dump and readout....22
3.3.5 shift enable clock (X|Y-SE_CLK)....26
3.3.6 input configuration switches....27
3.3.7 extreme dump rate operation....31
3.3.8 overrun detection and recovery....32
3.3.9 on-line detection and reporting of errors....35
3.3.10 X-Y dump enable offset detection....39
3.3.11 off-line detection and reporting of synchronization errors....40
3.3.12 MCSR status flags....41
3.3.13 correlation holdoff....41
3.3.14 frame abort....42
3.3.15 dump command capture....42
3.3.16 timestamp capture....44
3.3.17 data stream ID capture....45
3.3.18 enable phase rotation....45
3.3.19 enable correlation....46
3.3.20 verification....46
3.3.21 autocorrelation mode....47
4 Test Cases....48
4.1 Test case 0....49
4.2 Test case 1....55
4.3 Test case 2....60
4.4 Test case 3....65
4.5 Test case 4....70
4.6 Test case 5....75
4.7 Test case 6....80
4.8 Test case 7....85
4.9 Test case 8....90
4.10 Test case 9....95
4.11 Test case 10....100
4.12 Test case 11....105
4.13 Test case 12....110
4.14 Test case 13....115

4.15 Test case 14....120
4.16 Test case 15....125
4.17 Test case 16....130
4.18 Test case 17....135
4.19 Test case 18....140
4.20 Test case 19....145
4.21 Test case 20....150
4.22 Test case 21....155
4.23 Test case 22....160
4.24 Test case 23....165
4.25 Test case 24....170
4.26 Test case 25....175
4.27 Test case 26....180
4.28 Test case 27....185
4.29 Test case 28....190
4.30 Test case 29....195
4.31 Test case 30....200
4.32 Test case 31....205
4.33 Test case 32....210
4.34 Test case 33....215
4.35 Test case 34....220
4.36 Test case 35....225
4.37 Test case 36....230
4.38 Test case 37....235
4.39 Test case 38....240
4.40 Test case 39....245
4.41 Test case 40....250
4.42 Test case 41....255
4.43 Test case 42....261
4.44 Test case 43....267
4.45 Test case 44....272
4.46 Test case 45....277
4.47 Test case 46....282
4.48 Test case 47....288
4.49 Test case 48....294
4.50 Test case 49....299
4.51 Test case 50....304
4.52 Test case 51....310
4.53 Test case 52....316
4.54 Test case 53....321
4.55 Test case 54....326
4.56 Test case 55....332
4.57 Test case 56....338
4.58 Test case 57....343
4.59 Test case 58....348
4.60 Test case 59....353
4.61 Test case 60....358
4.62 Test case 61....363
4.63 Test case 62....368
4.64 Test case 63....373
4.65 Test case 64....378
4.66 Test case 65....383

4.67 Test case 66.....389
4.68 Test case 68.....395
4.69 Test case 69.....400
4.70 Test case 70.....405

1 Preamble

1.1 Revision history

Revision	*Date*	*Notes*	*Author*
0.0	August 17, 2004	Created under title "EVLA Correlator Chip Validation and Verification Plan", Aardvark Resources doc. no. EVLA-CV-R01	R. Smegal
1.0	September 26, 2004	**EVLA document type:** Test and Verification Plan; **EVLA document title:** "EVLA Correlator Chip Functional Test Cases"; **EVLA document no.:** A25082N0002; the number of test cases was extended; numerous errors fixed; updated test vector description; include source listing of test cases	R. Smegal
2.0	January 14, 2005	Included verification test cases 54, 55; updated test cases 15, 23, 24, 31, 36, 64; added autocorrelation section and test cases 56 through 63; added table of operations under test; added test case priority rating; corrected overrun and recovery section; made many minor corrections throughout the document.	R. Smegal
3.0	April 27, 2005	All test cases have been updated to accommodate the most recent test bench and subsequent verification; CMAM and correlation, accumulator overflow sections rewritten to reflect new test procedures; added section on converting to verification test cases; added comments on FIFO overrun testing with test cases 52 and 53; fixed errors in the dump enable offset table; minor text changes made throughout the document to reflect the new test bench.	R. Smegal
4.0	November 30, 2005	The section on conversion to verification mode test cases has been removed; reference to test vector set DS15 added; text regarding the validation process and limitations added to the section on normal dump and readout; the section regarding on-line error detection has been expanded to include test cases 65 and 66; the section on BBID capture now reflects changes in an updated test bench; many errors corrected in tables and text. The following test cases were added or updated: 3, 5, 6, 10, 11, 12, 17, 18, 19, 20, 21, 22, 23, 25, 26, 28, 30, 48, 49, 52, 53, 61, 65, 66	R. Smegal
4.1	December 16, 2005	Add reference to verification test case 67; update test case 07; add test cases 68, 69, 70; add test vector set DS16 to table.	R. Smegal
4.2	January 30, 2006	EVLA project document revision designation changed from **DRAFT** to **Revision 1.0**; test cases 68, 69, 70 added to data switch configuration tables; minor formating changes.	R. Smegal

1.2 Document organization

Section 1 (this section) pertains to the document itself and does not introduce or discuss technical information.

Section 2 contains introductory material; how validation, the main subject of this document, fits into the overall testing paradigm of the correlator chip.

Section 3 contains a detailed technical description of the proposed test cases that constitute the validation programme, organized according to chip operations.

Section 4 contains Verilog source code listings of all test cases described in the previous section.

1.3 Document objectives and scope

This document describes a validation programme for the EVLA Correlator ASIC (Application Specific Integrated Circuit) and proposes a set of test cases that extend the work on validation and verification initiated by Brent Carlson of the National Research Council of Canada. Validation and verification, as defined within the context of this document (see section 2.1), are two distinct phases within the overall test programme[1]. This document specifically addresses the former, although verification is an essential part of the overall test programme and sequel to the validation work. The overall chip test programme is described by the draft *EVLA Correlator Chip, Test and Verification Plan* document [TVP]. The *EVLA Correlator Chip*, RFS document, Revision 2.3 [RFS] and the behavioural model of the correlator provided by Brent Carlson [Mem001] comprise the correlator chip specification on which the validation is based.

The test cases described by, and included within, this document are expressed in the Verilog hardware description language of IEEE std. 1364. Extensions and revisions to these test cases will be documented in the post simulation report together with the source code for the version of the behavioural model used in this validation process.

Reference to, and inclusion of, parts of this document within other EVLA documentation is expected and allowed.

1.4 Limitations

(1) Validation and verification programmes described by this document are limited to model simulation with applied test vectors; formal verification methods [KroFHV], [Pix01] are not considered.

(2) The test vector set is designed on the basis of operational coverage. No claim is made for complete hardware and state coverage of the correlator chip functional model.

(3) The test bench used to stimulate the functional model of the chip is provided by the National Research Council of Canada. The test bench was not examined in detail by the Author to verify its operation and is accepted as given to stimulate the correlator chip model according to data level specifications (section 2.2) described in [RFS].

1 The term *Functional Verification* is often used in current literature to conjunctively refer to the processes of validation and verification as defined in this document.

2 Overview: Correlator chip testing

2.1 Testing, validation and verification

Correlator chip testing begins with a chip functional model, expressed as synthesizable code in the Verilog hardware description language, and ultimately continues with the fabricated prototype chip. The overall test programme is summarized as follows.

(1) The functional model undergoes an in-house validation process after which is submitted to a selected vendor for place-and-route.

(2) The post-place-and-route version of the functional model is returned for in-house verification to insure that functionality of the chip has not been compromised during the aforementioned process.

(3) Design refinements suggested as a consequence of the place-and-route process and vendor analysis of hardware coverage may initiate additional in-house validation-verification cycles until design convergence followed by successful verification is achieved.

(4) An in-house review of I/O performance will be made on the basis of vendor supplied models obtained following place-and-route.

(5) Hardware testing of prototypes with the objective of uncovering manufacturing defects will be the responsibility of the vendor. These tests do not require the test vector set required for validation and verification of the functional model of steps 1 to 3 above.

(6) Operational testing of the prototype chip will commence in-house by populating one or more baseline boards with a complete set of prototype chips supplied by the vendor; then performing both lab bench and in situ system tests.

The scope of this document (section 1.3) is limited to functional model testing that encompasses validation and verification, hence, remaining discussion will concentrate on these subjects.

A functional model of the correlator chip as implemented in the Verilog hardware description language is subjected to validation and verification programmes. Validation, in this context, is a process in which the functional model is stimulated and the resultant behaviour is directly compared (on a bit exact or statistical basis) to a specification. *In the case of verification*, model behaviour is compared on a bit exact basis to a previous generation or implementation of the chip with the intent on assuring that all subsequent implementations of the chip, from the pre-place-and-route model as expressed in the hardware description language to tape out and ultimately, production of the chip, are functionally identical.

Although validation and verification could ideally share a common set of stimuli, practical considerations may limit the intersection of the two sets. Stimuli for use in the validation process is selected to maximize operational coverage (that is, functionality) in the interest of demonstrating compliance with the specification but may be incomplete and exhibit redundancy from the perspective of hardware coverage. Verification, on the other hand, may exercise similar operational features, perhaps with non-realistic input data in an attempt to achieve 100% hardware coverage in terms of gate excitation and as much sequential state space as practically possible[2]. The stimuli applied by the test programme described in section 3.3 are designed primarily on the basis of validation and effective verification requires supplementation with additional tests to enhance hardware coverage. Impetus for such tests will likely arise from vendor advice on increasing hardware coverage following place-and-route.

2 Formal verification methods [KroFHV], [Pix01] offer an alternative to simulation but are not considered here, primarily due to the moderate size (in function and gate count) of this ASIC design.

2.2 Overview of the validation process

Model specification and behavioural compliance, in the context of validation, occurs on two levels, namely,

(1) *data level*, which is concerned with data handling (transfer, buffering, timing, synchronization, integrity checks, etc.), interface (bus format, handshaking, etc.) and control issues (register and buffer clear, reset, clocks and strobes, etc.);

(2) *operational (top) level* in which model behaviour is determined by the input data applied to the model and configuration data written into the registers.

Salient features of the *data level* such as interface format, timing and handshaking are described in the [RFS] and the chip model inputs are stimulated in a manner consistent with this specification. Compliance at the data level is a prerequisite to top level operation, that is to say, validation at the *operational level* also validates the *data level.*

Operational (top) level validation is achieved upon successful comparison of data generated by the functional model of the chip to that of a mathematically based behavioural model [Mem001], implemented as non-synthesizable code, when both models operate on the same input data set and both models are configured to carry out identical operations on the data. Operational level validation is complete when all chip configurations and representative input data patterns have been applied to the functional model of the chip. Unspecified chip behaviour is acceptable in the case where undefined stimuli is applied providing that the chip can return to a specified state without extraordinary intervention (such as a hardware reset).

The chip functional model is embedded in a *test bench* that is also coded in Verilog. The test bench encompasses the data level specification of the chip in that data level stimulation is hard coded within and serves to move data and chip configuration information in and out of chip buffers and registers, provide chip reset and clocking. Operational level stimuli are described by *test vectors* and *test cases*. Test vectors are contained in data files read by the test bench and subsequently applied to the chip model according to the data level specification; these test vectors are also read as input data by the behavioural model. Test cases are Verilog source files included with compilation of the test bench and describe different chip configurations and modes of operation.

The behavioural model is expressed in the C programming language and a listing of the source code for the version used in this validation process will be published in the post simulation report along with the results.

Test bench configuration data, also contained within the test case, can instruct the test bench to modify normal data level behaviour (for example, apply data synchronization errors) in order to validate predictable (as specified in [RFS]) operations of the functional model (for example, detecting synchronization errors).

The validation process is summarized as follows.

(1) Test vectors are generated by independent means prior to any testing and identified by a data set name. More than one data set may be required to provide the desired operational and hardware coverage of the chip. These data sets are described in section 3.2 .

(2) A number of test cases designed to individually invoke specific chip operations (including configuration sequences) are written in Verilog and included during compilation of the test bench. These test cases are described in detail within section 3.3 .

(3) The test bench is compiled and then linked with the functional model of the chip and one or more test cases in preparation for a simulations of the chip.

(4) The simulation is initiated; test vectors are read by the test bench and applied to the functional model; upon completion of each (test case) simulation, the test bench reads the chip registers via the **MCB** (**M**onitor and **C**ontrol **B**us) and one or more output data frames via the **LTA** Controller Interface (**LCI**) for saving to files for subsequent post simulation processing.

(5) The test vectors are read as input data by the behavioural model. A number of behavioural model runs are

executed, each with a specified number of lags and integration time consistent with respective test cases of the functional model. The results are stored as a set of files.

(6) Post simulation processing reads the files saved by the test bench (4) and behavioural model (5); interprets and analyze their contents to complete the validation process.

(7) Upon successful validation (6), the **MCB** register files and output frame files created by the test bench in (5) are saved as a set of "golden" files for use in the verification process.

2.3 Post simulation processing

Post simulation processing refers to the reduction, inspection and analysis of the data generated by the simulations and involves some combination of automatic and interactive procedures implemented in software. The process is summarized as follows.

(1) The **MCB** register files and output frame files saved by the test bench are parsed and interpreted.

(2) The behavioural model output files are read and the data matched to a corresponding test case of the functional model.

(3) Concatenated lag frames, sorted by dump time are created from the output frame data on the basis of the **MCB** register contents and concatenation flags of the output lag frame.

(4) The lag frames of (3) are compared to the output data of the behavioural model on a statistical and, in some cases, bit exact basis; both frequency and lag domains of the model data are examined.

(5) The register contents (in **MCB** register files) are checked for compliance with specifications.

(6) Lag data is inspected for specified qualitative behaviour, for example, distinct spectral features in the frequency or lag domains.

Steps (1) to (3) are procedural processes and automated by post simulation software. Arithmetic differences between the two models and generation of statistics in step (4) are automatic; sample results are subject to inspection in the form of plots and tables. However, test cases involving many dump cycles and many chains, do not reliably or easily yield to visual inspection and analysis, in which case flagging of bad statistical fits between the models is automated. Similar reasoning is applied to step (5) where test cases involving many dumps will have status bits and register contents checked with post simulation scripts. Step (6) is primarily a visual process although some simple sanity checks such as spectral peak counting can be automated if warranted.

Two sets of tools, developed independently of each other, are employed in the post simulation processing and analysis. One provides an interactive GUI based user interface and runs in a Microsoft Windows® environment [LauTAS]. The other is a library of Octave© functions [OctWWW] that read register and frame data into structures for use in analysis and reporting scripts. The former is easy to use and lends itself to quick inspection of the simulation results while the latter is flexible, easily extensible and allows scripting to automatically process and analyze large amounts of data. The Octave post simulation functions and scripts will be listed in the post simulation report.

2.4 Correlator chip verification

Verification is applied to the functional model after successful validation. The process is summarized as follows

(1) The identical test bench and test cases used for validation are compiled (with ideally the same compiler and simulator version) together with the post-place-and-route model (that is, gate level model) of the chip provided by the vendor.

(2) The same test vectors used for validation are read by the test bench and applied to the chip model; the simulation executed and the register and output frame files saved.

(3) The chip is successfully verified if the output frame and register contents represented in the golden files of the validation process match the corresponding post-place-and-route files on a bit exact basis.

Software utilities for automating the bit exact comparison of (3) have been developed for this test environment [LauTS].

Due to excessive execution time in simulating the post-place-and-route gate level model, only a subset of the validation test cases may be selected for verification once the gate level model of the chip is made available by the manufacturer. Moreover, parameters of the test cases selected for verification may be adjusted to decrease execution time and, as a consequence, a final validation run performed as part of the verification programme. Details regarding the verification process together with final validation and verification results will be made available in a final verification report.

An additional verification step may be applied to the post-place-and-route model. This test entails applying random input test vector stimulation to all inputs, data and control, of the functional model without regard to the data level specification. The test vectors will be designed to cover as much state space as is practical. The primary test objective is to uncover any state cycles that finite state machinery within the model cannot leave. This test has not been included within the test case set outlined in section 3.3 but is part of the overall test and verification programme [TVP].

3 Correlator chip validation programme

3.1 Nomenclature and symbol definition

The following conventions and symbols are used within the remainder of this report.

(1) **BOLD** font: names of quantities referred to in the [RFS] such as register names, control lines and data stream names (examples, **XSTATUS**, **XSyncerr**, **X-DVALID**, **X-SE_CLK** also see points 8, 9, 10).

(2) n`b####: binary notation, *n* digits decreasing in significance from left to right.

(3) `h####: hexadecimal notation, digits decreasing in significance from left to right.

(4) XY: shorthand notation for *X* and *Y*.

(5) X|Y: shorthand notation for (*X or Y*) or (*XY*), that is, inclusive *or*

(6) chain: (*aka* correlator cell chain, cell chain, CC chain, CCC chain) a concatenated series of adjacent correlator cells within the chip.

(7) input configuration notation, *inpX(nb:ne)inpY* : a concatenated chain of complex correlator cells where *inpX* and *inpY* are the *X* and *Y* inputs respectively; *nb* is the starting (*X* input) cell number and *ne* is the end cell according to the numbering scheme described in [RFS]. The symbol, - , in place of *inpX* or *inpY* indicates a *no-data*[3] input. An uppercase X in place of *inpY* indicates that the autocorrelation mode is invoked. Chains configured with undefined switch settings are not shown.

(8) input data streams: four bit, [0:3], parallel data in any or all of eight streams, [0:7], **X|Y-SDATA[0:7][0:3]**; **X|Y-PHASE[0:7][0:3]**; single bit data valid line, **X|Y-DVALID[0:7]**

(9) control lines: refer to any of the serial data lines **X|Y-SCHID_FRAME***, **X|Y-DUMP_SYNC**, **X|Y-DUMP_EN[0:7]**

(10) dump control lines: **X|Y-DUMP_SYNC**, **X|Y-DUMP_EN[0:7]**

(11) *[f-b]* : arithmetic difference between complex lags of the functional and behavioural models, is an array of N_L elements such that $[f-b]=f(i)-b(i),\ \ for\ i=0..(N_L-1)$ where *i* is the lag number, *f* and *b* are arrays of the N_L complex correlation coefficients of the functional and behavioural models respectively, N_L is the number of lags in the chain.

(12) $[f\text{-}f]_j$: arithmetic difference between complex lags of two adjacent dumps of the functional model, is an array of N_L elements such that $[f-f]=f(i)-f(i),\ \ for\ i=0..(N_L-1)$ where *i* is the lag number, *f* is an array of N_L complex correlation coefficients and *j* refers to a pair of adjacent dumps (*j* increasing with the time of the dump), N_L is the number of lags in the chain.

(13) $[b\text{-}b]_j$: A behavioural model analogy to that of the functional model in (12).

(14) $[f^F\text{-}b^F]_j$, $[f^F\text{-}f^F]_j$, $[b^F\text{-}b^F]_j$: Analogous quantities of (11) , (12), (13) where the *F* superscript indicates the lag data has been transformed to the frequency domain.

(15) *n-a* in tables is interpreted as “not-applicable”.

3 Cells of a chain containing such an input will not generate output lag data.

3.2 Model test vectors

The test vectors are data contained within files, read by the test bench and then applied to the functional model in accordance to the data level specification. These data are also read as input to the behavioural model. Data sets are generated by independent means (see [Mem001] for background information) as opposed to an integral function of the test bench or simulator; their size and features listed in Table 3.2.1 .

These data encompass characteristics of real world operation and are designed to invoke easily recognizable model behaviour. Specifically, *X* and *Y* data associated with the *same stream number, n,* will provide both continuum and spectral line correlation where the number of spectral lines is $n+1$. Otherwise, no continuum correlation exists and the number of correlated spectral lines is the lesser of $1+n_X$ or $1+n_Y$. The prominent features of correlated continuum and spectral components are a single line in the lag domain and multiple lines in the frequency domain, respectively. The test vectors contain phase rotation as well as occasional and independent *X* and *Y* data valid resets.

test set	***number of samples***	***input stream numbers***	***description***
DS4	2,500,000	0 through 7	Data valid asserted, in general; used for most test cases.
DS10	50,000	0	Created from the first 50,000 samples of DS4. Data valid is edited low for the first 4,000 samples and again for the remaining samples following an unmodified block of 10000 samples. This data set has been deprecated and is not used by the latest revision of the test case suite. However, this data set can be used for bit exact comparison between functional and behavioural models over a longer integration time than with DS4.
DS11	150,000	0	Created from the first 150,000 samples of DS4; data valid is edited low for the first 4,000 samples and again for the remaining samples following an unmodified block of 50,000 samples. This data set has been deprecated and is not used by the latest revision of the test case suite. However, this data set can be used for bit exact comparison between functional and behavioural models over an integration of sufficient length to toggle all bits of a 23 bit accumulator without invoking the split accumulator mode.
DS14	50,000	0	Created from the first 50,000 samples of DS4. Data valid is edited low for the first 3,000 samples and again for the remaining samples following an unmodified block of 5800 samples. This data set is used for bit exact comparison between functional and behavioural models over an integration interval of sufficient length to toggle all bits of a 23 bit accumulator operating in split accumulator mode but short enough to allow verification and validation with a common test case.
DS15	48,000	X:4, Y:0	Created from the first 50,000 samples of DS4. Data is edited to more closely resemble that applied to the RTL model by the test bench. This data set designation is reserved for special experiments.
DS16	50,000	1	See DS14. Used for bit exact tests that involve stream 1.

Table 3.2.1 The sets of test vectors read by both the test bench (and applied to the functional model) and behavioural model are tabulated above. These data reside in files within the test environment.

3.3 Tests by chip operation

3.3.1 Summary

A number of chip operations are tested individually by compiling and executing the test bench with a series of test cases. Some operations require the conjunctive execution of several test cases while some test cases are involved in the testing of more than one operation. A description of chip operations, together with the test cases and post simulation checks required for validation, follow in subsequent sections; the post simulation report of results will follow a similar structure.

A summary of chip operations under test and the primary set of test cases required for their validation appears in Table 3.3.1 below; individual test cases are summarized by numerical order in Table 3.3.3 . A test case priority rating is included in Table 3.3.3 and interpretation of the numerical levels in Table 3.3.2.

Operations under test (section)	*Test cases*	*Notes*
CMAM and correlation (3.3.2)	0, 32, 33	These tests feature a bit exact comparison with the behavioural model and all bits of the accumulator are toggled.
accumulator overflow(3.3.3)	1	
normal dump and readout (3.3.4)	2, 3, 5, 6, 8, 9, 10, 11, 12, 14, 21, 22, 23, 25, 30, 64, 69, 70	These tests exercise the dump and readout operations of the chip; multiple dumps and dump rates are used with various chip configurations.
configuration: 2048 lag, single input, 1 pol.	0, 4, 32, 33	Modulo shift enable clocking is applied in test case 33. Input taken from stream 1 in test case 4 but with frequent input synchronization errors applied.
configuration: 1024 lag, 1 pair, 2 pol./pair	64	
configuration: 512 lags, 1 pair, 4 pol./pair	14, 34	Test case 14 is required to test modulo shift enable clocking in this configuration.
configuration: 512 lags, 4 input, 1 pol./input	3, 6	Tests run with odd and even numbered streams; shift enable clocking is applied in test case 6.
configuration: 256 lags, 4 distinct pair, 2 pol./pair	10, 11	Modulo shift enable clocking is used in test case 10; occasional input synchronization errors are applied in 11.
configuration: 128 lags, 4 distinct pair, 4 pol./pair	30	A distinct baseband input stream pair is active for each quad.
redundant quad configuration: 128 lags, 1 pair, 4 pol./pair	48, 49	All four quads of the chip are fed from a common input pair; test cases 48 and 49 are coupled with X-Y dump enable offset tests.
redundant cell configuration: 128 lags, single input, 1 pol.	2, 5	All cells of the chip are fed from a common input stream; both master chip input streams (0, 1) are used.
configuration: secondary input stream connection of quad 0	15	This tests a configuration that results in no frames being generated for cells of quad 0.
configuration: disable cells with an *undefined* cell switch setting	8, 9, 23, 68	Selected cells are disabled by setting either the *X* or *Y* inputs of the cell to *undefined*; no lags are generated by cells within the chain beginning or terminating with disabled cells.
configuration: 6x0, 6x1, 7x0, 7x1 in quad 0	68	
configuration: extend data switch state coverage	15, 16, 17, 18, 19, 20, 21, 22, 28 50, 51, 52, 53, 54	These tests consider additional chip and quad configurations that, taken together with the remaining tests, extend the state coverage of the cell and quad input data switches.

Operations under test (section)	*Test cases*	*Notes*
modulo shift enable clocking (3.3.5)	4, 8, 9 ,10 , 32, 33, 14, 34	Multiple clock modulus applied with respective input streams; various concatenated chain lengths. These tests are run and analyzed together with the normal dump and readout tests.
overrun detection and recovery (3.3.8)	24, 29, 31, 69, 70	Dump overruns are invoked with varying rates of occurrence. Check for detection; frames that do not experience an overrun are recovered.
extreme dump rate operation (3.3.7)	7, 69, 70	Use multiple dump rates for the chip where the rates for some cells approach the minimum of 1 μs. Recovery from overrun cannot occur; watch for unexpected chip operation.
X|Y dump control	69, 70	Specific check for consistent operation between X and Y dump control.
on-line detection and reporting of errors (3.3.9)	3, 4, 5, 6, 11, 12, 13, 23, 26, 27, 35, 36, 37, 46, 47, 50, 51, 52, 53, 65, 66	These tests involve input stream, dump control and timestamp synchronization errors; missing shift enable clocks and *X-Y* misalignment of the XY-SCHID_FRAME.
off-line detection and reporting of errors (3.3.11)	38, 39, 40	Internal chip test vectors are enabled and the applied data contain synchronization errors in input streams, dump control lines, timestamp and shift enable clocks.
X-Y dump offset detection (3.3.10)	5, 11, 23, 37, 48, 49	Test cases 48, 49 experience an *X-Y* dump offset on odd and even cells respectively; the remaining test cases have an offset applied simultaneously to all dump enable lines.
correlation holdoff (3.3.13)	42	
frame abort (3.3.14)	43	A FRAME_ABORT* sequence applied to the LTA interface discards the current frame without affecting correlation.
dump command capture (3.3.15)	41, 42, 46, 47	Capture dump command, recirculation block and phase bin data from the dump control lines.
timestamp capture (3.3.16)	17, 18, 19, 20	Capture the 32 bit timestamp from the X|Y-TIMESTAMP line. *X-Y* selection is made on the basis of dump control.
data stream ID capture (3.3.17)	17, 18, 19, 20	Capture the station, baseband and sub-band identification from the input data streams.
enable phase rotation (3.3.18)	44	Phase rotation is normally enabled; this test disables phase rotation.
enable correlation (3.3.19)	45	Correlation is normally enabled; this test disables correlation.
input FIFO buffer length	52, 53	Test the depth of the input FIFO buffer.
autocorrelation mode	56, 57, 58, 59, 60, 61, 62, 63	
verification	54, 55, 67	Used for verification; many operations are tested although frame building and dump overrun are of primary interest in 54 and 55.

Table 3.3.1 The test programme is summarized according to chip operation in the above table. The test cases listed comprise a set of principal tests required to validate the operation. The section of this document that pertains to an operation and respective test is contained within the parenthesis of the leftmost column.

priority rating	*description*
5	Essential test cases that are used for verification; operations represented by these tests have been previously validated.
4	These test cases are essential for both validation and verification.
3	These test cases are essential for validation and recommended for verification.
2	This level applies to the majority of test cases; recommended for validation but optional for verification. In many cases, omitting these tests will result in one or more chip operations being inconclusively validated.
1	Suggested for validation; these tests increase confidence in the validation, extend the functional coverage of a set of tests or validate a minor chip operation.
0	These are optional test cases that are, for the most part, redundant but may enhance confidence in validation.

Table 3.3.2 The test case priority rating incorporated into Table 3.3.3 is described above. Test cases of levels 3 through 5 are essential to the programme.

Test case	***Input configuration inpX(c:c)inpY***	***Shift Enable clocking***	***Mult. dump***	***Mult. dump rate***	***Total test time[4] [clock cycles]***	***Principal operations under test (section)***	***Priority***
0	0(0:15)0	no	no	no	620	CMAM and correlation (3.3.2)	3
1	0(0:15)0	no	no	no	812	accumulator overflow(3.3.3)	3
2	0(*n*)0, *for n*=0...15	no	yes	no	60,000	normal dump and readout (3.3.4)	2
3	0(0:3)0 2(4:7)2 4(8:11)4 6(12:15)6	no	yes	yes	96,000	normal dump and readout (3.3.4); on-line detection of errors (3.3.9); *X-Y* dump enable offset detection (3.3.10)	2
4	1(0:15)1	yes	no	no	10,000	on-line detection of errors (3.3.9); shift enable clock (3.3.5);	1
5	1(*n*)1, *for n*=0...15	yes	yes	no	18,000	normal dump and readout (3.3.4); on-line detection of errors (3.3.9); *X-Y* dump enable offset detection (3.3.10)	2
6	1(0:3)1 3(4:7)3 5(8:11)5 7(12:15)7	yes	yes	yes	96,000	normal dump and readout (3.3.4); on-line detection of errors (3.3.9); shift enable clock (3.3.5);	2
7	0(0)0 0(1)0 0(2)0 0(3)0 2(4)2 2(5)2 2(6)2 2(7)2 4(8)4 4(9)4 4(10)4 4(11)4 6(12)6 6(13)6 6(14)6 6(15)6	no	yes	yes	20,000	extreme dump rate operation (3.3.7)	1
8	0(0)0 1(3)1 2(4)2 3(7)3 4(8)4 5(11)5 6(12)6 7(15)7	yes	yes	no	60,000	normal dump and readout (3.3.4); shift enable clock (3.3.5); input switch configuration (3.3.6)	2

4 This total does not include clock cycles for initialization and frame read out.

Test case	*Input configuration* *inpX(c:c)inpY*	*Shift Enable clocking*	*Mult. dump*	*Mult. dump rate*	*Total test time* *[clock cycles]*	*Principal operations under test (section)*	*Priority*
9	0(1)0 1(2)1 2(5)2 3(6)3 4(9)4 5(10)5 6(13)6 7(14)7	yes	yes	no	60,000	normal dump and readout (3.3.4); shift enable clock (3.3.5); input switch configuration (3.3.6)	2
10	0(0:1)0 1(2:3)1 2(4:5)2 3(6:7)3 4(8:9)4 5(10:11)5 6(12:13)6 7(14:15)7	yes	yes	no	40,000	normal dump and readout (3.3.4); shift enable clock (3.3.5)	1
11	0(0:1)0 1(2:3)1 2(4:5)2 3(6:7)3 4(8:9)4 5(10:11)5 6(12:13)6 7(14:15)7	no	yes	no	18,000	on-line detection of errors (3.3.9); normal dump and readout (3.3.4); *X-Y* dump enable offset detection (3.3.10)	2
12	0(0:1)0 1(2:3)1 2(4:5)2 3(6:7)3 4(8:9)4 5(10:11)5 6(12:13)6 7(14:15)7	no	yes	no	18,000	on-line detection of errors (3.3.9); normal dump and readout (3.3.4)	2
13	0(0:1)0 0(2:3)0 0(4:5)0 0(6:7)0 0(8:9)0 0(10:11)0 0(12:13)0 0(14:15)0	yes	no	no	5,000	on-line detection of errors (3.3.9)	1
14	0(0:3)0 0(4:7)1 1(8:11)0 1(12:15)1	yes	yes	no	30,000	normal dump and readout (3.3.4); shift enable clock (3.3.5)	2
15	7(0:3)- 1(4:7)1 1(8:11)1 1(12:15)1	no	no	no	5,000	input switch configuration (3.3.6)	1
16	1(0:3)1 1(4:7)1 1(8:11)1 1(12:15)1	yes	no	no	5,000	input switch configuration (3.3.6); shift enable clock (3.3.5)	1
17	1(0:3)1 1(4:7)3 1(8:11)5 1(12:15)7	no	yes	no	12,000	input switch configuration (3.3.6); data stream ID capture (3.3.17); timestamp capture (3.3.16)	2
18	1(0:3)1 3(4:7)1 5(8:11)1 7(12:15)1	no	yes	no	12,000	input switch configuration (3.3.6); data stream ID capture (3.3.17); timestamp capture (3.3.16)	2
19	0(0:3)0 0(4:7) 2 0(8:11)4 0(12:15)6	no	yes	no	12,000	input switch configuration (3.3.6); data stream ID capture (3.3.17); timestamp capture (3.3.16)	2
20	0(0:3)0 2(4:7)0 4(8:11)0 6(12:15)0	no	yes	no	12,000	input switch configuration (3.3.6); data stream ID capture (3.3.17); timestamp capture (3.3.16)	2
21	0(0:7)3 4(8:15)7	yes	yes	yes	24,000	normal dump and readout (3.3.4); shift enable clock (3.3.5);	1
22	0(0:7)3 4(8:15)7	yes	yes	no	24,000	shift enable clock (3.3.5); normal dump and readout (3.3.4)	1

Test case	*Input configuration* *inpX(c:c)inpY*	*Shift Enable clocking*	*Mult. dump*	*Mult. dump rate*	*Total test time* *[clock cycles]*	*Principal operations under test (section)*	*Priority*
23	7(0:2)0 3(6:7)3 4(8)4	no	yes	yes	20,000	input switch configuration (3.3.6); on-line detection of errors (3.3.9); *X-Y* dump enable offset detection (3.3.10); normal dump and readout (3.3.4)	2
24	1(0:1)1 0(2:3)0 0(4:5)1 1(6:7)0 1(8:9)1 0(10:11)0 0(12:13)1 1(14:15)0	no	yes	no	30,000	overrun detection and recovery (3.3.8)	1
25	1(0:1)1 0(2:3)0 0(4:5)1 1(6:7)0 1(8:9)1 0(10:11)0 0(12:13)1 1(14:15)0	yes	yes	no	60,000	normal dump and readout (3.3.4)	2
26	1(0:1)1 0(2:3)0 0(4:5)1 1(6:7)0 1(8:9)1 0(10:11)0 0(12:13)1 1(14:15)0	no	yes	no	12,000	on-line detection of errors (3.3.9)	1
27	1(0:1)1 0(2:3)0 0(4:5)1 1(6:7)0 1(8:9)1 0(10:11)0 0(12:13)1 1(14:15)0	no	no	no	5,000	on-line detection of errors (3.3.9)	1
28	0(0:2)1 1(3:7)3 5(8:15)6	no	yes	no	12,000	normal dump and readout (3.3.4)	1
29	0(0)0 0(1)0 0(2)0 0(3)0 2(4)2 2(5)2 2(6)2 2(7)2 4(8)4 4(9)4 4(10)4 4(11)4 6(12)6 6(13)6 6(14)6 6(15)6	no	yes	yes	24,000	overrun detection and recovery (3.3.8)	2
30	1(0)1 1(1)0 0(2)1 0(3)0 3(4)3 3(5)2 2(6)3 2(7)2 5(8)5 5(9)4 4(10)5 4(11)4 7(12)7 7(13)6 6(14)7 6(15)6	no	yes	yes	96,000	normal dump and readout (3.3.4)	2
31	1(0)1 1(1)0 0(2)1 0(3)0 3(4)3 3(5)2 2(6)3 2(7)2 5(8)5 5(9)4 4(10)5 4(11)4 7(12)7 7(13)6 6(14)7 6(15)6	no	yes	yes	64,000	overrun detection and recovery (3.3.8)	1
32	0(0:15)0	no	no	no	10,000	CMAM and correlation (3.3.2)	2
33	0(0:15)0	yes	no	no	20,000	CMAM and correlation (3.3.2)	4
34	0(0:3)0 0(4:7)1 1(8:11)0 1(12:15)1	no	no	no	40,000	shift enable clock (3.3.5)	0
35	0(0:3)0 0(4:7)1 1(8:11)0 1(12:15)1	no	no	no	5,000	on-line detection of errors (3.3.9)	1
36	0(0:1)0 1(2:3)1 2(4:5)2 3(6:7)3 4(8:9)4 5(10:11)5 6(12:13)6 7(14:15)7	no	no	no	5,000	on-line detection of errors (3.3.9)	2

Test case	*Input configuration* *inpX(c:c)inpY*	*Shift Enable clocking*	*Mult. dump*	*Mult. dump rate*	*Total test time* *[clock cycles]*	*Principal operations under test (section)*	*Priority*
37	0(0:15)0	no	no	no	5,000	on-line detection of errors (3.3.9); *X-Y* dump enable offset detection (3.3.10)	1
38	0(0:15)0	n-a	n-a	n-a	5,000	off-line error detection and reporting (3.3.11)	2
39	0(0:15)0	n-a	n-a	n-a	5,000	off-line error detection and reporting (3.3.11)	2
40	0(0:15)0	n-a	n-a	n-a	5,000	off-line error detection and reporting (3.3.11)	2
41	0(0:1)0 1(2:3)1 2(4:5)2 3(6:7)3 4(8:9)4 5(10:11)5 6(12:13)6 7(14:15)7	no	no	no	5,000	correlation holdoff (3.3.13); data stream ID capture (3.3.17); dump command capture (3.3.15)	2
42	0(0:1)0 1(2:3)1 2(4:5)2 3(6:7)3 4(8:9)4 5(10:11)5 6(12:13)6 7(14:15)7	no	no	no	5,000	correlation holdoff (3.3.13); data stream ID capture (3.3.17); dump command capture (3.3.15)	2
43	0(0:3)0 2(4:7)2 4(8:11)4 6(12:15)6	no	yes	yes	40,000	frame abort (3.3.14)	2
44	1(0)1 0(1)0 1(2)1 0(3)0 2(4)2 2(5)2 3(6)3 3(7)3 5(8)5 5(9)5 4(10)4 4(11)4 6(12)6 7(13)7 6(14)6 7(15)7	no	no	no	20,000	enable phase rotation (3.3.18)	2
45	0(0:15)0	no	no	no	500	enable correlation (3.3.19)	2
46	0(0:1)0 1(2:3)1 2(4:5)2 3(6:7)3 4(8:9)4 5(10:11)5 6(12:13)6 7(14:15)7	no	no	no	5,000	data stream ID capture (3.3.17); dump command capture (3.3.15); on-line detection of errors (3.3.9);	2
47	0(0:1)0 1(2:3)1 2(4:5)2 3(6:7)3 4(8:9)4 5(10:11)5 6(12:13)6 7(14:15)7	no	no	no	5,000	data stream ID capture (3.3.17); dump command capture (3.3.15); on-line detection of errors (3.3.9)	2
48	0(0)0 0(1)1 1(2)1 1(3)0 0(4)0 0(5)1 1(6)1 1(7)0 0(8)0 0(9)1 1(10)1 1(11)0 0(12)0 0(13)1 1(14)1 1(15)0	no	no	no	500	*X-Y* dump enable offset detection (3.3.10)	3
49	1(0)0 1(1)1 0(2)1 0(3)0 1(4)0 1(5)1 0(6)1 0(7)0 1(8)0 1(9)1 0(10)1 0(11)0 1(12)0 1(13)1 0(14)1 0(15)0	no	no	no	500	*X-Y* dump enable offset detection (3.3.10)	3

Test case	*Input configuration* *inpX(c:c)inpY*	*Shift Enable clocking*	*Mult. dump*	*Mult. dump rate*	*Total test time* *[clock cycles]*	*Principal operations under test (section)*	*Priority*
50	1(0)1 1(1)0 0(2)1 0(3)0 3(4)3 3(5)2 2(6)3 2(7)2 5(8)5 5(9)4 4(10)5 4(11)4 7(12)7 7(13)6 6(14)7 6(15)6	no	no	no	2,500	on-line detection of errors (3.3.9)	3
51	1(0)1 1(1)0 0(2)1 0(3)0 3(4)3 3(5)2 2(6)3 2(7)2 5(8)5 5(9)4 4(10)5 4(11)4 7(12)7 7(13)6 6(14)7 6(15)6	no	no	no	2,500	on-line detection of errors (3.3.9)	3
52	0(0:15)0	no	no	no	250	on-line detection of errors (3.3.9)	3
53	0(0:15)0	no	no	no	250	on-line detection of errors (3.3.9)	3
54	1(0)1 0(1)0 1(2)1 0(3)0 2(4)2 2(5)2 3(6)3 3(7)3 5(8)5 5(9)5 4(10)4 4(11)4 6(12)6 7(13)7 6(14)6 7(15)7	yes	yes	yes	40,000	verification (3.3.20)	5
55	7(0)- 7(1)0 0(2)- 0(3)0 1(4)1 1(5)2 2(6)2 2(7)1 5(8)0 5(9)5 0(10)5 0(11)5 7(12)0 7(13)0 0(14)7 7(15)7	yes	yes	yes	5,120	verification (3.3.20)	5
56	0(0:15)X	yes	no	no	10,000	autocorrelation (3.3.21)	2
57	1(0:15)X	yes	no	no	10,000	autocorrelation (3.3.21)	2
58	2(0:15)X	yes	no	no	10,000	autocorrelation (3.3.21)	2
59	3(0:15)X	yes	no	no	10,000	autocorrelation (3.3.21)	2
60	4(0:15)X	yes	no	no	10,000	autocorrelation (3.3.21)	2
61	5(0:15)X	yes	no	no	10,000	autocorrelation (3.3.21)	2
62	6(0:15)X	yes	no	no	10,000	autocorrelation (3.3.21)	2
63	7(0:15)X	yes	no	no	10,000	autocorrelation (3.3.21)	2
64	0(0:7)0 1(8:15)1	yes	yes	no	20,000	normal dump and readout (3.3.4)	2
65	1(0)1 1(1)0 0(2)1, 0(3)0 3(4)3 3(5)2 2(6)3 2(7)2 5(8)5 5(9)4 4(10)5 4(11)4 7(12)7, 7(13)6 6(14)7 6(15)6	no	no	no	2,500	on-line detection of errors (3.3.9)	2
66	1(0)1 1(1)0 0(2)1, 0(3)0 3(4)3 3(5)2 2(6)3 2(7)2 5(8)5 5(9)4 4(10)5 4(11)4 7(12)7, 7(13)6 6(14)7 6(15)6	no	no	no	2,500	on-line detection of errors (3.3.9)	2

Test case	*Input configuration* *inpX(c:c)inpY*	*Shift Enable clocking*	*Mult. dump*	*Mult. dump rate*	*Total test time* *[clock cycles]*	*Principal operations under test (section)*	*Priority*
67	1(0)1 1(1)0 0(2)1 0(3)0 3(4)3 3(5)2 2(6)3 2(7)2 5(8)5 5(9)4 4(10)5 4(11)4 7(12)7 7(13)6 6(14)7 6(15)6	yes	no	no	250	verification only	5
68	6(0)0 6(1)1 7(2)0 7(3)1 0(8)0 0(9)1 1(10)0 1(11)1 6(12)0 6(13)1 7(14)0 7(15)1	yes	no	no	500	input switch configuration (3.3.6)	3
69	0(0)0 0(1)0 1(2)1 1(3)1 2(4)2 2(5)2 3(6)3 3(7)3 4(8)4 4(9)4 5(10)5 5(11)5 6(12)6 6(13)6 7(14)7 7(15)7	no	yes	yes	64,000	overrun detection and recovery (3.3.8); extreme dump rate operation (3.3.7); normal dump and readout (3.3.4)	2
70	0(0)0 0(1)0 1(2)1 1(3)1 2(4)2 2(5)2 3(6)3 3(7)3 4(8)4 4(9)4 5(10)5 5(11)5 6(12)6 6(13)6 7(14)7 7(15)7	no	yes	yes	64,000	overrun detection and recovery (3.3.8); extreme dump rate operation (3.3.7); normal dump and readout (3.3.4)	2

Table 3.3.3 The above table contains a summary of test cases together with the principal operations they address by subsection name; the subsection number is contained within the parenthesis following the name.

3.3.2 CMAM and correlation

Test cases: 0, 32, 33

Description: The basic correlation operation of the functional model provides an identical result (within floating point arithmetic error of the post simulation processor) to the behavioural model when identical data is processed by both. The **CMAM** is intimately involved in correlation, hence, operation of the **CMAM** is validated, in part, by successful correlation. In addition to basic correlation, validation of the **CMAM** requires that all bits of the accumulator be exercised during an integration.

Stimulation: The test bench applies data streams, generated from input test vectors, to the functional model of the correlator chip while using a special data saving feature to facilitate bit exact comparisons with the behavioural model. Bit exact comparisons (which are extremely sensitive tests of **CMAM** operation) between the two models require identical data be applied to both the lag chain of the functional model and input to the behavioural model. Use of test vectors designed specifically for bit exact comparison, in conjunction with the data saving facility incorporated within the test bench, guarantees that identical data participate in correlation within both models.

Bit exact test vectors contain data valid bits that are reset for samples corresponding to over one lag chain length (1024 for a fully concatenated chip) at both beginning and end of an integration interval, effectively removing data registration uncertainty due to pipeline delays accrued by the input data as a consequence of buffering and synchronization. In addition, these invalid sample blocks are long enough to contain data that is lost during transfer from active to storage registers of the functional model during a dump sequence (approximately 1 μs) so that these data are not correlated within the behavioural model.

The data saving feature of the test bench resets the data valid bit corresponding to station channel identification data embedded in the applied input streams and saves these data to special data sets, designated by the *-S* suffix in Table 3.3.4; these data are subsequently applied to the behavioural model.

Bit exact test vectors are created by editing the data valid bit within set DS4 such that a block of samples corresponding to the start and end of an integration are made generally invalid. The resulting vector sets are given new designations according to Table 3.2.1 and applied to the test bench along with test cases involving a single dump (integration) of compatible duration. Test cases 32 and 33 have dump intervals of 10,000 and 20,000 samples respectively. They utilize the test vector set DS14 which contains a block of approximately 5000 *XY-valid* (*X* and *Y* data valid bits are coincidentally asserted) samples out of a total of 50,000.

In selecting the dump interval, the minimum size considered was determined by a desire to establish a base statistical comparison between these test cases and statistically similar tests of like sample size when the (non bit exact) test vector set DS4 is used. Moreover, the integration is long enough to toggle most of the bits (20) of a 23 bit accumulator in normal (non split) mode. The maximum limit of the dump interval is constrained by simulation time, especially for verification.

Test case 0 sets the *split accumulator mode* within the functional model in order to facilitate testing all accumulator bits. When the split accumulator mode is set, the top 14 bits of the accumulator are split into two, identically incremented, 7 bit segments. Hence, all bits can be toggled over a short dump interval; essential for verification where model execution can be extremely slow.

The test bench is configured (by a parameter set in test case 0) to generally reset the data valid line and accumulate only the 64/*sample* bias[5] over the dump interval. A 660 sample dump interval provides an accumulated bias of 42240 (16b`1010010,100000000). Splitting the accumulator yields a value of 5416192 (23b`1010010,1010010,100000000). The correlation holdoff (Section 3.3.13) and dump interval were selected to guarantee[6] an accumulation of 660 samples.

The chip is configured for 2048 lag concatenation in test cases 0, 32 and 33 although test case 33 invokes a shift enable clock (SE_CLK) of modulus 2 with intent to demonstrate, with bit exact precision, correct operation of the shift enable clock (see Section 3.3.5).

Post simulation checks: A bit exact comparison between model simulations with both test cases 32 and 33 is made by obtaining the output lag difference, $[f\text{-}b]^S$ where the *S* superscript specifies that the bit exact data set is applied to the behavioural model. A null difference that is consistent with the floating point arithmetic limit of the machine used for post simulation processing indicates a successful test.

The bias associated with each cell and all correlation coefficients obtained for each of the 128 lags of a cell in test case 0 contain the value 5416192 (23b`1010010,1010010,100000000) for a successful test; the split accumulator mode is presumed set.

The bit exact functional-behavioural model comparison using the special saved data sets comprise the validation process; post place and route verification does not require the use of these data. Verification can be performed with test case 32 and 33 using test vector set DS4 (provided of course, that test vector set DS4 is applied to at least one simulation run of all models).

Test cases 32 and 33 are also run as a validation with test vector set DS4 (in addition to DS14) to establish a statistical model benchmark for use in subsequent tests. The choice of data set for test case 0 is irrelevant since the data valid line is reset throughout the simulation.

5 This arithmetic bias is a consequence of an accumulator design that accommodates 3-level phase rotation.

6 This was checked by running test case 0 in normal accumulator mode.

test case	*input configuration*	*SE clock modulus*	*samples / dump (functional model)*	*samples / dump (behavioural model)*	*valid xy samples (behavioural model)*	*functional model input data set*	*behavioural model input data set*
32	0(0:15)0	n-a	10,000	10,000	8741	DS4	DS4
33	0(0:15)0	2	20,000	10,000	8741	DS4	DS4
32	0(0:15)0	n-a	10,000	10,000	5342	DS14	DS14-S32
33	0(0:15)0	2	20,000	10,000	5342	DS14	DS14-S33

Table 3.3.4 Configuration and input stimulation used for validation and verification of the CMAM hardware and fundamental correlation operation. The "S" extension in the right most column designates a special version of the input stream that is saved by the test bench for application to the behavioural model. Verification can be performed with test cases 32 and 33 using test vector set DS4.

3.3.3 accumulator overflow

Test cases: 1

Description: The **AOV** bit in the **MCSR** and status bits in the output frame are set when an accumulator overflow is detected.

Stimulation: Test case 1 sets the *split accumulator mode* within the functional model (see Section 3.3.2). The test bench is configured (by a parameter set in test case 1) to generally reset the data valid line and accumulate only bias over the dump interval. A 1044 sample dump interval provides an accumulated bias of 66816 (17b`1,0000010,100000000). Splitting the accumulator yields a value of 132352 (23b`0000010 ,0000010,100000000). The correlation holdoff (Section 3.3.13) and dump interval are selected to guarantee[7] an accumulation of 1044 samples.

Post simulation checks: The bias associated with each cell and all correlation coefficients obtained for each of the 128 lags of a cell in test case 1 contain the value 132352 (23b`0000010,0000010,100000000) for a successful test in which the split accumulator mode is set. Assure that the overflow status bit for each cell in the output frame and the **AOV** bit in the **MCSR** are set upon completion of the test.

3.3.4 normal dump and readout

Test cases: 2, 3, 5, 6, 8, 9, 10, 11, 12, 14, 21, 22, 23, 25, 28, 30, 64

Description: The tests of this section demonstrate normal dump and readout operations for various chip configurations. The chip is operated so that valid chains and output lag frames are formed. Dump operations that produce abnormal lag frames (with, for example, over run, over flow and correlation abort conditions) are considered in other sections.

Stimulation: The test bench is run with test cases that stimulate the functional model in various dump modes, dump rates, shift enable clock rates and chip input configurations (with input synchronization errors embedded within the input data streams of some test cases). The data saving feature of the test bench is not invoked with multiple dump tests and the input data does not contain leading invalid data as in the bit exact tests of Section 3.3.2.

7 This was checked by running test case 1 in normal accumulator mode.

Post simulation checks: Validation of a normal dump and readout operation occurs on a comparative basis between test cases and using either test case 32 or 33 as a reference. On the basis of bit exact tests, statistical parameters obtained for test case 32 and 33 represent those of a validated model and are thus defined as a reference. Exceptionally large deviations of observed statistical parameters (points 4, 5 below) from those of the reference indicate possible validation failure. Lack of a statistical model prevents a rigorous determination of thresholds for validation failure, hence, these thresholds are set empirically through experiments in which select test cases are modified to simulate validation failure. See the note below regarding *limitations*.

The functional - behavioural model difference, *[f-b]* , is checked to ascertain that

1. the expected number of dumps occur,
2. identically configured chains (with common input data) generate identical lag data (within floating point arithmetic error of the post simulation processor),
3. no overruns occur for integration counts of 10,000 samples or greater (sufficient time between dumps to allow successful readout of active registers and formation of output frames),
4. few outlying sample correlation coefficients greater than $3\sigma_{[f-b]}$ are counted,
5. the standard deviation, $\sigma_{[f-b]}$ about zero mean is scaled by a factor $\sqrt{\frac{m_i}{m_k}\frac{n_k}{n_i}}$ where n_i and n_k are the valid sample counts, m_i and m_k the clock modulus of similarly configured (length and input streams) chains i and k respectively (between test cases or chains of a given test case),
6. no coincidence between dumps of lag indexes exhibiting greater than $3\sigma_{[f-b]}$ (of a given chain) are found,
7. $\sigma_{[f-b]}$ is approximately equal to $\sigma_{[b-b]}$ (suggest within ± 15 %).

Observed deviations from the expected checks above can either be flagged as a test failure (failure to meet points 1,2,3,7) , enumerated or severity estimated in order to participate in a test quality rating (points 4,5,6).

Similarly, the frequency domain equivalent of points 4, 5 and 7 above are also checked with the added condition that no coincidence between dumps of frequency bins exhibiting excessive levels of $[f^F-b^F]$ is observed. Empirical observations suggest that the peak detection threshold in the frequency domain be somewhat higher that that for lag coefficients, say $10\sigma_{[f-b]}$ as opposed to $3\sigma_{[f-b]}$.

A correlation peak should occur at the centre lag in the lag domain in cases where the *X*-input and *Y*-input stream numbers, n_X and n_Y respectively, are equal. The total number of (frequency domain) spectral peaks is n_X+1 in cases where $n_X = n_Y$, otherwise, the number of spectral peaks is the lesser of $1+n_X$ or $1+n_Y$. Deviations from these expected observations constitute a test failure.

The test bench sets the dump enable lines of the unselected dump control signal to *don't care* states which then propagate through to observable output quantities. Hence, the observed behaviour of the functional model is a demonstration of correct X|Y dump control selection (the **X|Y-DS** bit in the **MCSR**). There is no specific test for this operation. Dump control selection is tabulated in (3.3.5) for normal dump tests; both *X* and *Y* dump control are represented.

Limitations: Confidence in validation is limited by lack of a (statistical) model but somewhat ameliorated by the inclusion of *both* lag and frequency domain statistics. Confidence in validation is also limited by the selected reference (test cases 32 and 33) in which only a single lag number (a power of 2) and integration length was tested on a bit exact basis. Although the bit exact test is highly convincing, it is conceivable that differences (in data between correlator models) occurring over an integration can conspire to form the expected sum. A rigorous analysis of the likelihood of such an occurrence has not been carried out.

Test case	*chain #*	*chain configuration*	*dump control*	*number of dumps*	*SE clock modulus*	*samples/dump (functional model)*	*samples/dump (behavioural model)*
2	all	0(*n*)0, *for n*=0..15	X	10	n-a	12,000	10,000
3	1	0(0:3)0	X	16	n-a	12,000	12,000
3	2	2(4:7)2	X	8	n-a	24,000	24,000
3	3	4(8:11)4	X	4	n-a	48,000	48,000
3	4	6(12:15)6	X	2	n-a	96,000	96,000
5	all	1(*n*)1, *for n*=0..15	Y	3	2	12,000	6,000
6	1	1(0:3)1	Y	16	2	12,000	6,000
6	2	3(4:7)3	Y	8	2	24,000	12,000
6	3	5(8:11)5	Y	4	2	48,000	24,000
6	4	7(12:15)7	Y	2	2	96,000	48,000
8	1	0(0)0	X	3	2	40,000	20,000
8	2	1(3)1	X	3	3	40,000	13,333
8	3	2(4)2	X	3	4	40,000	10,000
8	4	3(7)3	X	3	5	40,000	8,000
8	5	4(8)4	X	3	6	40,000	6,666
8	6	5(11)5	X	3	7	40,000	5,714
8	7	6(12)6	X	3	8	40,000	5,000
8	8	7(15)7	X	3	9	40,000	4,444
9	1	0(1)0	Y	3	2	40,000	20,000
9	2	1(2)1	Y	3	3	40,000	13,333
9	3	2(5)2	Y	3	4	40,000	10,000
9	4	3(6)3	Y	3	5	40,000	8,000
9	5	4(9)4	Y	3	6	40,000	6,666
9	6	5(10)5	Y	3	7	40,000	5,714
9	7	6(13)6	Y	3	8	40,000	5,000
9	8	7(14)7	Y	3	9	40,000	4,444
10	1	0(0:1)0	Y	2	2	40,000	20,000
10	2	1(2:3)1	Y	2	3	40,000	13,333
10	3	2(4:5)2	Y	2	4	40,000	10,000
10	4	3(6:7)3	Y	2	5	40,000	8,000
10	5	4(8:9)4	Y	2	6	40,000	6,666
10	6	5(10:11)5	Y	2	7	40,000	5,714
10	7	6(12:13)6	Y	2	8	40,000	5,000

Test case	*chain #*	*chain configuration*	*dump control*	*number of dumps*	*SE clock modulus*	*samples/dump (functional model)*	*samples/dump (behavioural model)*
10	8	7(14:15)7	Y	2	9	40,000	4,444
11, 12	1	0(0:1)0	X	3	n-a	12,000	12,000
11, 12	2	1(2:3)1	X	3	n-a	12,000	12,000
11, 12	3	2(4:5)2	X	3	n-a	12,000	12,000
11, 12	4	3(6:7)3	X	3	n-a	12,000	12,000
11, 12	5	4(8:9)4	X	3	n-a	12,000	12,000
11, 12	6	5(10:11)5	X	3	n-a	12,000	12,000
11, 12	7	6(12:13)6	X	3	n-a	12,000	12,000
11, 12	8	7(14:15)7	X	3	n-a	12,000	12,000
14	1	0(0:3)0	X	3	2	20,000	10,000
14	2	0(4:7)1	X	3	2	20,000	10,000
14	3	1(8:11)0	X	3	2	20,000	10,000
14	4	1(12:15)1	X	3	2	20,000	10,000
21	1	0(0:7)3	X	2	2	24,000	12,000
21	2	4(8:15)7	X	4	n-a	12,000	12,000
22	1	0(0:7)3	X	2	n-a	24,000	24,000
22	2	4(8:15)7	X	2	2	24,000	12,000
23	1	7(0:2)0	X	8	n-a	6,000	6,000
23	2	3(6:7)3	X	4	n-a	12,000	12,000
23	3	4(8)4	X	2	n-a	24,000	24,000
23	4	-	X	1	n-a	48,000	48,000
25	1	1(0:1)1	X	10	n-a	12,000	12,000
25	2	0(2:3)0	X	10	n-a	12,000	12,000
25	3	0(4:5)1	X	10	n-a	12,000	12,000
25	4	1(6:7)0	X	10	n-a	12,000	12,000
25	5	1(8:9)1	X	10	n-a	12,000	12,000
25	6	0(10:11)0	X	10	n-a	12,000	12,000
25	7	0(12:13)1	X	10	n-a	12,000	12,000
25	8	1(14:15)0	X	10	n-a	12,000	12,000
28	1	0(0:2)1	X	2	n-a	10,000	12,000
28	2	1(3:7)3	X	2	n-a	10,000	12,000
28	3	5(8:15)6	X	2	n-a	10,000	12,000

Test case	*chain #*	*chain configuration*	*dump control*	*number of dumps*	*SE clock modulus*	*samples/dump (functional model)*	*samples/dump (behavioural model)*
30	1	1(0)1	X	16	n-a	12,000	12,000
30	2	1(1)0	X	16	n-a	12,000	12,000
30	3	0(2)1	X	16	n-a	12,000	12,000
30	4	0(3)0	X	16	n-a	12,000	12,000
30	5	3(4)3	X	8	n-a	24,000	24,000
30	6	3(5)2	X	8	n-a	24,000	24,000
30	7	2(6)3	X	8	n-a	24,000	24,000
30	8	2(7)2	X	8	n-a	24,000	24,000
30	9	5(8)5	X	4	n-a	48,000	48,000
30	10	5(9)4	X	4	n-a	48,000	48,000
30	11	4(10)5	X	4	n-a	48,000	48,000
30	12	4(11)4	X	4	n-a	48,000	48,000
30	13	7(12)7	X	2	n-a	96,000	96,000
30	14	7(13)6	X	2	n-a	96,000	96,000
30	15	6(14)7	X	2	n-a	96,000	96,000
30	16	6(15)6	X	2	n-a	96,000	96,000
64	1	0(0:7)0	X	2	2	20,000	10,000
64	2	1(8:15)1	X	2	2	20,000	10,000

Table 3.3.5 The above table contains summary of test cases that demonstrate normal dump and readout operation of the chip. Operations that inhibit normal dump operation such as overrun and correlation abort are not considered in this section. Tabulation of the dump control line input demonstrates that both X and Y dumping are represented.

3.3.5 shift enable clock (X|Y-SE_CLK)

Test cases: see section 3.3.4

Description: This operation refers to the application of the the shift enable clock associated with each of the input data streams; all clocks are streamed along with their respective data.

Stimulation: The test bench applies the shift enable clock in one of four modes;

(1) clock line held to zero (no clocking),

(2) clock line held high (data shifted through at maximum rate),

(3) test **SE_CLK** mode 1: modulo 2 (*m=2*) cycle count clock (50 % duty cycle) applied with all input streams,

(4) test **SE_CLK** mode 2: modulo *m=i+2* applied to the lag chain with input *i, i=0 ... 7*.

All clock modes are available for test cases in which *X*-input and *Y*-input stream numbers are the same. In cases where the input stream numbers differ, mode 2 shift enable clocking will result in indeterminate operation and is therefore avoided (although it can, in principle, be used for verification cases).

A number of test cases are run with various shift enable clock modes as tabulated in Table 3.3.5.

Post simulation checks: The number of distinct data samples processed, N_S is related to the shift enable clock (**X|Y-SE_CLK**) modulus of the functional model, m by

$$N_S = \frac{n}{m}$$

where n is the integration count in cycles of the 256 MHz correlator clock. The functional model result is compared to the behavioural model, applying N_S samples input to the latter.

Check that,

(1) $\sigma_{[f-f]}$ is within ± 15 % of $\sigma_{[b-b]}$ (see Section *3.3.4*),

(2) the standard deviation is scaled between chains with differing clock modulus, designated by i and k, and having integration counts n_i and n_k respectively by,

$$(\sigma_{[f-b]})_i = \sqrt{\frac{m_i}{m_k}\frac{n_k}{n_i}}(\sigma_{[f-b]})_k$$

Also see notes under *post simulation checks* and *limitations* in 3.3.4.

3.3.6 input configuration switches

Test cases: 15, 16, 17, 18, 19, 20, 68

Description: Input configuration refers to the combined setting of both the correlator cell input data switches (quad configuration) and of the quad data input switches (chip configuration).

Stimulation: The switch states are defined by the contents of by the **MCSR** which are loaded upon the execution of each test case. Minimal switch state coverage is demonstrated by,

(1) all cell and quad switches placed in all their respective states,

(2) the sixteen *X* and *Y* cell switch pairs within a quad operated independently,

(3) *X* switches of cells in a quad operated independently of the *Y* switches,

(4) the four *X*, *Y* switch pairs of the chip are operated independently,

(5) *X* switches of the chip operated independently of the *Y* switches and

(6) no-lag and no-data switch settings tested.

Table 3.3.6 contains those test cases that are added to the suite of existing tests specifically to increase switch state coverage. A state coverage assessment is made by tabulating test cases against the switch states as defined by [RFS] and assuring that all points above are addressed; the result is presented in Table 3.3.7 and Table 3.3.8.

Post simulation checks: Switch operation is validated when

(1) the contents of the **MCSR** are read following execution of the test case, the switch settings interpreted and compared positively to the applied stimulus,

(2) the expected operation (summarized in table 3.3.6) of the chip is observed and

(3) all normal dump and readout operations are validated by the tests of Section 3.3.4. Also see notes under *post simulation checks* and *limitations* in 3.3.4.

Since the dump interval of test case 68 (also used for verification) is short, the model phase plot is used to detect continuum correlation that is contrary to expectation in the cells of quad 0. The products of quad 2 (previously validated) demonstrate that inspection of the phase plot with a short dump interval can detect unexpected continuum correlation. A bit exact comparison between quads 0 and 3 of similarly configured cells completes the validation of this configuration.

Test case	***Input configuration inpX(c:c)inpY***	***Shift Enable clock modulus***	***number of dumps***	***samples per dump***	***post simulation checks***
15	7(0:3)- 1(4:7)1 1(8:11)1 1(12:15)1	n-a	1	10,000	no lags produced for chain 1; identical lag frames between chains 2 to 4
16	1(0:3)1 1(4:7)1 1(8:11)1 1(12:15)1	2	1	20,000	identical lag frames between chains 1 to 4; DVCOUNT in lag frame approx. 10,000; statistical agreement with case 15
17	1(0:3)1 1(4:7)3 1(8:11)5 1(12:15)7	n-a	2	12,000	count 2 spectral lines; statistical agreement with the behavioural model (see section 3.3.4)
18	1(0:3)1 3(4:7)1 5(8:11)1 7(12:15)1	n-a	2	12,000	count 2 spectral lines; statistical agreement with the behavioural model (see section 3.3.4)
19	0(0:3)0 0(4:7) 2 0(8:11)4 0(12:15)6	n-a	2	12,000	count 1 spectral line; statistical agreement with the behavioural model (see section 3.3.4)
20	0(0:3)0 2(4:7)0 4(8:11)0 6(12:15)0	n-a	2	12,000	count 1 spectral line; statistical agreement with the behavioural model (see section 3.3.4)
68	6(0)0 6(1)1 7(2)0 7(3)1 0(8)0 0(9)1 1(10)0 1(11)1 6(12)0 6(13)1 7(14)0 7(15)1	2	1	1,000	no evidence of correlation in phase plots of cells 0 to 3; no lags generated for cells 4 to 7; bit exact comparison between similarly configured cells of quads 0 and 3.

Table 3.3.6 Test cases added specifically to extend the switch state coverage.

Cell #	*X1*	*X0*	*X-adjacent*	*X-undefined*	*Y1*	*Y0*	*Y-adjacent*	*Y-undefined*
0	4 5 6 15 16 17 18 23 25 30	2 3 8 10 11 14 19 20 21 22 28 32 33 64 68 69 70		9	5 30	2 8 68 69 70	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	9
1	5 30	2 9 68 69 70	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	8	5 25 68	2 9 10 11 30 69 70	3 4 6 14 15 16 17 18 19 20 21 22 23 28 32 33 64	8
2	5 9 10 11 68 69 70	2 25 30	3 4 6 14 15 16 17 18 19 20 21 22 23 28 32 33 64	8	5 9 28 30 69 70	2 23 68	3 4 6 10 11 14 15 16 17 18 19 20 21 22 25 32 33 64	8
3	5 8 28 68 69 70	2 30	3 4 6 10 11 14 15 16 17 18 19 20 21 22 25 32 33 64	9 23	5 6 8 10 11 15 16 17 18 68 69 70	2 3 14 19 20 25 30	4 21 22 23 28 32 33 64	9
4	5 6 15 16 17 18 30	2 3 8 10 11 14 19 20 25 69 70	4 21 22 23 28 32 33 64	9 68	5 30	2 8 69 70	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	9 68
5	5 30	2 9 69 70	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	8 68	5 23 25	2 9 10 11 30 69 70	3 4 6 14 15 16 17 18 19 20 21 22 28 32 33 64	8 68
6	5 9 10 11 23 25 69 70	2 30	3 4 6 14 15 16 17 18 19 20 21 22 28 32 33 64	8 68	5 9 30 69 70	2	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	8 68
7	5 8 69 70	2 30	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	9 68	5 6 8 10 11 14 15 16 17 18 21 22 23 28 69 70	2 3 19 20 25 30 64	4 32 33	9 68
8	5 6 14 15 16 17 18 25 28 30 64	2 3 8 10 11 19 20 21 22 23 68 69 70	4 32 33	9	5 30	2 8 23 68 69 70	3 4 6 10 11 14 15 16 17 18 19 20 21 22 25 28 32 33 64	9
9	5 30	2 9 68 69 70	3 4 6 10 11 14 15 16 17 18 19 20 21 22 25 28 32 33 64	8 23	5 25 68	2 9 10 11 30 69 70	3 4 6 14 15 16 17 18 19 20 21 22 23 28 32 33 64	8
10	5 9 10 11 23 68 69 70	2 25 30	3 4 6 14 15 16 17 18 19 20 21 22 28 32 33 64	8	5 9 30 69 70	2 68	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	8

Cell #	*X1*	*X0*	*X-adjacent*	*X-undefined*	*Y1*	*Y0*	*Y-adjacent*	*Y-undefined*
11	5 8 68 69 70	2 30	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	9	5 6 8 10 11 15 16 17 18 68 69 70	2 3 14 19 20 25 30	4 21 22 23 28 32 33 64	9
12	5 6 14 15 16 17 18 30	2 3 8 10 11 19 20 25 68 69 70	4 21 22 23 28 32 33 64	9	5 30	2 8 68 69 70	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	9
13	5 30	2 9 68 69 70	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	8	5 25 68	2 9 10 11 30 69 70	3 4 6 14 15 16 17 18 19 20 21 22 23 28 32 33 64	8
14	5 9 10 11 25 68 69 70	2 30	3 4 6 14 15 16 17 18 19 20 21 22 23 28 32 33 64	8	5 9 30 69 70	2 68	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	8
15	5 8 68 69 70	2 30	3 4 6 10 11 14 15 16 17 18 19 20 21 22 23 25 28 32 33 64	9	4 5 6 8 10 11 14 15 16 17 18 21 22 64 68 69 70	2 3 19 20 25 28 30 32 33	23	9

Table 3.3.7 The above table shows quad configuration coverage by associating cell data switch states with the test cases that demonstrate them. The numerical entries within body of the table are test case numbers. Verification test cases and those for which the operation under test is independent of the input data configuration are omitted. The notation used within the table header to designate switch state is defined in [RFS].

Quad #	*X1-pri*	*X1-sec*	*X0-pri*	*X0-sec*	*Y1-pri*	*Y1-sec*	*Y0-pri*	*Y0-sec*
Q0	4 5 6 8 9 10 11 16 17 18 25 28 30 69 70	15 23 68	2 3 8 9 10 11 14 19 20 21 22 25 28 30 32 33 64 69 70	68	5 6 8 9 10 11 16 17 18 25 28 30 68 69 70	15	2 3 8 9 10 11 14 19 20 23 25 30 68 69 70	
Q1	6 8 9 10 11 18 23 30 69 70	5 15 16 17 25	3 8 9 10 11 20 30 69 70	2 14 19 25	6 8 9 10 11 17 21 22 23 28 30 69 70	5 14 15 16 18 25	3 8 9 10 11 19 30 69 70	2 20 25 64
Q2	6 8 9 10 11 18 23 28 30 69 70	5 14 15 16 17 25 64 68	3 8 9 10 11 20 21 22 23 30 69 70	2 19 25 68	6 8 9 10 11 17 30 69 70	5 15 16 18 25 68	3 8 9 10 11 19 23 30 69 70	2 14 20 25 68
Q3	6 8 9 10 11 18 30 68 69 70	5 14 15 16 17 25	3 8 9 10 11 20 30 68 69 70	2 19 25	6 8 9 10 11 17 21 22 30 69 70	4 5 14 15 16 18 25 64 68	3 8 9 10 11 19 28 30 69 70	2 20 25 32 33 68

Table 3.3.8 The above table shows chip configuration coverage by associating quad data switch states with the test cases that demonstrate them. The numerical entries within body of the table are test case numbers. Verification test cases and those for which the operation under test is independent of the input data configuration are omitted.

3.3.7 extreme dump rate operation

Test cases: 7

Description: The minimum interval between dump requests that can be accommodated by the correlator chip is approximately 1 μs (below which is beyond operational specification). Extreme dump rate operation involves dump intervals approaching this minimum specification and for which two or more successive failures to generate a lag frame upon dump request occurs as a consequence of overrun. Cells of the chip that are not subject to extreme dump request rates should continue to operate along side those that are and the chip should not exhibit any behaviour that counters expectations.

Stimulation: A dump and readout stimulation is applied in which the time between dump requests associated with one data stream is between 1 and 2 μs. Dump intervals that are not extreme but short enough to invoke a dump overrun and intervals that allow a successful dump and readout without overrun are simultaneously applied to the remaining streams. Cells are not concatenated and all cells of a quad share the same data stream thereby allowing similarity checks between cells of the same configuration.

Post simulation checks: The following behaviour is expected.

(1) The overrun flag will be set in all output frames (except those of the first dump) of cells 0 through 3 and most or all frames associated with cells 4 through 7.

(2) The overrun flag of the **MCSR** will be set after successful execution of the test case.

(3) Cells within quads dumping at an excessive rate (no recovery from overrun) will not produce lag data in all expected frames beyond the first dump.

(4) Cells dumping at normal rates will generate lag data for all expected dumps and these data should be identical (bit exact) between identically configured cells.

(5) **DATA_BIAS** is the difference between the applied (requested) dump interval and $64N_d$ where N_d is approximately $1\mu s$ (integration dead time); both quantities are expressed in number of samples (at 256 *Msample/second*).

Test case	*chain #*	*chain configuration*	*number of dump requests*	*samples/dump (functional model)*
7	1	0(0)0	80	500
7	2	0(1)0	80	500
7	3	0(2)0	80	500
7	4	0(3)0	80	500
7	5	2(4)2	20	2000
7	6	2(5)2	20	2000
7	7	2(6)2	20	2000
7	8	2(7)2	20	2000
7	9	4(8)4	4	10,000
7	10	4(9)4	4	10,000
7	11	4(10)4	4	10,000
7	12	4(11)4	4	10,000
7	13	6(12)6	2	20,000
7	14	6(13)6	2	20,000
7	15	6(14)6	2	20,000
7	16	6(15)6	2	20,000

Table 3.3.9 A test case that exercises the chip at extreme dump rates.

3.3.8 overrun detection and recovery

Test cases: 24, 29, 31, 69, 70

Description: Dump overrun recovery occurs when the overrun bit is cleared and overruns are infrequent enough to accommodate a normal dump and readout in one or more subsequent frames associated with a given cell or chain.

Stimulation: The test cases in table (3.3.10) set up dump rates that invoke occasional dump overruns on some chains; the frequency of overrun occurrence depends on the dump rate associated with the stream of the chain. Test case 29 is similar to 7 of section(3.3.7) but without the extreme rates. Four polarizations with concatenated chains are represented in test case 24, four polarizations with different dump rates per quad in test case 31.

Post simulation checks: Checks are made for the following (expected) behaviour.

(1) The overrun bit in the output frame should clear upon normal dump and readout between overrun occurrences.

(2) Lag frames with set overrun bits contain the expected data (lag and frequency domain plots appear as expected).

(3) The difference between output lag data (of similarly configured chains) with set and reset overrun bits can be reconciled with approximately 1 μs of missing correlation time which occurs during the transfer of data from active to internal storage registers. (See point 6 below).

(4) Cells within chains associated with streams having low dump rates (no overrun) should dump and readout normally along side those experiencing overrun.

(5) Cells dumping at normal rates and cells experiencing overrun with recovery will generate frames with all expected timestamps; output lag data will be identical (bit exact) between respective cells of identically configured chains and similar overrun bit status.

(6) **DATA_BIAS** is the difference between the applied (requested) dump interval and $64N_d$ where N_d is approximately $1\mu s$ (integration dead time); both quantities are expressed in number of samples at 256 *Msample/second* . (See point 3).

(7) The overrun flag, **OVR** of the **MCSR** will be set after successful execution of the test case.

(8) Test cases 69 and 70 are identical except that dump control is set to *X* and *Y* respectively. Dump control is validated if all lag coefficients between the two cases are identical on a bit exact basis.

(9) Test cases 69 and 70 contain eight dump rates that differ between eight non-concatenated pairs of cells and range from extreme as defined in section 3.3.7 to very long (data bias approaches accumulator over flow). Post simulation checks described in 3.3.7 regarding extreme dump rates apply to these two test cases in addition to those of the above points.

Test case	***chain #***	***chain configuration***	***requested number of dumps***	***samples/dump (functional model)***
24	1	1(0:1)1	10	8,000
24	2	0(2:3)0	10	8,000
24	3	0(4:5)1	10	8,000
24	4	1(6:7)0	10	8,000
24	5	1(8:9)1	10	8,000
24	6	0(10:11)0	10	8,000
24	7	0(12:13)1	10	8,000
24	8	1(14:15)0	10	8,000
29	1	0(0)0	16	3,000
29	2	0(1)0	16	3,000
29	3	0(2)0	16	3,000
29	4	0(3)0	16	3,000
29	5	2(4)2	8	6,000
29	6	2(5)2	8	6,000
29	7	2(6)2	8	6,000
29	8	2(7)2	8	6,000
29	9	4(8)4	4	12,000
29	10	4(9)4	4	12,000
29	11	4(10)4	4	12,000
29	12	4(11)4	4	12,000
29	13	6(12)6	2	24,000
29	14	6(13)6	2	24,000
29	15	6(14)6	2	24,000
29	16	6(15)6	2	24,000

Test case	*chain #*	*chain configuration*	*requested number of dumps*	*samples/dump (functional model)*
31	1	1(0)1	32	4,000
31	2	1(1)0	32	4,000
31	3	0(2)1	32	4,000
31	4	0(3)0	32	4,000
31	5	3(4)3	16	8,000
31	6	3(5)2	16	8,000
31	7	2(6)3	16	8,000
31	8	2(7)2	16	8,000
31	9	5(8)5	8	16,000
31	10	5(9)4	8	16,000
31	11	4(10)5	8	16,000
31	12	4(11)4	8	16,000
31	13	7(12)7	4	32,000
31	14	7(13)6	4	32,000
31	15	6(14)7	4	32,000
31	16	6(15)6	4	32,000
69, 70	1	0(0)0	~128	4,000
69, 70	2	0(1)0	~128	4,000
69, 70	3	1(2)1	~64	4,000
69, 70	4	1(3)1	~64	4,000
69, 70	5	2(4)2	32	8,000
69, 70	6	2(5)2	32	8,000
69, 70	7	3(6)3	16	8,000
69, 70	8	3(7)3	16	8,000
69, 70	9	4(8)4	8	16,000
69, 70	10	4(9)4	8	16,000
69, 70	11	5(10)5	4	16,000
69, 70	12	5(11)5	4	16,000
69, 70	13	6(12)6	2	32,000
69, 70	14	6(13)6	2	32,000
69, 70	15	7(14)7	1	32,000
69, 70	16	7(15)7	1	32,000

Table 3.3.10 Test cases that result in dump overruns.

3.3.9 on-line detection and reporting of errors

Test cases: 3, 4, 5, 6, 11, 12, 13, 17, 18, 23, 26, 27, 35, 36, 37, 46, 47, 50, 51, 52, 53, 65, 66

Description: This operation concerns errors that occur in any of the input data streams, dump control lines, timestamps or shift enable clock lines. These errors are reported by the following methods.

(1) Occurrence of a synchronization error on any one of the input lines of the data stream, control, shift enable clock and timestamps set respective bits in the **X|YSTATUS** registers.

(2) *X-Y* offsets of **DUMP_SYNC** and **SCHID_FRAME*** set respective bits in the **X|YSTATUS** registers.

(3) Synchronization errors in the input data stream, **X|Y-DUMP_EN** control, **X|Y-TIMESTAMP** or shift enable clock (**X|Y-SE_CLK**) that affect a particular output frame, set a single bit flag (**X|YSyncerr**) within that frame. An *X-Y* offset of **DUMP_SYNC** or *X-Y* offset of **SCHID_FRAME*** (**SCHID_SYNC**) are *not* reported in the output frame.

(4) The **SyncER** bit of the **MCSR** is set if any of the **X|Ysyncerr** bits in any of the output frames have been set. Detection of input errors during *off-line testing* does not set either the **X|YSyncerr** or **SyncER** bit as output frames are not generated during such tests.

Synchronization checks for the data streams are performed by the chip following each **XY-SCHID_FRAME*** at which point all the lines must be at prescribed levels to satisfy a null error detection; the **X|Y-DUMP_EN** is similarly checked upon occurrence of the **X|Y-DUMP_SYNC** signal.

Stimulation: The test bench is capable of applying input synchronization errors to the input data stream and **SE_CLK** lines by specifying, for each test case,

(1) a rate of occurrence (in terms of **X|Y-SCHID_FRAME*** count between errors) which is applied to all lines (not simultaneously), or

(2) data valid lines to maintain in a generally reset state (simulating loss of connectivity).

The latter method is preferred as it allows for tester control over which register bits are to report errors. The structure of the test case files allow modification of the (synchronizing) serial bit fill patterns of the **X|Y-DUMP_EN** and **X|Y-TIMESTAMP**, thereby providing an additional method of simulating input synchronization errors in these quantities.

Test cases that specifically target the detection and reporting of input errors are tabulated in Table 3.3.11. Test cases 50, 51, 65 and 66 generally reset select data valid lines (point (2) above) as described in Table 3.3.12. Test cases 65 and 66 invoke a test bench facility to set data valid lines during a normally reset interval at the start of a **SCHID** frame thereby setting corresponding **X|Y-DVALID** bits of the **X|YSTATUS** register.

An offset between **X-SCHID_FRAME*** and **Y-SCHID_FRAME*** can be specified for each test case. If the offset exceeds the input FIFO buffer length, the **X-SCHID_FRAME*** and **Y-SCHID_FRAME*** will be misaligned at the buffer output and, as a result, the **SCHID_SYNC** bit in the **XYSTATUS** register will be set. This condition should also set the **XYsyncerr** bit in the output frame and the **SyncER** bit in the **MCSR**. The current test bench is incapable of providing offsets that exceed the buffer length of the latest chip model without also creating an *X-Y* offset of **DUMP_SYNC** as a side effect. Test cases 52 and 53 applies such an offset for *X* both leading and lagging *Y*. All **DESSR** bits will be set as a consequence of the aforementioned test bench limitation.

Post simulation checks:

(1) **X|YSyncerr** bits set in a frame have corresponding bits set in the **X|YSTATUS** register. **X|YSyncerr** bit settings are not necessarily identical for all cells within a chain ([RFS] does not address this issue).

(2) **X|YSTATUS** lines are set consistent with stimulation (see table 3.3.13).

(3) **TvER** is not set (since **TVEN** is not set).

(4) **SyncER** is set when either of the **X|YSyncerr** bits are set in any of the frames; these bits are checked after each test as part of the general post simulation analysis and a warning issued if any are unexpectedly set.

Note: Checks of points 1, 3 and 4 above are performed in all test cases except where noted.

test case	*active[8] SE_CLK*	*active[9] data streams*	*SCHID_FRAME* length [samples]*	*number of SCHID_FRAME* between errors*	*X-Y offset on DUMP_SYNC*	*TIMESTAMP fill synch. (normal: 2`b01)[10]*	*DUMP_EN fill synch. (normal: 2`b01)*
3	X: 0, 2, 4, 6 Y: 0, 2, 4, 6	X: 0, 2, 4, 6 Y: 0, 2, 4, 6	512	n-a	yes	2`b01	2`b01
4	X: 1 Y: 1	X: 1 Y: 1	512	15	no	2`b01	2`b01
5	X: 1 Y: 1	X: 1 Y: 1	512	n-a	yes	2`b01	2`b01
6	X: 1,3,5,7 Y: 1,3,5,7	X: 1,3,5,7 Y: 1,3,5,7	512	130	no	2`b01	2`b01
11	X: 0,1,2,3,4, 5,6,7 Y: 0,1,2,3,4, 5,6,7	X: 0,1,2,3,4,5,6,7 Y: 0,1,2,3,4,5,6,7	1024	22	yes	2`b01	2`b01
12	0,1,2,3,4,5,6,7	0,1,2,3,4,5,6,7	512	n-a	no	2`b01	2`b11
13	0,1,2,3	0	512	n-a	no	2`b01	2`b01
17	X: 1 Y: 1,3,5,7	X: 1 Y: 1,3,5,7	512	n-a	no	2'b01	2'b01
18	X: 1,3,5,7 Y: 1	X: 1,3,5,7 Y: 1	512	n-a	no	2'b01	2'b01
23	X: 0,3,4,5,7 Y:0,3,4,5,7	X: 0,3,4,5,7 Y:0,3,4,5,7	512	n-a	yes	2`b01	2`b01
26	0,1,2,3,4,5,6,7	0	512	n-a	no	2`b11	2`b01
27	X: 0,1,2,3,4,5,6,7 Y: 0,2,4,6	X: 0,1,2,3,4,5,6,7 Y: 0,1,2,3,4,5,6,7	512	n-a	no	2`b01	2`b01
35	X: 0, 2, 4, 6 Y: 0, 2, 4, 6	X: 0, 1 Y: 0, 1	512	n-a	no	2`b01	2`b01
36	X: 0,1,2,3,4, 5,6,7 Y: 0,1,2,3,4, 5,6,7	X: 0,1,2,3,4,5,6,7 Y: 0,1,2,3,4,5,6,7	512	n-a	no	2`b11	2`b11

8 a non-active SE_CLK is generally reset.
9 active refers to streams for which data and dump commands are generated by the test bench.
10 the test bench sets a normal fill on the non-dumping timestamp.

test case	*active SE_CLK*	*active data streams*	*SCHID_FRAME* length [samples]*	*number of SCHID_FRAME* between errors*	*X-Y offset on DUMP_SYNC*	*TIMESTAMP fill synch. (normal: 2`b01)*	*DUMP_EN fill synch. (normal: 2`b01)*
37	X: 0,1,2,3,4, 5,6,7 Y: 0,1,2,3,4, 5,6,7	X: 0 Y: 0	512	n-a	yes	2`b01	2`b01
46	X: 0,1,2,3,4, 5,6,7 Y: 0,1,2,3,4, 5,6,7	X: 0,1,2,3,4, 5,6,7 Y: 0,1,2,3,4, 5,6,7	512	n-a	no	2`b01	2`b11
47	X: 0,1,2,3,4, 5,6,7 Y: 0,1,2,3,4, 5,6,7	X: 0,1,2,3,4, 5,6,7 Y: 0,1,2,3,4, 5,6,7	512	n-a	no	2`b01	2`b11

Table 3.3.11 Test cases that apply specific input error conditions.

test case	*X-SDATA [stream][bit]*	*X-PHASE [stream][bit]*	*X-SE_CLK [stream]*	*X-DVALID [stream]*	*Y-SDATA [stream][bit]*	*Y-PHASE [stream][bit]*	*Y-SE_CLK [stream]*	*Y-DVALID [stream]*
50	[0:7][0,2]	[0:7][0,2]	n-a	n-a	[0:7][1,3]	[0:7][1,3]	n-a	n-a
51	[0:7][1,3]	[0:7][1,3]	n-a	n-a	[0:7][0,2]	[0:7][0,2]	n-a	n-a
65	[0,2,4,6][0:3]	[0,2,4,6][0:3]	[0,2,4,6]	[0,2,4,6]	[1,3,5,7][0:3]	[1,3,5,7][0:3]	[1,3,5,7]	[1,3,5,7]
66	[1,3,5,7][0:3]	[1,3,5,7][0:3]	[1,3,5,7]	[1,3,5,7]	[0,2,4,6][0:3]	[0,2,4,6][0:3]	[0,2,4,6]	[0,2,4,6]

*Table 3.3.12: Quantities for which the corresponding data valid lines are generally reset. In the case of **X**\|**Y-DVALID** (the data valid lines themselves) a set state is forced where a reset state is normally expected.*

test case	*XSTATUS error reports*	*YSTATUS error reports*	*SyncER*	*notes*
3	SE_CLK error on inputs 1, 3, 5, 7	SE_CLK error on inputs 1, 3, 5, 7	0	output frames do not involve input streams 1,3,5,7
4	input errors reported	input data errors reported	1	by register bits corresponding to all streams; does not include DVALID
5	DUMP_SYNC, (SE_CLK error on inputs 0, 2, 3 ,4 , 5, 6, 7)	DUMP_SYNC, (SE_CLK error on inputs 0, 2, 3, 4, 5, 6, 7)	0	out put frames do not involve input streams 0, 2, 3, 4, 5, 6, 7
6	input errors reported	input errors reported	1	by register bits corresponding to all streams; does not include DVALID
11	occasional input data errors reported	occasional input data errors reported	1	errors generated infrequently; check for expected reporting in **X|YSyncerr** of the output frame
12	DUMP_EN on inputs 0, 2, 4, 6	n-a	1	bad fill pattern applied to dumps associated with even inputs
13	SE_CLK error on inputs 4, 5, 6, 7	SE_CLK on error inputs 4, 5, 6, 7	0	out put frames do not involve input streams 4, 5, 6, 7
17	SE_CLK error on inputs 0, 2, 3 ,4 , 5, 6, 7	SE_CLK on error on inputs 0, 2, 4, 6	0	
18	SE_CLK error on inputs 0, 2, 4, 6	SE_CLK error on inputs 0, 2, 3 ,4 , 5, 6, 7	0	
23	DUMP_SYNC; SE_CLK error on inputs 1, 2, 6	DUMP_SYNC; SE_CLK error on inputs 1, 2, 6	0	contains cells with undefined connections
26	TIMESTAMP		1	
27	n-a	SE_CLK error on input 1, 3, 5, 7	1	indeterminate operation on chains involving odd numbered streams
35	SE_CLK error on inputs 1, 3, 5, 7	SE_CLK error on inputs 1, 3, 5, 7	1	only 0x0 products will generate valid correlation; errors on odd input **XY-SE_CLK**
36	n-a	TIMESTAMP, DUMP_EN lines 1, 3, 5, 7	1	bad fill pattern applied to dumps associated with odd streams
37	DUMP_SYNC	DUMP_SYNC	0	
46	DUMP_EN lines 1, 3, 5, 7	n-a	1	bad fill pattern applied to dumps associated with odd streams
47	n-a	DUMP_EN on inputs 0, 2, 4, 6	1	bad fill pattern applied to dumps associated with even streams
50	SDATA, PHASE error on even numbered bits on all streams	SDATA, PHASE error on odd numbered bits on all streams	1	
51	SDATA, PHASE error on odd numbered bits on all streams	SDATA, PHASE error on even numbered bits on all streams	1	
65	SDATA, PHASE, SE_CLK, DVALID error on even numbered streams	SDATA, PHASE, SE_CLK, DVALID error on odd numbered streams	1	all bits of a word
66	SDATA, PHASE, SE_CLK, DVALID error on odd numbered streams	SDATA, PHASE, SE_CLK, DVALID error on even numbered streams	1	all bits of a word

Table 3.3.13 Expected states of error report bits are tabulated for the test cases of Table 3.3.11 and Table Table 3.3.12. The output frame error status bits, **X|YSyncerr** *are reflected in the report of the MCSR status bit,* **SyncER**.

3.3.10 X-Y dump enable offset detection

Test cases: 3, 5, 11, 23, 37, 48, 49

Description: Bits within the **DESSR** register correspond to each of the sixteen cells and are set when the **Y-DUMP_EN** input of the respective cell is not properly synchronized (offset in time) to the **X-DUMP_EN** on the same cell. These registers are cleared upon read.

Stimulation: An *X-Y* offset can be applied to all of the **DUMP_EN** lines simultaneously. The test bench reads the **DESSR** contents at the end of a run and writes the result to the MCB register file. Test cases 3, 5, 11 and 37 will result in all bits of the **DESSR** being set. An *X-Y* offset is applied to all **DUMP_EN** in test case 23 but this case is configured so that only cells 6, 7 and 8 will generate output frames. An offset can be applied to each cell independently by configuring the chip such that slightly different dump intervals are associated with the *X* and *Y* inputs of a given cell. Moreover, the test bench is able to maintain specified **DUMP_EN** lines in a reset state, thereby simulating loss of connectivity. Table 3.3.14 lists test cases pertaining to the validation of *X-Y* dump enable offset detection.

Post simulation check: Assure that bits of the **DESSR** and **DUMP_SYNC** within the **STATUS** registers are set according to Table 3.3.14. All bits of the **DESSR** are checked after each test as part of general post simulation checks and a warning issued if any are unexpectedly set. Where applicable, check that output lag data between identically configured cells compares on bit exact basis.

test case	*chain inpX(c:c)inpY*	*dump control*	*X-input samples/dump*	*Y-input samples/dump*	*X-Y offset on DUMP_SYNC*	*cells flagged in DESSR*	*DUMP_SYNC set in X,YSTATUS*
3	0(0:3)0	X	12,000	12,000	yes	0:3	yes
3	2(4:7)2	X	24,000	24,000	yes	4:7	yes
3	4(8:11)4	X	48,000	48,000	yes	8:11	yes
3	6(12:15)6	X	96,000	96,000	yes	12:15	yes
5	1(*n*)1, *for n*=0...15	Y	12,000	12,000	yes	0:15	yes
11	all pairs and input streams	X	12,000	12,000	yes	0:15	yes
23	7(0:2)0	X	12,000	12,000	yes	0:2	yes
23	3(6:7)3	X	12,000	12,000	yes	6:7	yes
23	4(8)4	X	12,000	12,000	yes	8	yes
37	0(0:15)0	X	10,000	10,000	yes	0:15	yes
48	0(0)0	X	500	500	no	x	no
48	0(1)1	X	500	1000	no	1	no
48	1(2)1	X	1000	1000	no	x	no
48	1(3)0	X	1000	500	no	3	no
48	0(4)0	X	500	500	no	x	no

test case	*chain inpX(c:c)inpY*	*dump control*	*X-input samples/dump*	*Y-input samples/dump*	*X-Y offset on DUMP_SYNC*	*cells flagged in DESSR*	*DUMP_SYNC set in X,YSTATUS*
48	0(5)1	X	500	1000	no	5	no
48	1(6)1	X	1000	1000	no	x	no
48	1(7)0	X	1000	500	no	7	no
48	0(8)0	X	500	500	no	x	no
48	0(9)1	X	500	1000	no	9	no
48	1(10)1	X	1000	1000	no	x	no
48	1(11)0	X	1000	500	no	11	no
48	0(12)0	X	500	500	no	x	no
48	0(13)1	X	500	1000	no	13	no
48	1(14)1	X	1000	1000	no	x	no
48	1(15)0	X	1000	500	no	15	no
49	1(0)0	X	1000	500	no	0	no
49	1(1)1	X	1000	1000	no	x	no
49	0(2)1	X	500	1000	no	2	no
49	0(3)0	X	500	500	no	x	no
49	1(4)0	X	1000	500	no	4	no
49	1(5)1	X	1000	1000	no	x	no
49	0(6)1	X	500	1000	no	6	no
49	0(7)0	X	500	500	no	x	no
49	1(8)0	X	1000	500	no	8	no
49	1(9)1	X	1000	1000	no	x	no
49	0(10)1	X	500	1000	no	10	no
49	0(11)0	X	500	500	no	x	no
49	1(12)0	X	1000	500	no	12	no
49	1(13)1	X	1000	1000	no	x	no
49	0(14)1	X	500	1000	no	14	no
49	0(15)0	X	500	500	no	x	no

Table 3.3.14 Test cases pertaining to the validation of X-Y dump enable offset detection. An "x" under the DESSR column indicates that all bits are reset for cells within the corresponding chain. "X-Y offset on DUMP_SYNC" refers to a feature of the test bench that applies an X-Y offset to all DUMP_EN lines simultaneously.

3.3.11 off-line detection and reporting of synchronization errors

Test cases: 38, 39, 40

Description: Setting test vector enable, **TVEN**, in the **MCSR**, enables generation of the test vector sequence and subsequent detection of input synchronization errors that may occur in the input data streams, dump control lines, timestamps and shift enable clock lines. These errors are reported in the **X|YSTATUS** registers. Occurrence of an error also sets the **TvER** bit in the **MCSR**. No output lag frames are generated and the **SyncER** bit of the **MCSR** is not affected.

Stimulation: The test vector enable bit, **TVEN** can be set by the test bench and the time between error generation specified in terms of number of samples. Although **CEN** is set within the test case, the chip will reset this bit internally according to [RFS].

Post simulation check:

(1) Errors are detected when generated by the test bench.

(2) Errors are reported in the **X|YSTATUS** registers and the **TvER** bit in the **MCSR** is set.

(3) **SyncER** is not set upon the detection of errors when **TVEN** is set.

test case	*SCHID_FRAME* length [samples]*	*enable error generation*	*number of samples between errors*	*test length in samples*
38	512	no	n-a	10,000
39	512	yes	100	10,000
40	512	yes	2000	10,000

Table 3.3.15 Test cases that run off-line tests with the chip test vector generator. Synchronization errors are generated within the test bench for the purpose of validation.

3.3.12 MCSR status flags

Test cases: all

Description: The **MCSR** status bits **SynchER**, **TvER**, **OVR**, **AOV** are set by conditions described in sections 3.3.9, 3.3.11, 3.3.8 and 3.3.3 respectively. They are reset upon read.

Stimulation: The test bench performs a read as part of the test initialization and once again when the test is complete. The current test bench does not provide a facility to read and consequently reset the **MCSR** during a test run.

Post simulation checks: The **MCSR** status bits are checked after each test and a warning issued if any are unexpectedly set.

3.3.13 correlation holdoff

Test case: 42

Description: Correlation can be delayed by maintaining the **X|Y-DUMP_EN** line set after a dump sequence (approximately 1 μs in total).

Stimulation: The test bench allows for test case specification of the dump sequence that is applied to each **DUMP_EN** line. Correlation holdoff is simulated by extending the number of bits held high at the end of the dump sequence. The dump sequence is applied serially at the 128 MHz clock rate and has its length in bits initialized to 72 within the test bench; this is extended to a total of 312 bits within test case 42 in order to exceed the 1 μs dump time. The holdoff, in number of bits, is incremented by one for each successive **Y-DUMP_EN** line, using **Y-DUMP_EN[0]** as the base (see Table 3.3.16).

Post simulation check: The bias expressed in samples for both cells of each chain should decrease with holdoff according to *bias(n)=bias(1)-2(n-1)* where *n* is the chain number and *bias(1)* the bias associated with each cell of *chain 1.* The correlation holdoff applied to each chain is listed in Table 3.3.16.

test case	*chain #*	*chain*	*bias decrease (number of samples)*
42	1	0(0:1)0	0
42	2	1(2:3)1	2
42	3	2(4:5)2	4
42	4	3(6:7)3	6
42	5	4(8:9)4	8
42	6	5(10:11)5	10
42	7	6(12:13)6	12
42	8	7(14:15)7	14

*Table 3.3.16 The correlation holdoff is manifested by a decrease of 2 samples in observed **DATA_BIAS** for each bit added to the correlation holdoff. The base holdoff is taken to be that of **DUMP_EN[0]** (associated with stream 0).*

3.3.14 frame abort

Test cases: 43

Description: A **FRAME_ABORT*** sequence applied to **LTA** interface discards the current frame; correlation is unaffected.

Stimulation: The test bench has a built in feature that, when set, will initiate a **FRAME_ABORT*** on alternate dumps of every cell. A different dump rate is applied to 4 different input streams of test case 43; multiple dumps are requested with a frame abort every second dump.

Post simulation checks: Assure that alternate dumps of all cells within a chain have been discarded.

3.3.15 dump command capture

Test cases: 41, 42, 46, 47

Description: The 3 bit dump command, the 8 bit recirculation block (**RECIRC_BLK-X**, **RECIRC_BLK-Y**) and 16 bit phase bin are associated with, and embedded in, the individual dump enable control lines. The discard and dump command, 3`b100 is the only such data that is used by the correlator chip. The others are captured and routed to appropriate fields in the output lag frame.

Stimulation: The discard and dump command is applied to the chip during the test initialization to assure that the chip begins the test in a "clean" state. Capture and routing of the other four recognized (dump and discard is excluded) and three unused dump commands are tested by embedding them in the serial dump sequence applied to each **X|Y-DUMP_EN** line, together with the phase bin, recirculation block and correlation holdoff. A combination of these quantities that exercise a representative combination of bits of the dump sequence is tabulated in Table 3.3.17. A different dump control sequence is applied to each **DUMP_EN** line. The dump command, phase bin and correlation holdoff are selected on the basis of dump control. The recirculation block, **RECIRC_BLK-X|Y** is read from the respective **X|Y-DUMP_EN** line. The recirculation block corresponding to the inactive dump control is set to zero.

Post simulation checks: Assure that the output frame data is consistent with Table 3.3.17 and that all cells of a chain exhibit identical dump command, recirculation block and phase bin data.

test case	*chain inpX(c:c)inpY*	*dump control*	*dump command*	*phase (LTA) bin*	*RECIRC_BLK-X*	*RECIRC_BLK-Y*
41	0(0:1)0	X	3`b000	21840	80	0
41	1(2:3)1	X	3`b001	21841	81	0
41	2(4:5)2	X	3`b010	21842	82	0
41	3(6:7)3	X	3`b011	21843	83	0
41	4(8:9)4	X	3`b110	21844	84	0
41	5(10:11)5	X	3`b101	21845 16`b0101010101010101	85 (8`b01010101)	0
41	6(12:13)6	X	3`b011	21856	86	0
41	7(14:15)7	X	3`b111	21847	87	0
42	0(0:1)0	Y	3`b011	43690 16`b1010101010101010	0	80
42	1(2:3)1	Y	3`b101	43691	0	81
42	2(4:5)2	Y	3`b110	43692	0	82
42	3(6:7)3	Y	3`b010	43693	0	83
42	4(8:9)4	Y	3`b111	43694	0	84
42	5(10:11)5	Y	3`b010	43695	0	85 (8`b01010101)
42	6(12:13)6	Y	3`b001	43696	0	86
42	7(14:15)7	Y	3`b000	43697	0	87
46	0(0:1)0	X	3`b000	21840	170 (8`b10101010)	0
46	1(2:3)1	X	3`b001	21841	171	0
46	2(4:5)2	X	3`b010	21842	172	0
46	3(6:7)3	X	3`b011	21843	173	0
46	4(8:9)4	X	3`b010	21844	174	0
46	5(10:11)5	X	3`b010	21845 16`b0101010101010101	175	0
46	6(12:13)6	X	3`b011	21856	176	0
46	7(14:15)7	X	3`b011	21847	177	0
47	0(0:1)0	Y	3`b000	21840	0	170 (8`b10101010)
47	1(2:3)1	Y	3`b001	21841	0	171
47	2(4:5)2	Y	3`b010	21842	0	172
47	3(6:7)3	Y	3`b011	21843	0	173

test case	*chain inpX(c:c)inpY*	*dump control*	*dump command*	*phase (LTA) bin*	*RECIRC_BLK-X*	*RECIRC_BLK-Y*
47	4(8:9)4	Y	3`b010	21844	0	174
47	5(10:11)5	Y	3`b010	21845 16`b0101010101010101	0	175
47	6(12:13)6	Y	3`b011	21856	0	176
47	7(14:15)7	Y	3`b011	21847	0	177

*Table 3.3.17 Test cases in which specific dump control sequence patterns are applied as stimulus. The phase bin is selected on the basis of the dump control. The **R**ECIRC_BLK corresponding to the inactive dump control is set to zero in the current version of the test bench.*

3.3.16 timestamp capture

Test cases: 17, 18, 19, 20

Description: Two 32 bit words that comprise the timestamp are captured from the timestamp line (**X|Y-TIMESTAMP**). The **X|Y-TIMESTAMP** is synchronized to **X|Y-DUMP_SYNC** and *X-Y* selection is made according to the dump control.

Stimulation: The test bench applies a timestamp that starts from a base value set in the test case and increments for each dump. The default for most tests uses two different, easily recognizable values for **X-TIMESTAMP** and **Y-TIMESTAMP**. Successful timestamp capture is further demonstrated in the test cases of Table 3.3.18 where the first 32 bit word is set to an alternating bit pattern and the second word set to the complementary pattern.

Post simulation checks: Output lag frames are sorted and concatenated on the basis of the timestamp so that timestamp capture is demonstrated to some degree by successful execution of post simulation analysis scripts. The test cases of Table 3.3.18 are *specifically* checked for consistency between applied and captured timestamps.

test case	*dump control*	*number of dumps*	*X-TIMESTAMP: word_0*	*X-TIMESTAMP: word_1*	*Y-TIMESTAMP: word_0*	*Y-TIMESTAMP: word_1*
17	X	2	h`AA AAA AAA 2 863 311 530	h `55 555 555 1 431 655 765	222 333 444	555 666 777
18	X	2	h`55 555 555 1 431 655 765	h`AA AAA AAA 2 863 311 530	222 333 444	555 666 777
19	Y	2	222 333 444	555 666 777	h`AA AAA AAA 2 863 311 530	h`55 555 555 1 431 655 765
20	Y	2	222 333 444	555 666 777	h`55 555 555 1 431 655 765	h`AA AAA AAA 2 863 311 530

Table 3.3.18 Test cases that specifically check for correct **X|Y-TIMESTAMP**. *The timestamp selected by dump control is assigned alternating patterns ...1010... and ...0101... for all 32 bits of each word respectively.*

3.3.17 data stream ID capture

Test cases: 17, 18, 19, 20

Description: The station ID, baseband ID and sub-band ID identify the data source and are associated with, and embedded in, the individual data streams.

Stimulation: The 8 bit station IDs, **SID-X**, **SID-Y** are assigned fixed values within each test case and applied to the input data stream header. The 5 bit sub-band IDs, **SBID-X**, **SBID-Y** are each assigned a base value in each test case which, when added to the input number (0 to 7) form the value ultimately applied to the input stream header. The station ID and sub-band ID base are varied between test cases of Table 3.3.19. Baseband IDs, **BBID-X** and **BBID-Y** are generated within the test bench using the associated input stream and test case numbers. The value applied to the model is given by $BBID = N_S + (N_{TC} \, mod \, 8)$ where N_S is the stream number, N_{TC} the test case number and *mod ,* the modulo operator.

Post simulation checks: Assure that

(1) the test values shown in Table 3.3.19 match corresponding values in the output lag frame,

(2) all cells of a chain have identical **SID**, **BBID** and **SBID**,

(3) the captured value for **BBID** is $N_S + (N_{TC} \, mod \, 8)$ as described above.

The checks of points 2 and 3 above are performed in all test cases unless otherwise noted.

test case	***Input configuration inpX(c:c)inpY***	***SID-X***	***SID-Y***	***base SBID-X***	***base SBID-Y***
17	1(0:3)1 1(4:7)3 1(8:11)5 1(12:15)7	85 (8`b01010101)	170 (8`b10101010)	21 (5`b10101)	10 (5`b01010)
18	1(0:3)1 3(4:7)1 5(8:11)1 7(12:15)1	85 (8`b01010101)	170 (8`b10101010)	21 (5`b10101)	10 (5`b01010)
19	0(0:3)0 0(4:7) 2 0(8:11)4 0(12:15)6	170 (8`b10101010)	85 (8`b01010101)	10 (5`b01010)	21 (5`b10101)
20	0(0:3)0 2(4:7)0 4(8:11)0 6(12:15)0	170 (8`b10101010)	85 (8`b01010101)	10 (5`b01010)	21 (5`b10101)

Table 3.3.19 Test cases in which specific data stream ID patterns are applied as stimulus. The base value in the case of **SBID-X|Y** *is that to which the input stream number is added.*

3.3.18 enable phase rotation

Test case: 44

Description: Phase rotation is disabled in all correlator cells if the **Ph**ase rotation **EN**able bit, **PhEN**, in the **MCSR** is reset. The correlator operates with the phase fixed at zero.

Stimulation: Phase rotation is disabled in test case 44 by resetting bit **PhEN** in the **MCSR**.

Post simulation checks: Output frames are generated but correlation is not detectable in plots of the resulting lag data. The resultant lag data is real; the imaginary part within the output frame is identical to the **DATA_BIAS**. Identically configured cells (sharing the same input streams) should generate identical (bit exact) lag data.

3.3.19 enable correlation

Test case : 45

Description: Correlation is disabled for all correlator cells if the **C**orrelation **EN**able bit, **CEN**, in the **MCSR** is reset. The chip does not generate output frames although most internal operations on data continue normally.

Stimulation: Correlation is disabled by resetting bit **CEN** in the **MCSR**.

Post Simulation check: Output frames are not generated by test case 45.

3.3.20 verification

Test case: 54, 55

Description: These test cases simultaneously exercise a number of chip operations and are intended for model verification. These operations have been individually validated by previous test cases, hence test cases 54 and 55 are not required for functional model validation. However, a validation run is executed to confirm that these test cases operate as expected.

Principal operations addressed with test case 54:

(1) dump overrun (at least once for each cell),

(2) Y dump control,

(3) multiple dumps with multiple dump rates,

(4) 2 polarization pairs,

(5) large cell data switch state space,

(6) modulo shift enable clock,

(7) all inputs active,

(8) correlation holdoff,

(9) no input synchronization errors applied,

(10) normal dump and readout,

(11) *X-Y* offset on **DUMP_SYNC** (all **DESSR** and **X,Y-DUMP_SYNC** bits set),

(12) negative **Y_SCHID_FRAME*** offset.

Principal operations addressed with test case 55:

(1) mixed polarization,

(2) dump overrun on all cells,

(3) very short integration,

(4) X dump control,

(5) *no-data* input configuration on some cells,

(6) large quad data switch state space,

(7) **DUMP_EN** synchronization error applied,

(8) positive **Y_SCHID_FRAME*** offset,

(9) correlation holdoff,

(10) no *X*-*Y* offsets on **DUMP_SYNC.**

3.3.21 autocorrelation mode

Test cases: 56, 57, 58, 59, 60, 61, 62, 63

Description: The autocorrelation function of the correlator chip is tested.

Stimulation: The chip is placed in autocorrelation mode; one test case for each of the eight *X* input data streams. Each of the *n* input streams to be autocorrelated is applied to both ends of the lag chain external to the chip via two separate input ports (**X-SDATA[*n*][0:3]**, **X-PHASE[*n*][0:3]**). The test bench applies a single sample delay between these inputs which in turn shifts the autocorrelation peak by one sample and creates an easily discernible phase slope with respect to frequency in the frequency domain. Test vector set DS4 is applied.

Post simulation checks: Examine the lag data for an autocorrelation peak (lag domain), phase slope with respect to frequency and magnitude peak count consistent with Table 3.3.20 in the frequency domain.

test case	***autocorrelation input stream***	***number of magnitude peaks in the frequency domain***
56	0	1
57	1	2
58	2	3
59	3	4
60	4	5
61	5	6
62	6	7
63	7	8

Table 3.3.20 Autocorrelation test cases. Operation is validated by examining the frequency domain magnitude and phase.

4 Test Cases

This section contains the source code for the proposed test cases described earlier. Each test case has been executed with a prototype version of the test bench and correlator chip functional model, primarily to assure that the test case code is operational. Detailed analysis of the resulting output has not yet been carried out. Each of the following subsections contains a chip configuration diagram followed by the source listing. The diagram is a screen snapshot of output from the MS-Windows® based test analysis software [LauTAS] and is created from the mcb_register file generated by the test bench for the test case under consideration. Chains of concatenated cells are readily identified by contiguous highlighting in yellow; primary data stream connections are shown in red and secondary connections in blue. The listings are, for the most part, unmodified from the source code files that were submitted to the test bench; minor alterations in white space and page breaks were applied for clarity after insertion within this document.

Note: The test case source listing trailer lines

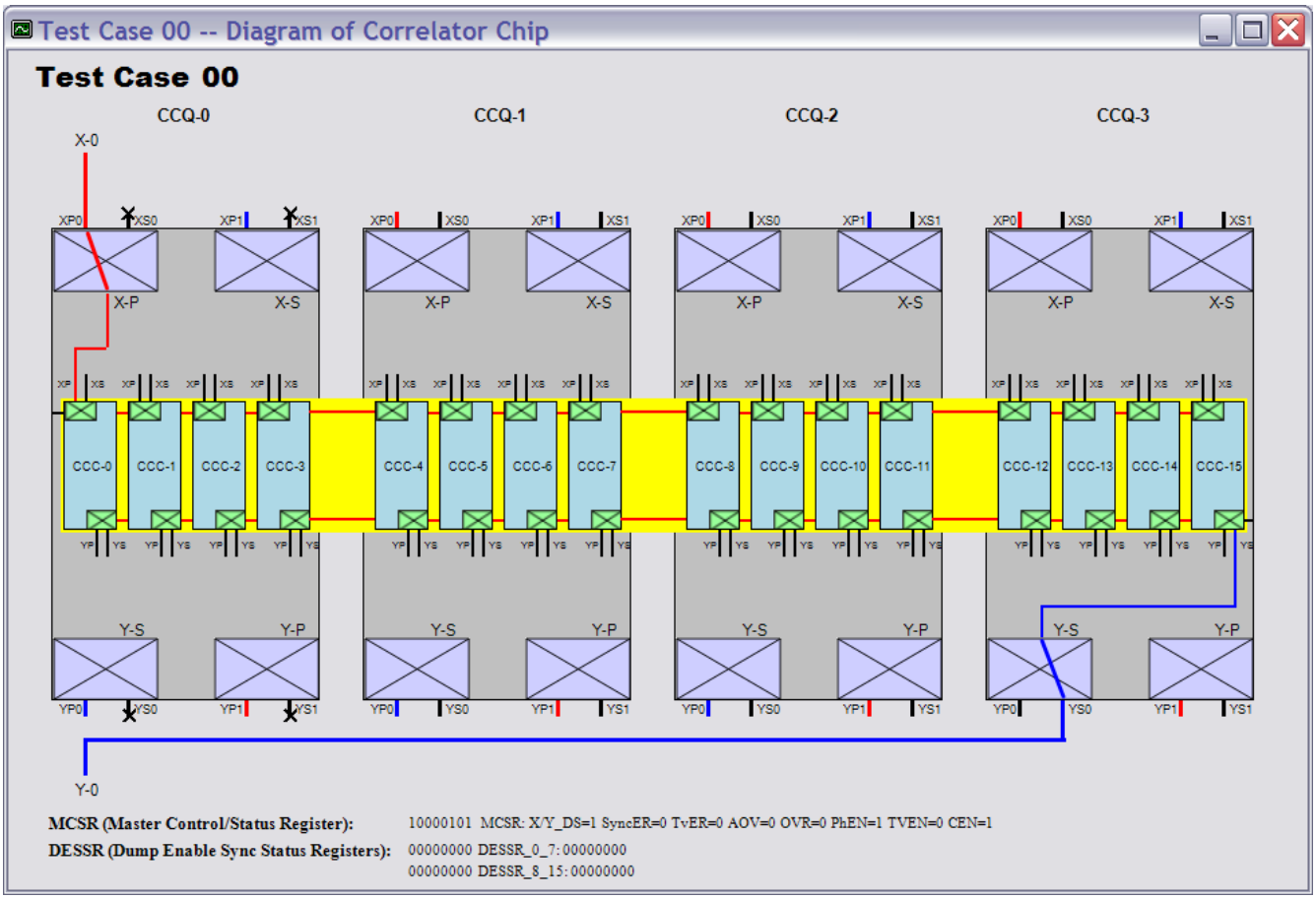
```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

may have been omitted on many test cases in order to save pages.

4.1 Test case 0




```
// ********************************************************************************************************************
// File name : tc00_apr05-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// ---                    Oct 28, 2003    ---              creation
// tc00_mar16-04_v1.v     Mar 16, 2004    1 - PRELIM       clean up format, start revision history
// tc00_jul20-04_v2.v     Jul 20, 2004    2                modify for new test bench
// tc00_aug30-04_v2.v     Aug 30, 2004    2                extend dump interval, single dump
// tc00_aug31-04_v2.v     Aug 31, 2004    2                edit description for 23 bit accumulator
// tc00_apr05-05_v3.v     Apr 05, 2005    3                split accumulator and short integration; add
//                                                         verification mode option
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 0;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 0:
# ver: v3,05apr05
# Synopsis: CONCATENATE ALL, X DUMP, INPUT 0, SPLIT ACCUMULATOR.
# Description:
#              Purpose of Test:   Test the accumulator bits; all bits are toggled using
#                                 split accumulator mode.
#
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 660-0
#              Distinct samp/dump:   660-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  split accumulator bias 5416192
#              Additional details:   valid data reset; 660 sample bias is 42240.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b10;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------
```

```
//
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

dts_len_mem[tc]= 312;  // increase size of dts_len to 312 bits total.

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to cover integration dead time
holdoff = 'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 256 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.
```

```
// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 620;
dump_period_mem1[tc] = 620;
dump_period_mem2[tc] = 620;
dump_period_mem3[tc] = 620;
dump_period_mem4[tc] = 620;
dump_period_mem5[tc] = 620;
dump_period_mem6[tc] = 620;
dump_period_mem7[tc] = 620;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------
```

```
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

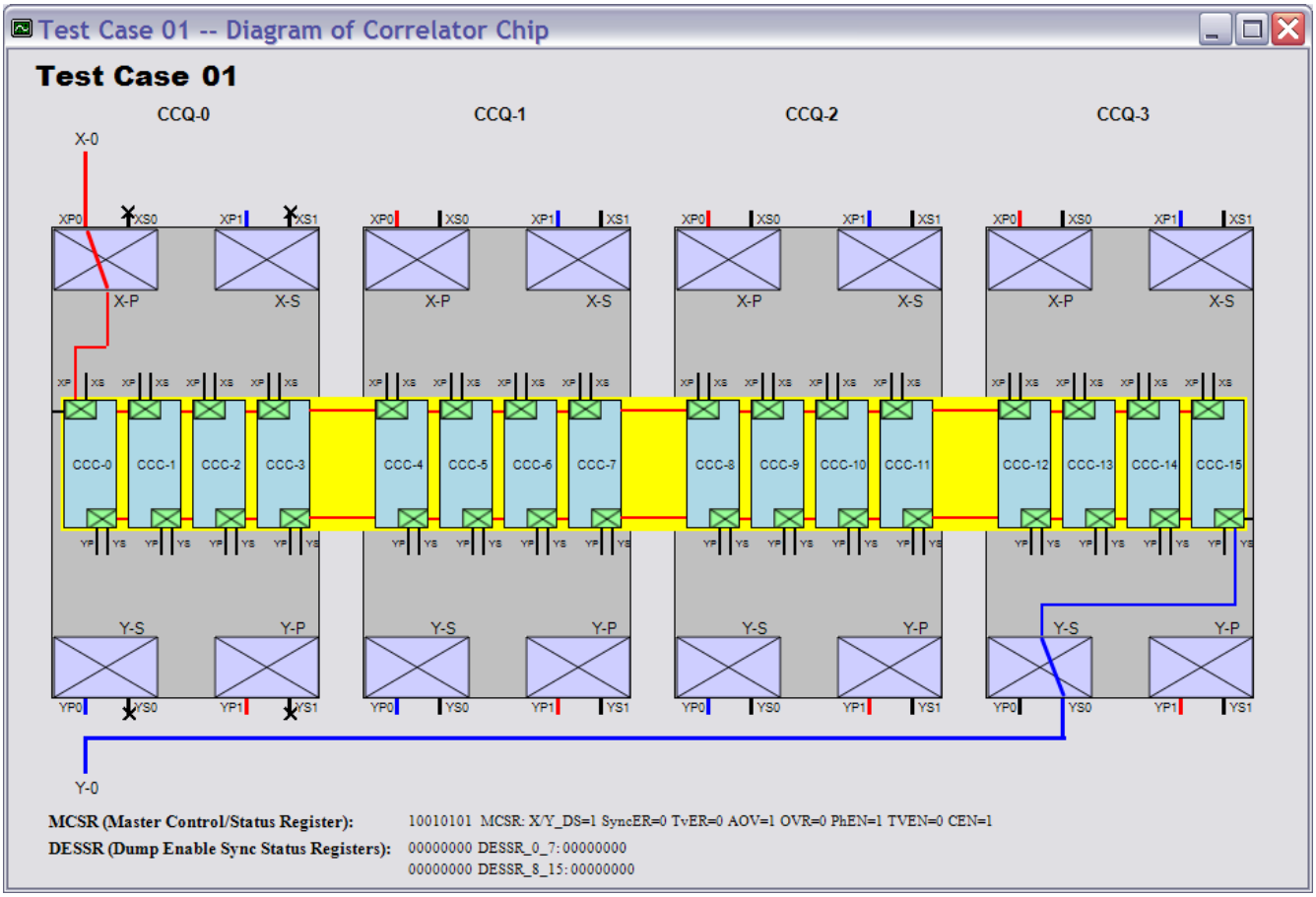
test_duration_mem[tc] = 620;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.2 Test case 1




```
// ********************************************************************************************************************
// File name : tc01_apr06-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE           VERSION          COMMENTS
// tc01_sep01-04_v1.v     Sep 1, 2004    1                creation
// tc01_apr06-05_v3.v     Apr 6, 2005    2                split accumulator and short integration; add
//                                                        verification mode option
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 1;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 0:
# ver: v2,05apr06
# Synopsis: CONCATENATE ALL, X DUMP, INPUT 0, SPLIT ACCUMULATOR OVERFLOW
# Description:
#              Purpose of Test:   Accumulate to overflow with split accumululator.
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 1044-0
#              Distinct samp/dump:   1044-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare results with: split accumulator bias 132352
#              Additional details:   valid data reset; 1044 sample bias is 66816.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_6_7_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_10_11_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//                             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b10;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

dts_len_mem[tc]= 312;  // increase size of dts_len to 312 bits total.

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to cover integration dead time
holdoff = 'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 256 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 812;
dump_period_mem1[tc] = 812;
dump_period_mem2[tc] = 812;
dump_period_mem3[tc] = 812;
dump_period_mem4[tc] = 812;
```

```
dump_period_mem5[tc] = 812;
dump_period_mem6[tc] = 812;
dump_period_mem7[tc] = 812;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

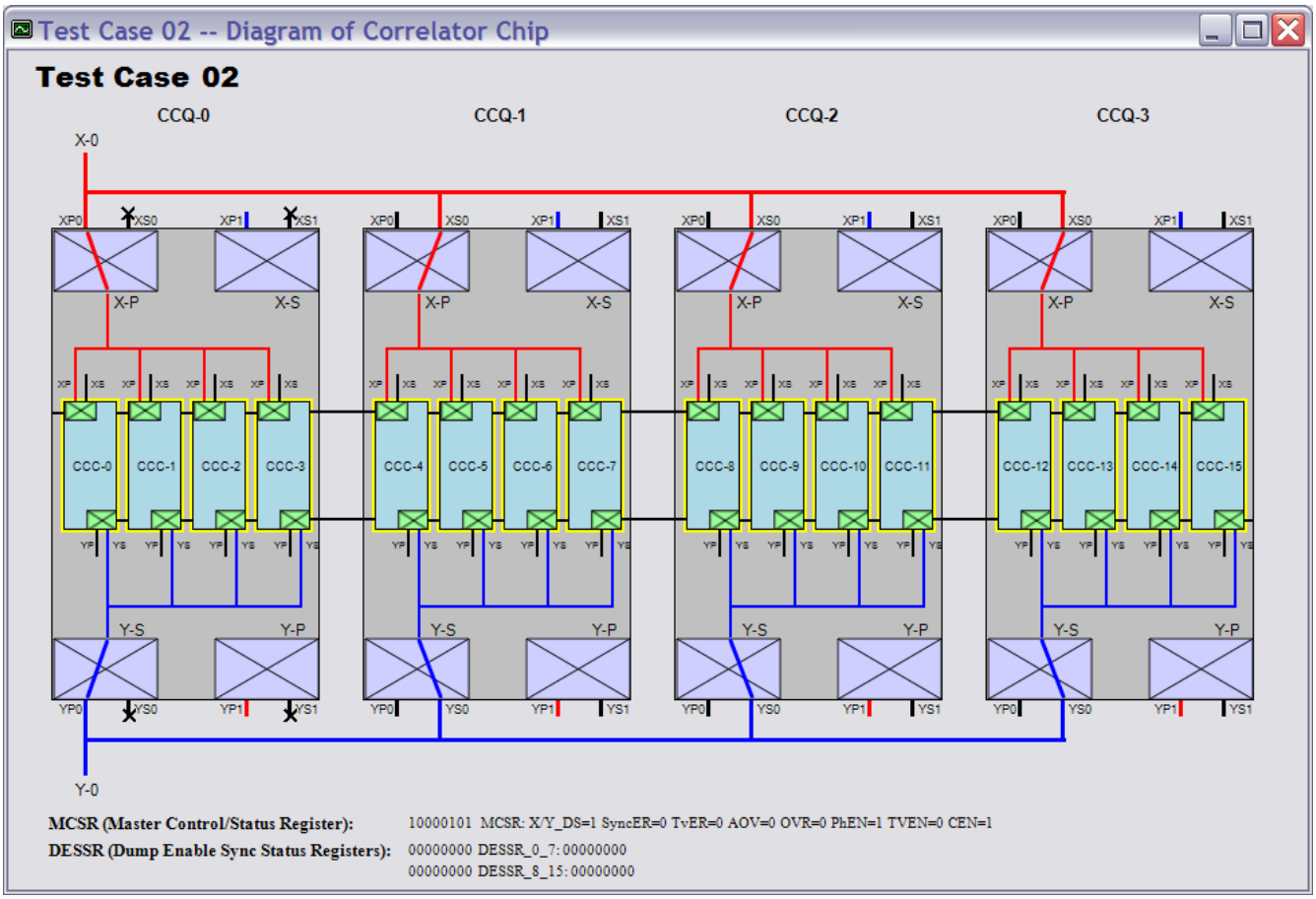
x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 812;
```

4.3 Test case 2




```
// ********************************************************************************************************************
// File name : tc02_apr26-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc02_mar08-04_v1.v     Mar 08, 2004    1                creation from base case B0
// tc02_mar10-04_v1.v     Mar 10, 2004    1                add to 'test_description' string
// tc02_aug28-04_v2.v     Aug 28, 2004    2                modify for new test bench, 10 dumps
// tc02_apr26-05_v3.v     Apr 26, 2005    3                add verification mode and split acc. options;
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 2;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 2:
# ver: v3,05apr26
# Synopsis: ALL INDEPENDENT CELLS, ALL INPUT 0, 10 DUMPS ON X
# Description:
#              Purpose of Test:      normal dump and readout
#              Concatenation: 0(0)0 , 0(1)0, 0(2)0, 0(3)0, 0(4)0, 0(5)0, 0(6)0,
#                             0(7)0 , 0(8)0, 0(9)0, 0(10)0, 0(11)0, 0(12)0, 0(13)0,
#                             0(14)0 , 0(15)0
#              Num of Dumps:         10
#              Dump Control:         X
#              Num samples/dump-inp: 10000-1
#              Distinct samp/dump:   10000-1
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  identical operation between cells
#              Additional details:
#
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_2_3_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_4_5_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_6_7_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_8_9_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_10_11_mem[tc] = 8'b_1______0______0______1______1______0______0______1;
cccscr_12_13_mem[tc] = 8'b_1______0______0______1______1______0______0______1;
cccscr_14_15_mem[tc] = 8'b_1______0______0______1______1______0______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b______0________1_________0________1_________0_________1_________0_________0;
ccqr_2_3_mem[tc] = 8'b______0________1_________0________1_________0_________1_________0_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;
```

```
//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

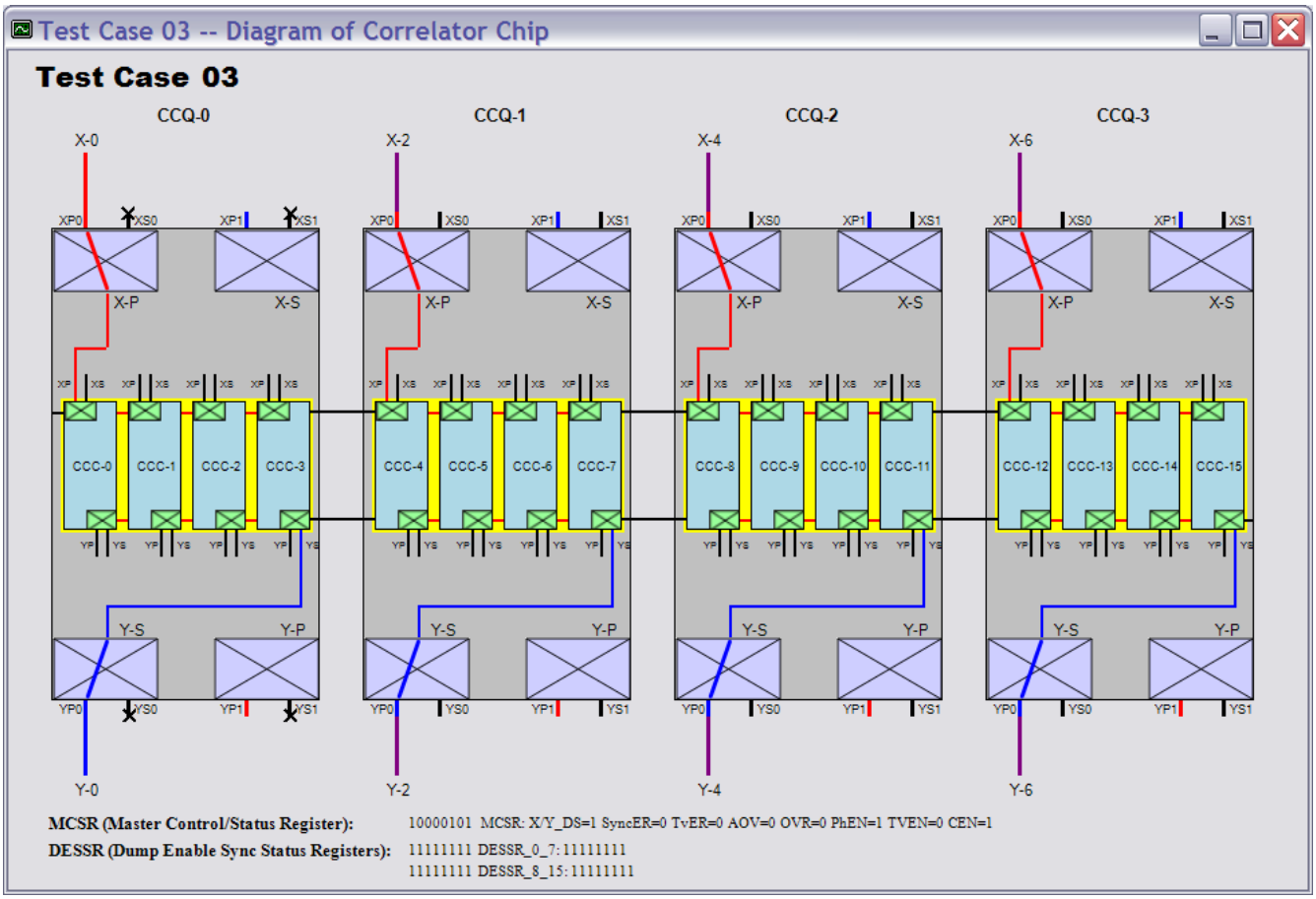
//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 50000;


//-------------------------------------------------------------------------------------------------------------
```

4.4 Test case 3




```
// ********************************************************************************************************************
// File name : tc03_jun14-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc03_mar08-04_v1.v     Mar 08, 2004    1                creation
// tc03_mar10-04_v1.v     Mar 10, 2004    1                add to 'test_description' string
// tc03_mar13-04_v1.v     Mar 13, 2004    1                add X-Y offset to dump_sync, fix description
// tc03_aug22-04_v2.v     Aug 22, 2004    2                modify for new test bench
// tc03_apr21-05_v3.v     Apr 21, 2005    3                add verification mode and split acc. options;
//                                                         reduce dump interval; increase dump count
// tc03_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
//
// ********************************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                    BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 3;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 3:
# ver: v3,05jun14
# Synopsis:   CONCAT CELLS IN QUADS, 4 INPUTS, 4 DUMP RATES ON X
# Description:
#              Purpose of Test:      Apply different dump rates between 4 inputs, multiple
#                                 dumps and quad concatenation
#              Concatenation:        0(0:3)0, 2(4:7)2, 4(8:11)4, 6(12:15)6
#              Num of Dumps-input:   16-0,8-2,4-4,2-6
#              Dump Control:         X
#              Num samples/dump-inp: 12000-0, 24000-2, 48000-4, 96000-6
#              Distinct samp/dump:   12000-0, 24000-2, 48000-4, 96000-6
#              Active SE clocks:     0, 2, 4, 6
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   no data synch errors generated;
#                                 apply Y-X offset on DUMP_SYNC.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___1______0______0______0______0______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___1______0______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______0______1;
cccscr_10_11_mem[tc] = 8'b_1______0______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0__________0__________0__________0__________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0__________0__________0__________0__________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;
y_se_clk_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;
y_dvalid_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 12000;
dump_period_mem3[tc] = 12000;
dump_period_mem4[tc] = 24000;
dump_period_mem5[tc] = 24000;
dump_period_mem6[tc] = 48000;
dump_period_mem7[tc] = 48000;
```

```
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 2;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

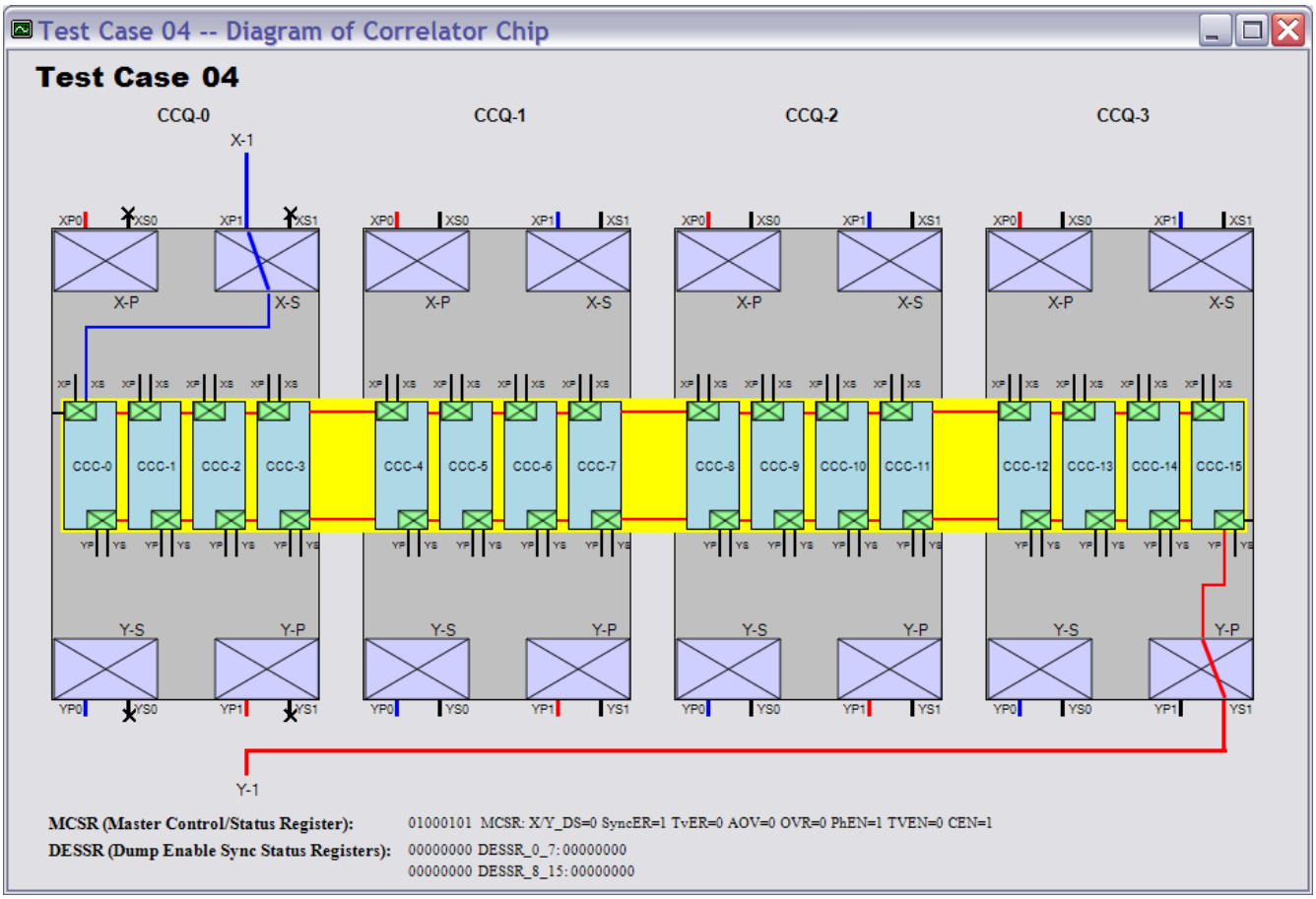
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 96000;


//-----------------------------------------------------------------------------------------------------------------
```

4.5 Test case 4




```
// ********************************************************************************************************************
// File name : tc04_apr21-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc04_mar07-04_v1.v     Mar 07, 2004    1                creation
// tc04_mar08-04_v1.v     Mar 08, 2004    1                no X-Y dump sync, apply inp sync errors
// tc04_mar10-04_v1.v     Mar 10, 2004    1                add to test_description string, modify the
//                                                         input sync error generation rate
//
// tc04_aug22-04_v2.v     Aug 22, 2004    2                modify for new test bench; activate all SE_CLK
// tc04_apr21-05_v3.v     Apr 21, 2005    3                add verification mode and split acc. options
//
// ********************************************************************************************************************
//
```

```
//
// ***************************************************************************************************************
// ***************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***************************************************************************************************************
// ***************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 4;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 4:
# ver: v3,05apr21
# Synopsis: CONCATENATE ALL, Y DUMP, INPUT 1 DATA ONLY.
# Description:
#              Purpose of Test:      All cell concatenation with alternate (to case 0)
#                                 input and dump control
#              Concatenation:        1(0:15)1
#              Num of dumps-inp:     1-1 after 10000 cycles
#              Dump Control:         y
#              Num samples/dump-inp: 20000-1
#              Distinct samp/dump:   10000-1
#              Active SE clocks:     input 1
#              Modulo se_clk gen:    modulo 2
#              Compare result with:  behavioural simulation test vectors for /DS4/CCinput0
#              Additional details:   data synch errors generated;
#                             no Y-X offset on DUMP_SYNC and SCHID_FRAME.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____0_______0_______0_____0_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________0__________0________0_________0_________0_________0_________0;
```

```
//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen0_mem[tc] = {70'bx,2'b0};
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
```

```
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 15;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

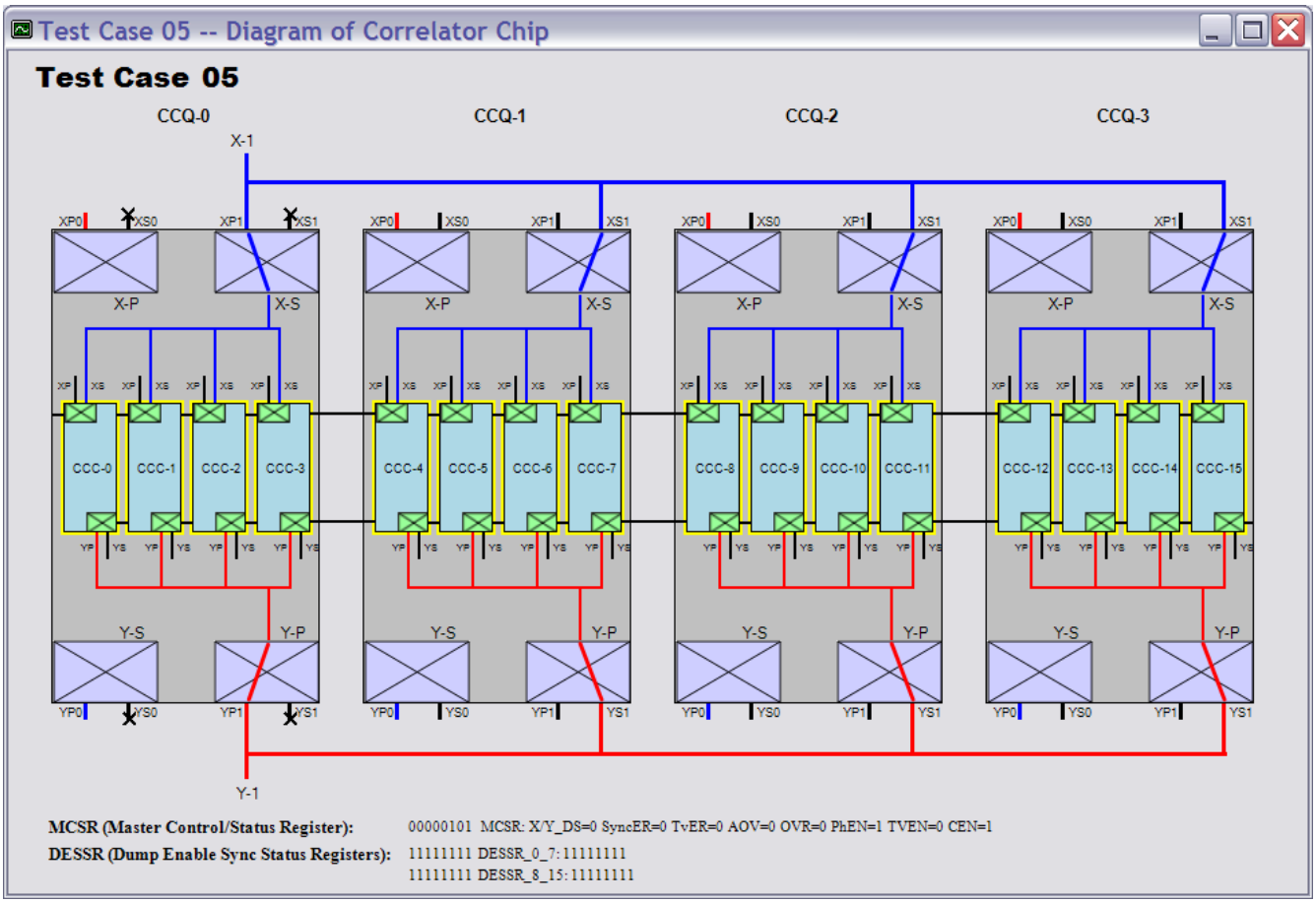
x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;
```

4.6 Test case 5




```
// ********************************************************************************************************************
// File name : tc05_jun14-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc05_mar08-04_v1.v     Mar 08, 2004    1                creation
// tc05_mar10-04_v1.v     Mar 10, 2004    1                add to 'test_description' string
// tc05_mar12-04_v1.v     Mar 12, 2004    1                add X-Y offset on dump_sync
// tc05_aug22-04_v2       Aug 22, 2004    2                modify for new test bench
// tc05_apr21-05_v3.v     Apr 21, 2005    3                add verification mode and split acc. options
// tc05_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
//
// ********************************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                    BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 5;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 5:
# ver: v3,05jun14
# Synopsis: ALL INDEPENDENT CELLS, ALL INPUT 1, 3 DUMPS ON Y, MOD CLK
# Description:
#              Purpose of Test:      Basic functional test of all cells
#              Concatenation:        1(0)1 , 1(1)1, 1(2)1, 1(3)1, 1(4)1, 1(5)1, 1(6)1,
#                                 1(7)1 , 1(8)1, 1(9)1, 1(10)1, 1(11)1, 1(12)1, 1(13)1,
#                                 1(14)1 , 1(15)1
#              Num of Dumps:         3
#              Dump Control:         Y
#              Num samples/dump-inp: 12000-1
#              Distinct samp/dump:   6000-1
#              Active SE clocks:     1
#              Modulo se_clk gen:    mod 2 clk
#              Compare result with:  identical operation between cells
#              Additional details:   no data sync errors applied;
#                                    apply Y-X offset on DUMP_SYNC.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____0_______0_______0_____0_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_2_3_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_4_5_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_6_7_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_8_9_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_10_11_mem[tc] = 8'b_0_____1_____1_____0_____0_____1_____1_____0;
cccscr_12_13_mem[tc] = 8'b_0_____1_____1_____0_____0_____1_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____1_____0_____0_____1_____1_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________0__________1________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________0__________1________0_________1_________0_________1_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping.  Only one input line is used.
// x and y dump enable are made identical to test the DESSR register operation
// dump_en is transmitted LSB first.
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen0_mem[tc] = {70'bx,2'b0};
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
```

```
dump_period_mem7[tc] = 6000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 4;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

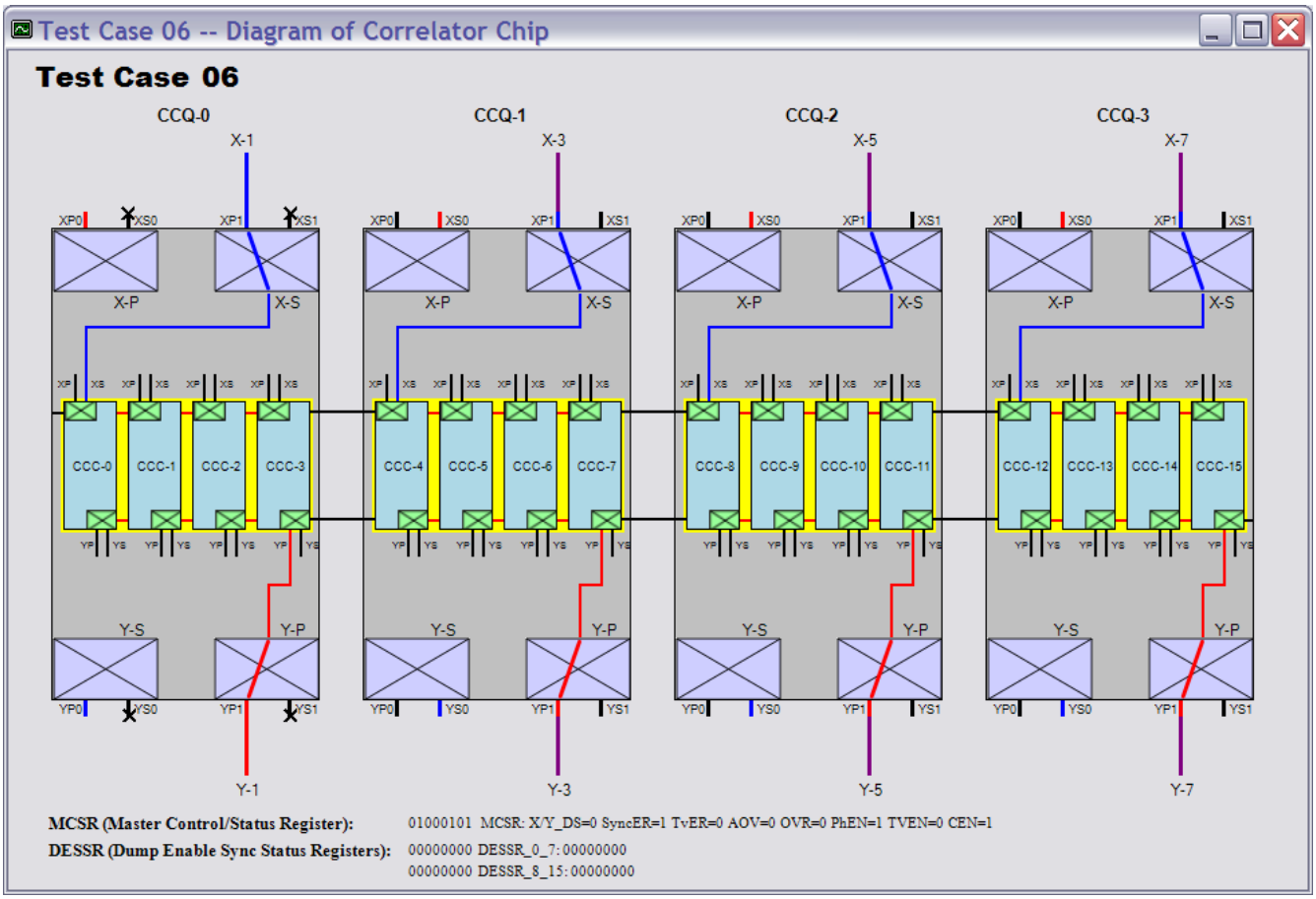
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 18000;

//-----------------------------------------------------------------------------------------------------------------
```

4.7 Test case 6




```
// ********************************************************************************************************************
// File name : tc06_jul24-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc06_mar09-04_v1.v     Mar 09, 2004    1                creation
// tc06_mar10-04_v1.v     Mar 10, 2004    1                add to 'test_description' string, modify input
//                                                         sync error generation rate.
// tc06_mar13-04_v1.v     Mar 13, 2004    1                fix 'test_description' string
// tc06_aug25-04_v2.v     Aug 25, 2004    2                modify for new test bench
// tc06_apr21-05_v3.v     Apr 21, 2005    3                add verification mode and split acc. options
// tc06_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
// tc06_jul24-05_v3.v     Jul 24, 2005    3                fix dump interval
//
// ********************************************************************************************************************
```

```
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 6;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 6:
# ver: v3,05jul24
# Synopsis:   CONCAT CELLS IN QUADS, 4 INPUTS, 4 DUMP RATES ON Y, MOD CLK, SYNC ERR
# Description:
#              Purpose of Test:      normal dump and readout with sync. error detection
#              Concatenation:        1(0:3)1, 3(4:7)3, 5(8:11)5, 7(12:15)7
#              Num of Dumps-input:   16-1,8-3,4-5,2-7
#              Dump Control:         y
#              Num samples/dump-inp: 12000-1, 24000-3, 48000-5, 96000-7
#              Distinct samp/dump:   6000-1, 12000-3, 24000-5, 48000-7
#              Active SE clocks:     1,3,5,7
#              Modulo se_clk gen:    mod 2 on all
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   data sync errors generated.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______1______0;
cccscr_2_3_mem[tc] = 8'b___0______1______0______0______0______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______1______0;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______1______0;
cccscr_10_11_mem[tc] = 8'b_0______1______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______1______0;
cccscr_14_15_mem[tc] = 8'b_0______1______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________1__________0________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1__________0________1_________0_________1_________0_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------------


//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______0_______1_______0_______1_______0_______1_______0;
y_se_clk_mem[tc] = 8'b_____1_______0_______1_______0_______1_______0_______1_______0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__1_______0_______1_______0_______1_______0_______1_______0;
y_se_clkmod_mem[tc] = 8'b__1_______0_______1_______0_______1_______0_______1_______0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______0_______1_______0_______1_______0_______1_______0;
y_dvalid_mem[tc] = 8'b_____1_______0_______1_______0_______1_______0_______1_______0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. Four input lines are used.
// dump_en is transmitted LSB first.
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen0_mem[tc] = {70'bx,2'b0};
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 12000;
dump_period_mem3[tc] = 12000;
dump_period_mem4[tc] = 24000;
```

```
dump_period_mem5[tc] = 24000;
dump_period_mem6[tc] = 48000;
dump_period_mem7[tc] = 48000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 130;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

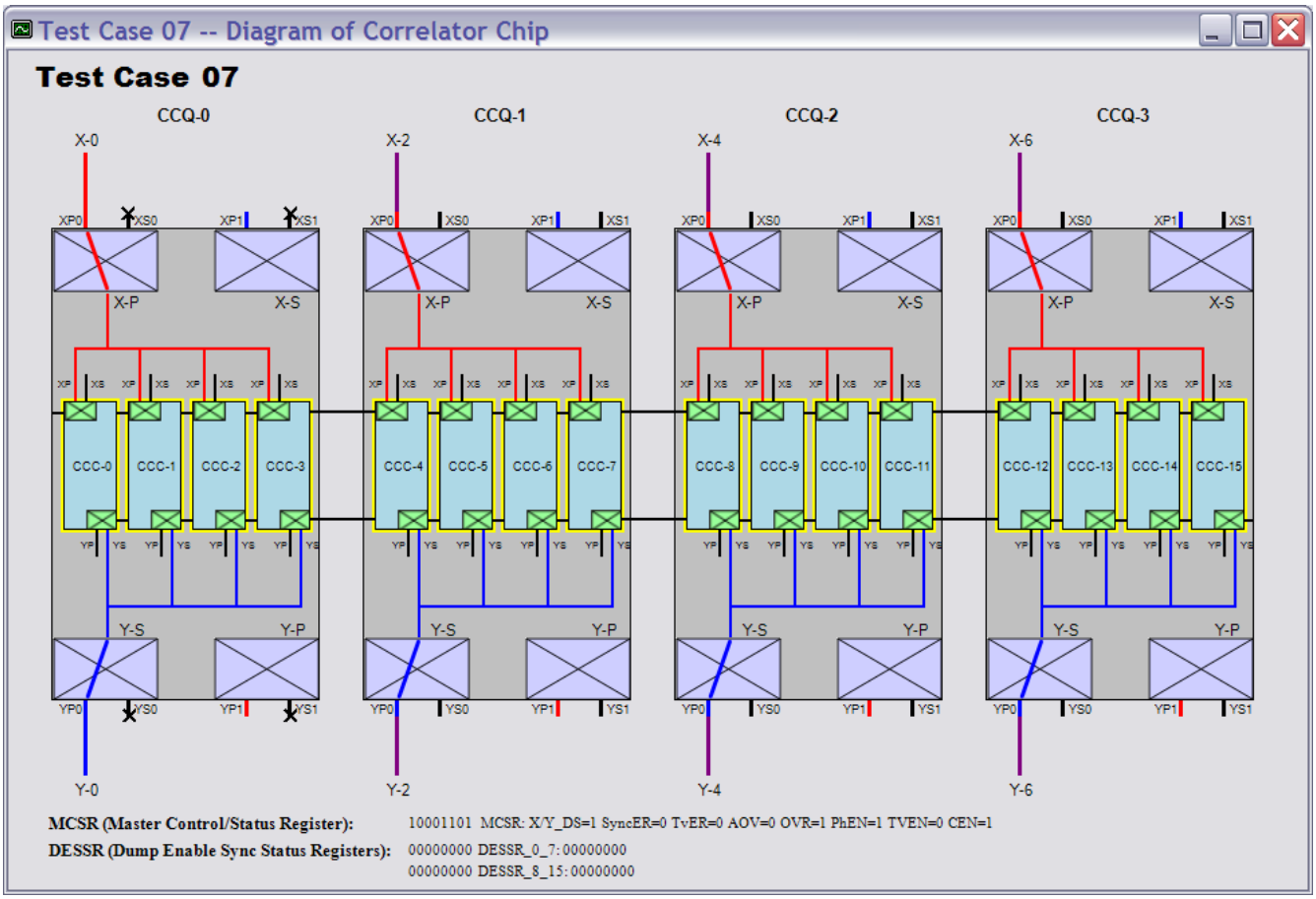
x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 96000;
```

4.8 Test case 7




```
// ********************************************************************************************************************
// File name : tc07_dec09-05_v4.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V4
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc07_mar09-04_v1.v     Mar 09, 2004    1                creation
// tc07_aug25-04_v2.v     Aug 25, 2004    2                modify for new test bench
// tc07_apr21-05_v3.v     Apr 21, 2005    3                modify for new test bench
// tc07_dec09-05_v4.v     Dec 09, 2005    4                increase dump interval for stream 0
//
//
// ********************************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                    BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 7;
test_description[tc] =
"# ---------------------------------------------------------------------------
# TEST 7:
# ver: v4,05dec09
# Synopsis:   INDEPENDENT CELLS, 4 INPUTS, 4 DUMP RATES ON X, EXTREME DUMP RATE
# Description:
#              Purpose of Test:      dump request at extreme rates
#              Concatenation:        0(0)0 , 0(1)0, 0(2)0, 0(3)0, 2(4)2, 2(5)2, 2(6)2,
#                           2(7)2 , 4(8)4, 4(9)4, 4(10)4, 4(11)4, 6(12)6, 6(13)6,
#                           6(14)6 , 6(15)6
#              Num of Dumps-input:   ~80-0, 20-2, 4-4, 2-6
#              Dump Control:         X
#              Num. of samples/dump: 500, 2000, 10000, 20000
#              Active SE clocks:     0,2,4,6
#              Modulo se_clk gen:    none
#              Compare result with:  between cell frames and expected operation
#              Additional details:
#
# ---------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____0_____1_____1_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____1_____1_____0_____0_____1;
cccscr_4_5_mem[tc] = 8'b___1_____0_____0_____1_____1_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____1_____1_____0_____0_____1;
cccscr_8_9_mem[tc] = 8'b___1_____0_____0_____1_____1_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____1_____1_____0_____0_____1;
cccscr_12_13_mem[tc] = 8'b_1_____0_____0_____1_____1_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____1_____1_____0_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                       On (1) or off (0).
//
```

```
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;
y_se_clk_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;
y_dvalid_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
```

```
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 250;
dump_period_mem1[tc] = 250;
dump_period_mem2[tc] = 1000;
dump_period_mem3[tc] = 1000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).
```

```
sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

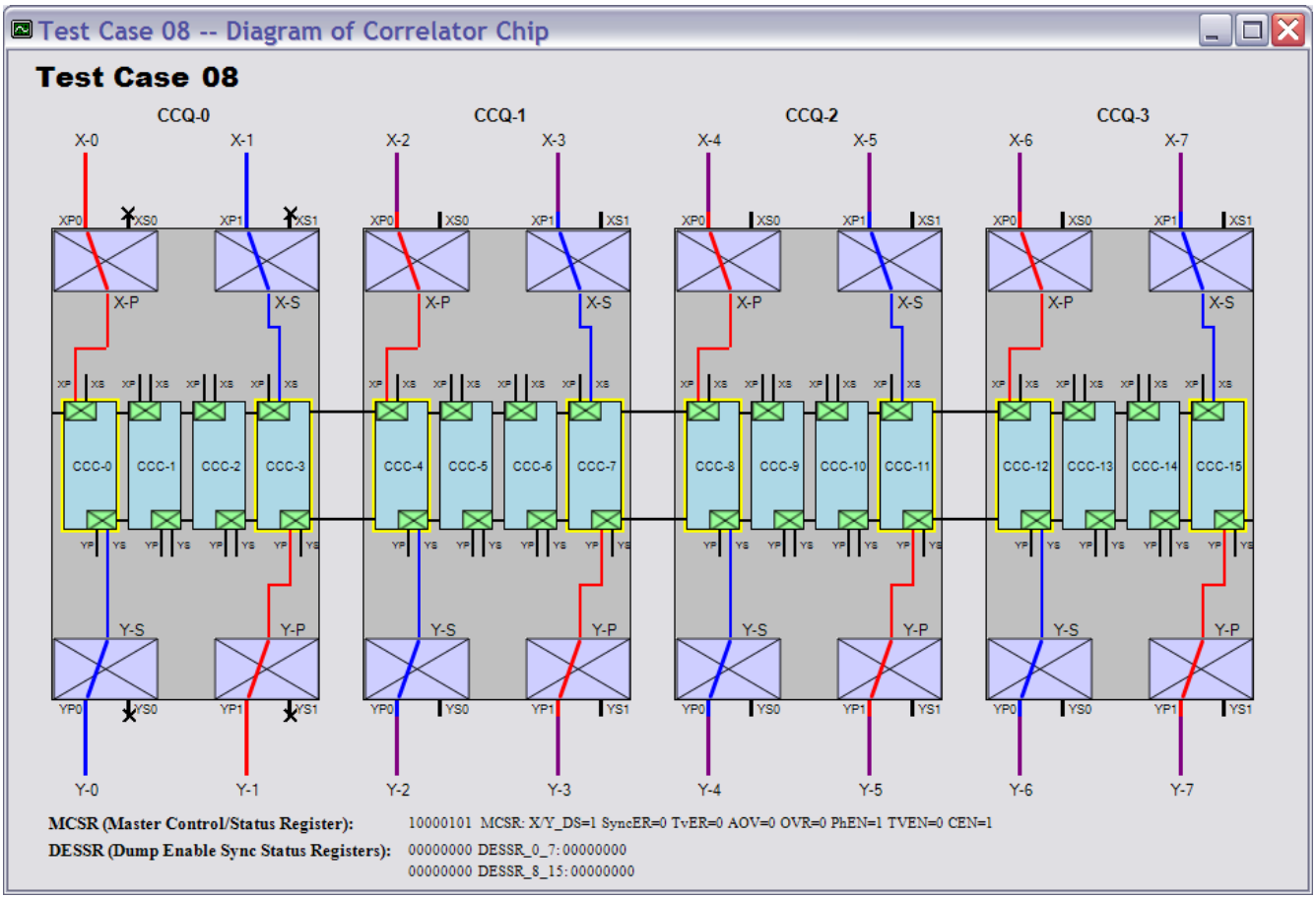
test_duration_mem[tc] = 20000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// *****************************************************************************************************************
```

4.9 Test case 8




```
// ********************************************************************************************************************
// File name : tc08_apr21-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc08_mar11-04_v1.v     Mar 11, 2004    1                creation
// tc08_jul26-04_v2.v     Jul 26, 2004    2                modify for new test bench, mult-dump
// tc08_apr21-05_v3.v     Apr 21, 2005    3                add verification mode and split acc. options
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 8;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 8:
# ver: v3,05apr21
# Synopsis:   ALL ACTIVE INP WITH MOD CLK, END CELLS OF QUADS, 3 DUMP ON X
# Description:
#              Purpose of Test:      check modulo SE clk operation; data path to end
#                                 cells of each quad
#              Concatenation:   0(0)0, 1(3)1, 2(4)2, 3(7)3, 4(8)4, 5(11)5, 6(12)6, 7(15)7
#              Num of Dumps-input:   3 on all
#              Dump Control:         X
#              Num samples/dump-inp: 40000 on all
#              Distinct samp/dump:   20000-0, 13333-1, 10000-2, 8000-3, 6666-4, 5714-5,
#                                 5000-6, 4444-7
#              Active SE clocks:     all
#              Modulo se_clk gen:    modulo (input# + 2)
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   no data synch errors applied;
#                                 no Y-X offset on DUMP_SYNC and SCHID_FRAME.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______1______1______1______1______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______1______0______1______1______1______1;
cccscr_4_5_mem[tc] = 8'b___1______1______1______1______1______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______1______0______1______1______1______1;
cccscr_8_9_mem[tc] = 8'b___1______1______1______1______1______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______1______0______1______1______1______1;
cccscr_12_13_mem[tc] = 8'b_1______1______1______1______1______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______1______0______1______1______1______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b______0________0_________0________0_________0_________0_________0_________0;
```

```
//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__1_______1_______1_______1_______1_______1_______1_______1;
y_se_clkmod_mem[tc] = 8'b__1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping. All lines are used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 20000;
dump_period_mem1[tc] = 20000;
dump_period_mem2[tc] = 20000;
dump_period_mem3[tc] = 20000;
dump_period_mem4[tc] = 20000;
dump_period_mem5[tc] = 20000;
```

```
dump_period_mem6[tc] = 20000;
dump_period_mem7[tc] = 20000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

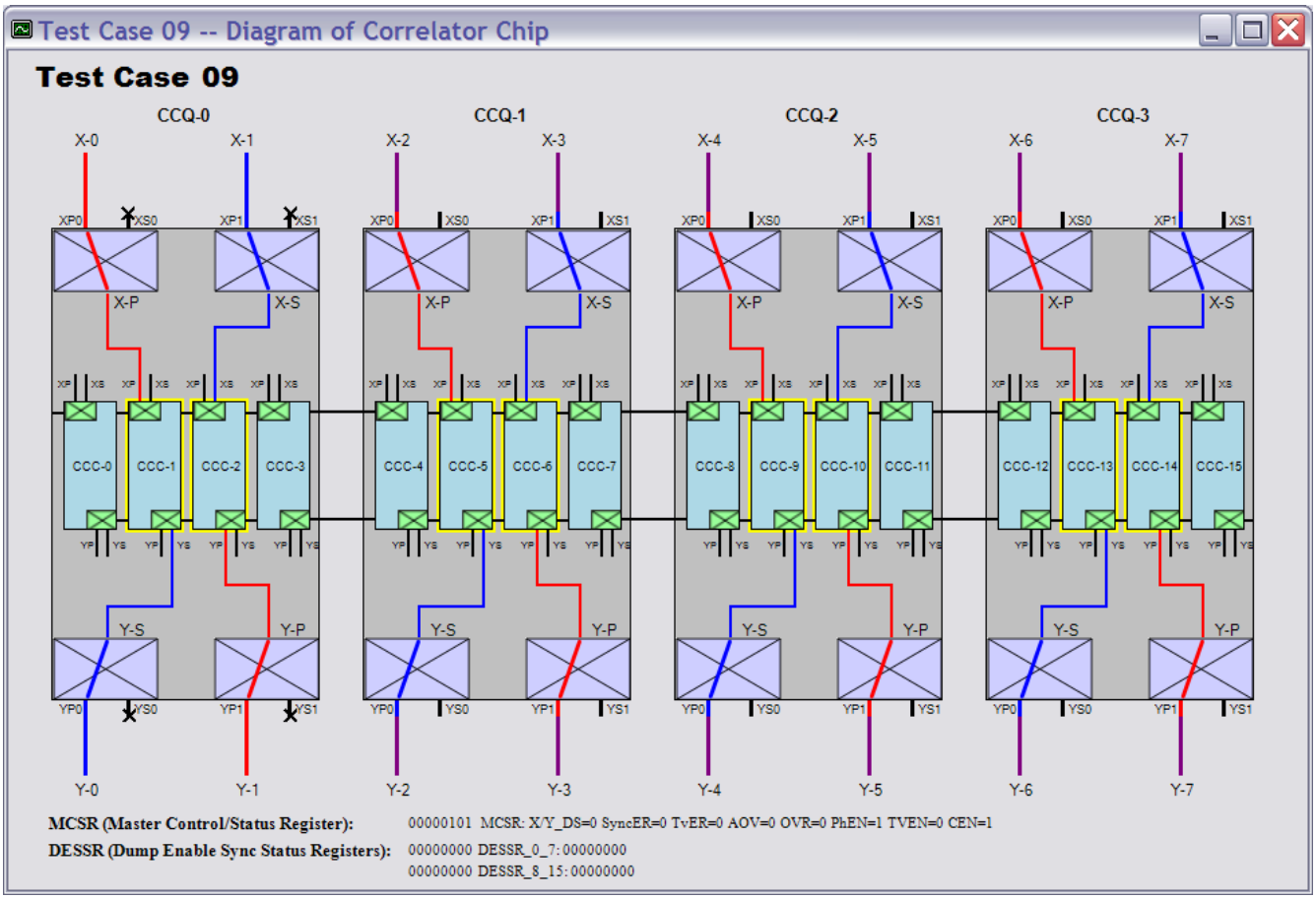
x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 60000;
```

4.10 Test case 9




```
// ********************************************************************************************************************
// File name : tc09_apr21-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc09_mar11-04_v1.v     Mar 11, 2004    1                creation
// tc09_jul26-04_v2.v     Jul 26, 2004    2                modify for new test bench, mult-dump
// tc09_apr21-05_v3.v     Apr 21, 2005    3                add verification mode and split acc. options
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 9;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 9:
# ver: v3,05apr21
# Synopsis:   ALL ACTIVE INP WITH MOD CLK, MID CELLS OF QUADS, 3 DUMP ON Y
# Description:
#              Purpose of Test:      check modulo SE clk operation; data path to end
#                                 cells of each quad
#              Concatenation:   0(1)0, 1(2)1, 2(5)2, 3(6)3, 4(9)4, 5(10)5, 6(13)6, 7(14)7
#              Num of Dumps-input:   3 on all
#              Dump Control:         Y
#              Num samples/dump-inp: 40000 on all
#              Distinct samp/dump:   20000-0, 13333-1, 10000-2, 8000-3, 6666-4, 5714-5,
#                                 5000-6, 4444-7
#              Active SE clocks:     all
#              Modulo se_clk gen:    modulo (input # + 2)
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   no data synch errors applied;
#                                 no Y-X offset on DUMP_SYNC and SCHID_FRAME.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______1______1______1______1______1;
cccscr_2_3_mem[tc] = 8'b___1______1______1______1______0______1______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______1______1______1______1______1;
cccscr_6_7_mem[tc] = 8'b___1______1______1______1______0______1______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______1______1______1______1______1;
cccscr_10_11_mem[tc] = 8'b_1______1______1______1______0______1______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______1______1______1______1______1;
cccscr_14_15_mem[tc] = 8'b_1______1______1______1______0______1______1______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b______0________0_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__1_______1_______1_______1_______1_______1_______1_______1;
y_se_clkmod_mem[tc] = 8'b__1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping. All lines are used.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 20000;
dump_period_mem1[tc] = 20000;
dump_period_mem2[tc] = 20000;
dump_period_mem3[tc] = 20000;
dump_period_mem4[tc] = 20000;
dump_period_mem5[tc] = 20000;
dump_period_mem6[tc] = 20000;
dump_period_mem7[tc] = 20000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

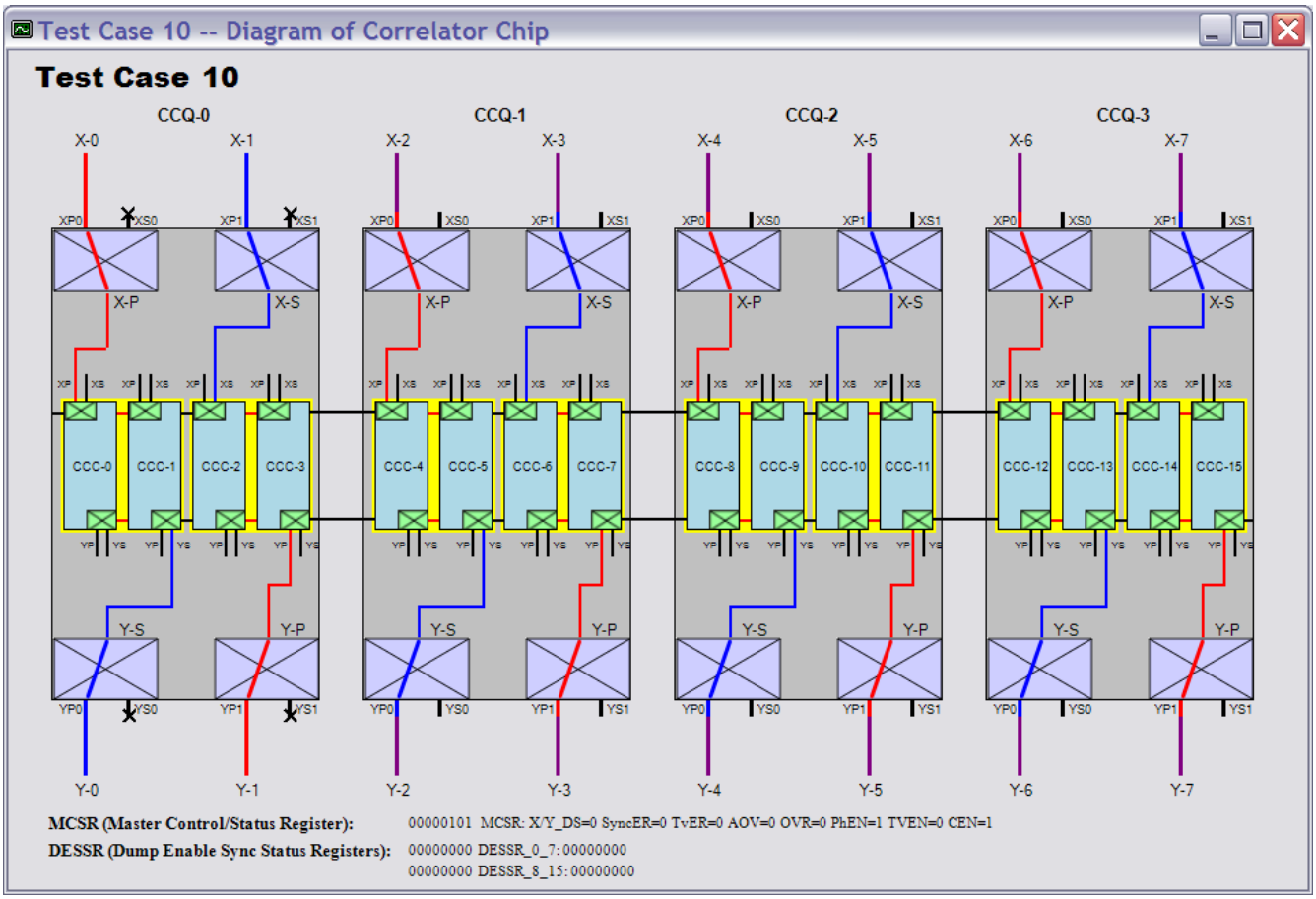
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 60000;

//-----------------------------------------------------------------------------------------------------------
```

4.11 Test case 10




```
// ********************************************************************************************************
// File name : tc10_jul24-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc10_mar12-04_v1.v     Mar 12, 2004    1                creation
// tc10_jul27-04_v2.v     Jul 27, 2004    2                modify for new test bench, mult-dump
// tc10_aug19-04_v2.v     Aug 19, 2004    2                comment revisions
// tc10_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc10_jul24-05_v3.v     Jul 24, 2005    3                increase run time to accomodate mult. dumps
//
// ********************************************************************************************************
//
//
// ********************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 10;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 10:
# ver: v3,05jul24
# Synopsis:   ALL ACTIVE INP WITH MOD CLK, CONCAT IN CELL PAIRS, 2 DUMP ON Y
# Description:
#              Purpose of Test:      normal dump and readout, SE clk in concat. pairs
#              Concatenation:        0(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)6, 7(14:15)7
#              Num of Dumps-input:   2 on all
#              Dump Control:         Y
#              Num samples/dump-inp: 40000 on all
#              Distinct samp/dump:   20000-0, 13333-1, 10000-2, 8000-3, 6666-4, 5714-5,
#                                 5000-6, 4444-7
#              Active SE clocks:     all
#              Modulo se_clk gen:    modulo (input # + 2)
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   no data synch errors applied;
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____0_______0_______0_____0_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____1_____0;
cccscr_4_5_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____1_____0;
cccscr_8_9_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____1_____0;
cccscr_12_13_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____1_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__1_______1_______1_______1_______1_______1_______1_______1;
y_se_clkmod_mem[tc] = 8'b__1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 20000;
dump_period_mem1[tc] = 20000;
dump_period_mem2[tc] = 20000;
dump_period_mem3[tc] = 20000;
dump_period_mem4[tc] = 20000;
dump_period_mem5[tc] = 20000;
dump_period_mem6[tc] = 20000;
dump_period_mem7[tc] = 20000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

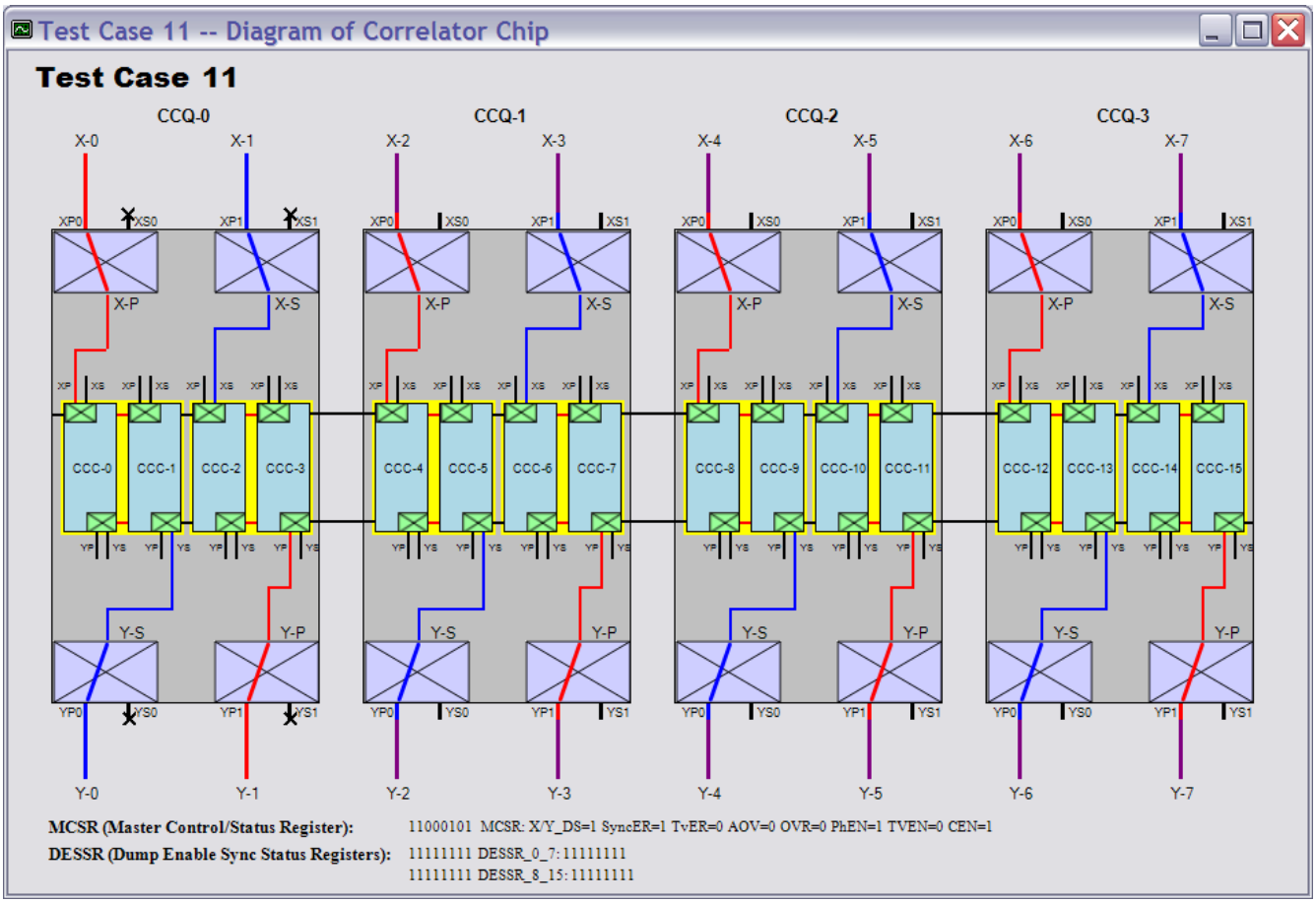
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 40000;

//-----------------------------------------------------------------------------------------------------------
```

4.12 Test case 11




```
// ********************************************************************************************************************
// File name : tc11_jun14-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc11_aug19-04_v1.v     Aug 19, 2004    1                creation
// tc11_apr22-05_v2.v     Apr 22, 2005    2                upgrade to new test bench
// tc11_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 11;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 11:
# ver: v2,05jun14
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, 3 DUMP ON X
# Description:
#              Purpose of Test:      check sync. error reporting
#              Concatenation:        0(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)6, 7(14:15)7
#              Num of Dumps-input:   3 on all
#              Dump Control:         X
#              Num samples/dump-inp: 12000 on all
#              Distinct samp/dump:   12000 on all
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with:  check error reporting
#              Additional details:   infrequent input synch errors applied
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         -----------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         -----------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______0______0______0______0______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______0______0______0______0______1______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits
```

```
// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 1024;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 1;

//-----------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 22;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 5;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

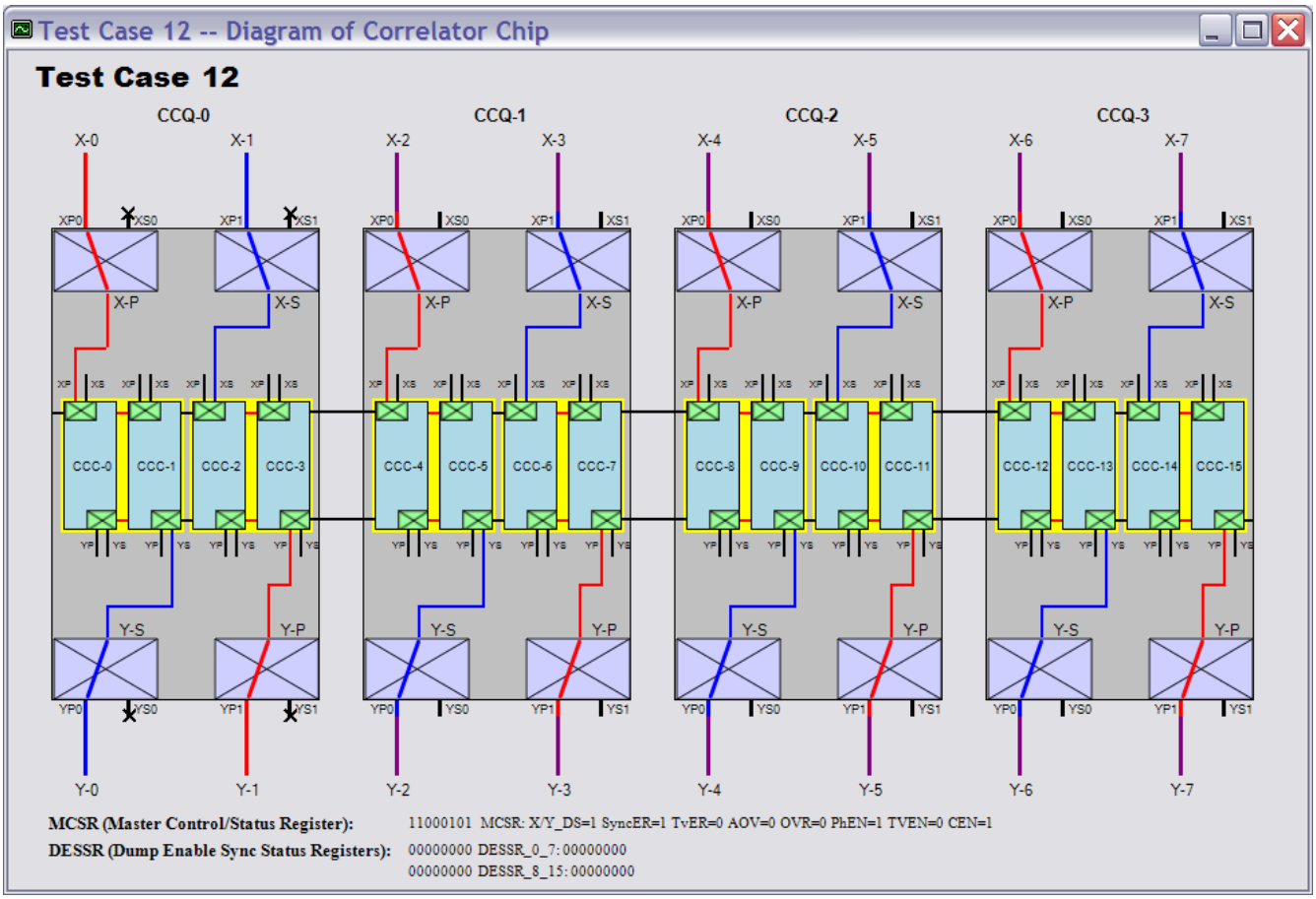
test_duration_mem[tc] = 18000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.13 Test case 12




```
// ********************************************************************************************************
// File name : tc12_may14-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc12_aug19-04_v1.v     Aug 19, 2004    1                creation
// tc12_apr22-05_v2.v     Apr 22, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc12_may14-05_v2.v     May 14, 2005    2                increase dump interval
//
// ********************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                    BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 12;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 12:
# ver: v2,05may14
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, 3 DUMP ON X
# Description:
#              Purpose of Test:      check sync. error reporting
#              Concatenation:        0(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)6, 7(14:15)7
#              Num of Dumps-input:   3 on all
#              Dump Control:         X
#              Num samples/dump-inp: 12000 on all
#              Distinct samp/dump:   12000 on all
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with: check error reporting on DUMP_EN
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0_____0_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____1_____0;
cccscr_4_5_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____1_____0;
cccscr_8_9_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____1_____0;
cccscr_12_13_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____1_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.

x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

fill_sync = 2'b11;                          // 2 bits
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
```

```
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

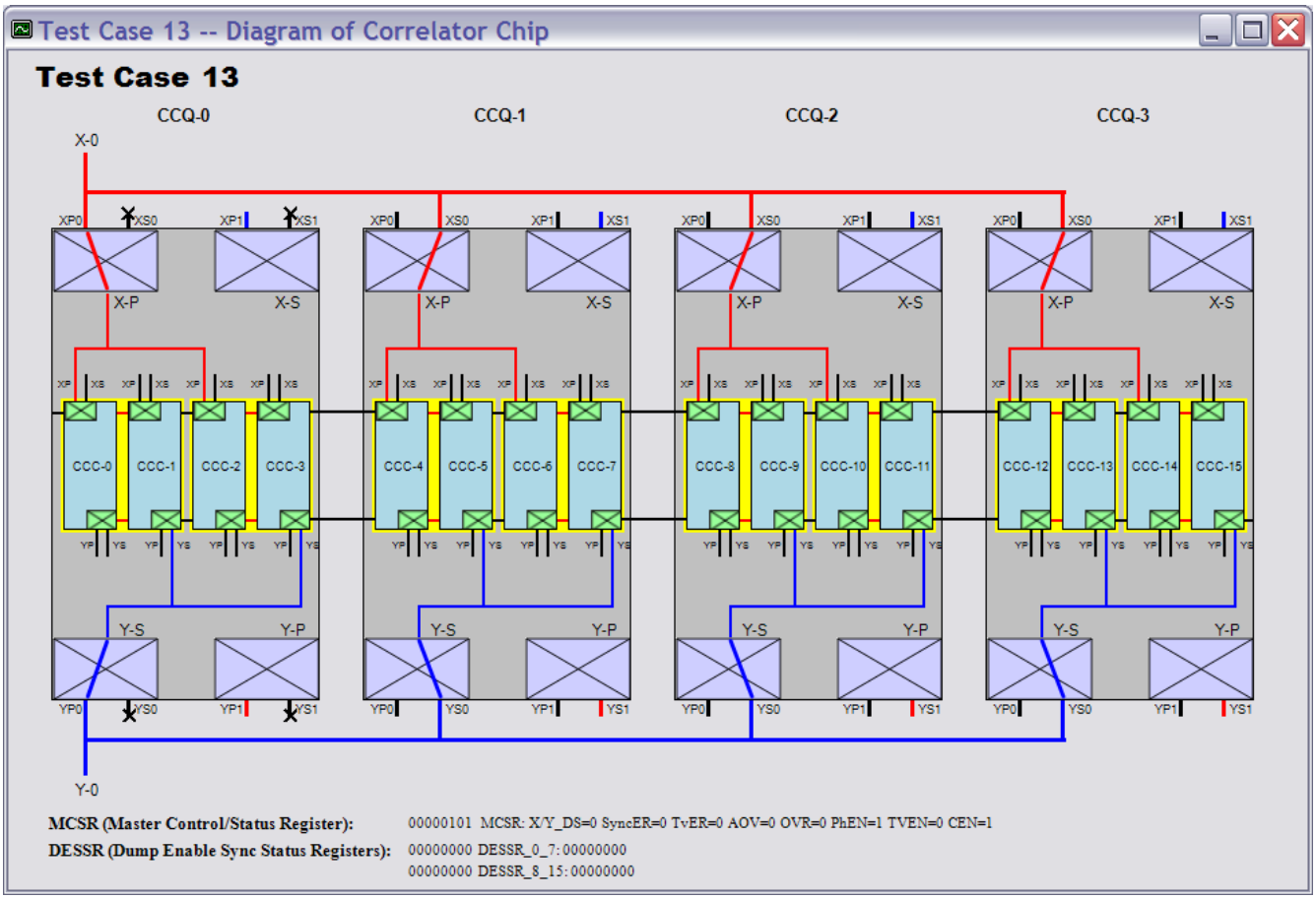
// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 18000;
```

4.14 Test case 13




```
// ********************************************************************************************************
// File name : tc13_apr22-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc13_mar13-04_v1.v     Mar 13, 2004    1                creation
// tc13_aug24-04_v2.v     Aug 24, 2004    2                modify for new test bench, concat. in pairs
// tc13_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 13;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 13:
# ver: v3,05apr22
# Synopsis:   CONCAT CELLS IN PAIRS, INPUT 0 TO ALL QUADS, 1 DUMP ON Y, MOD CLK
# Description:
#              Purpose of Test:      quad concatenation
#              Concatenation:        0(0:1)0, 0(2:3)0, 0(4:5)0, 0(6:7)0, 0(8:9)0,
#                                 0(10:11)0, 0(12:13)0, 0(14:15)0
#              Num of Dumps-input:   1-0
#              Dump Control:         Y
#              Num samples/dump-inp: 10000-0
#              Distinct samp/dump:   5000-0
#              Active SE clocks:     0,1,2,3
#              Modulo se_clk gen:    mod 2 on input 0
#              Compare result with:  identical operation between all quads
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_4_5_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_8_9_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_10_11_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_12_13_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1__________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1__________1________1_________1_________1_________1_________1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______1;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {70'bx,2'b0};
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

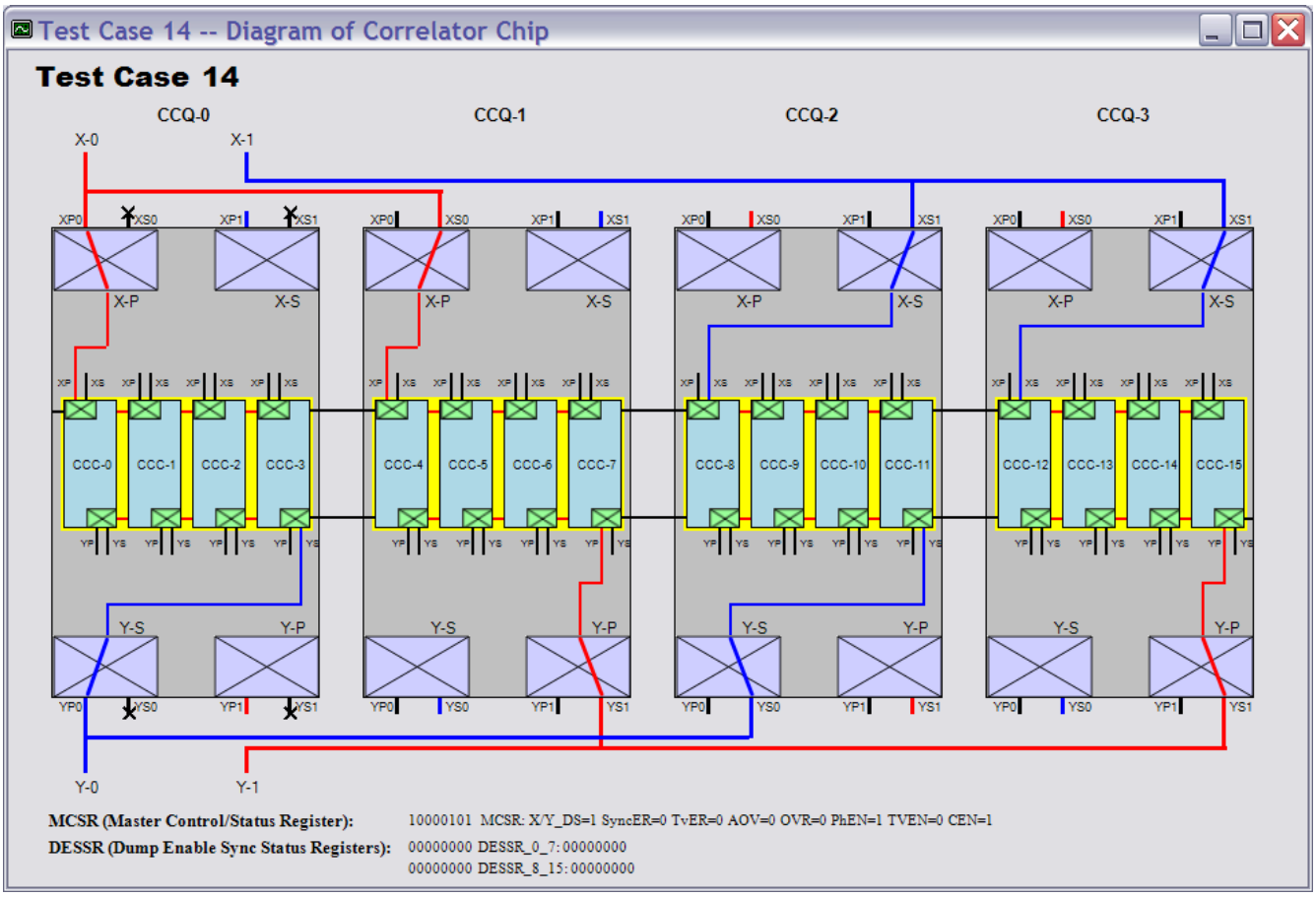
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
```

4.15 Test case 14




```
// ********************************************************************************************************************
// File name : tc14_apr22-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc14_mar13-04_v1.v     Mar 13, 2004    1                creation
// tc14_jul17-04_v2.v     Jul 17, 2004    2                new test bench, SE_CLK mod 2
// tc14_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
// ********************************************************************************************************************
//
```

```
//
// ***************************************************************************************************************
// ***************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***************************************************************************************************************
// ***************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 14;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 14:
# ver: v3,05apr22
# Synopsis:  CONCAT AS QUADS, INP 0,1 ALL POLN PRODUCTS, MOD 2 CLK, 3 DUMPS ON X
# Description:
#              Purpose of Test:      all poln products with quad concatenation
#              Concatenation:        0(0:3)0, 0(4:7)1, 1(8:11)0, 1(12:15)1
#              Num of Dumps-input:   3-0, 3-1
#              Dump Control:         X
#              Num samples/dump-inp: 20000-0, 20000-1
#              Distinct samp/dump:   10000-0, 10000-1
#              Active SE clocks:     0, 1
#              Modulo se_clk gen:    2-0, 2-1
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______1;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
```

```
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 30000;
```

4.16 Test case 15

This region is intentionally left blank.

```
// ********************************************************************************************************
// File name : tc15_apr22-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc15_mar13-04_v1.v     Mar 13, 2004    1                creation
// tc15_aug24-04_v2.v     Aug 24, 2004    2                modify for new test bench, all SE_CLK active
// tc15_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
// ********************************************************************************************************
//
```

```
//
// ***************************************************************************************************************
// ***************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***************************************************************************************************************
// ***************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 15;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 15:
# ver: v3,05apr22
# Synopsis:   CONCAT CELLS IN QUADS, INP 1, CELL SWITCH SEC, 1 DUMP ON X
# Description:
#              Purpose of Test:      quad cell concat, cell switches to secondary
#              Concatenation:        -(0:3)-, 1(4:7)1, 1(8:11)1, 1(12:15)1
#              Num of Dumps-input:   1-1
#              Dump Control:         X
#              Num samples/dump-inp: 10000-1
#              Distinct samp/dump:   10000-1
#              Active SE clocks:     0,1,2,3,4,5,6,7
#              Modulo se_clk gen:    none
#              Compare result with:  identical operation between cells, no data quad 0
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0_____0_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_2_3_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______0;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

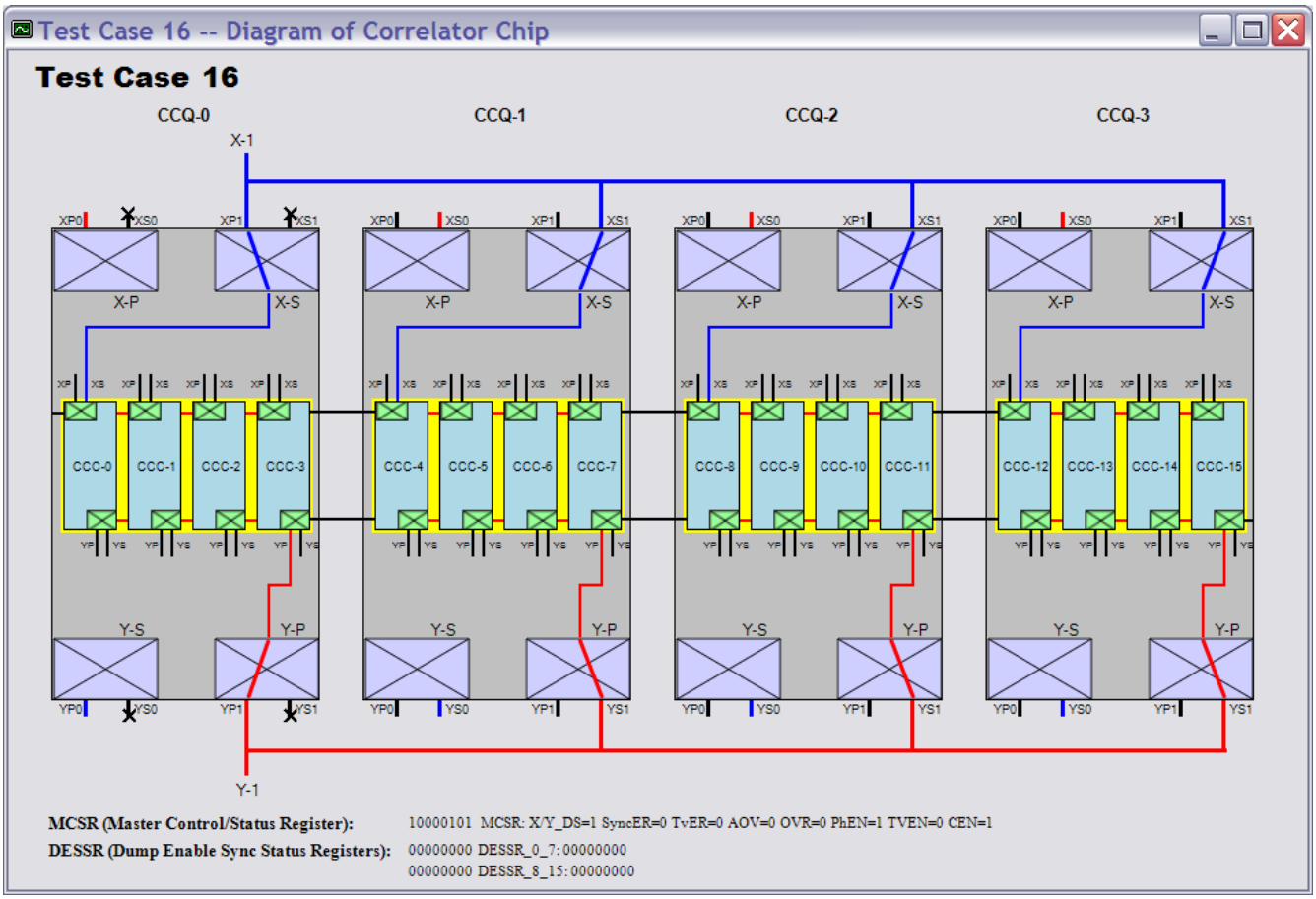
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
```

4.17 Test case 16




```
// ********************************************************************************************************
// File name : tc16_apr22-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc16_mar15-04_v1.v     Mar 15, 2004    1                creation
// tc16_aug24-04_v2.v     Aug 24, 2004    2                modify for new test bench, all SE_CLK active
// tc16_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 16;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 16:
# ver: v3,05apr22
# Synopsis:   CONCAT CELLS IN QUADS, INP 1 ON ALL CELLS, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      quad cell concat with inp 1
#              Concatenation:        1(0:3)1, 1(4:7)1, 1(8:11)1, 1(12:15)1
#              Num of Dumps-input:   1-1
#              Dump Control:         X
#              Num samples/dump-inp: 10000-1
#              Distinct samp/dump:   5000-1
#              Active SE clocks:     0,1,2,3,4,5,6,7
#              Modulo se_clk gen:    mod 2 on input 1
#              Compare result with:  identical operation between cells
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_2_3_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______0;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______0;


//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

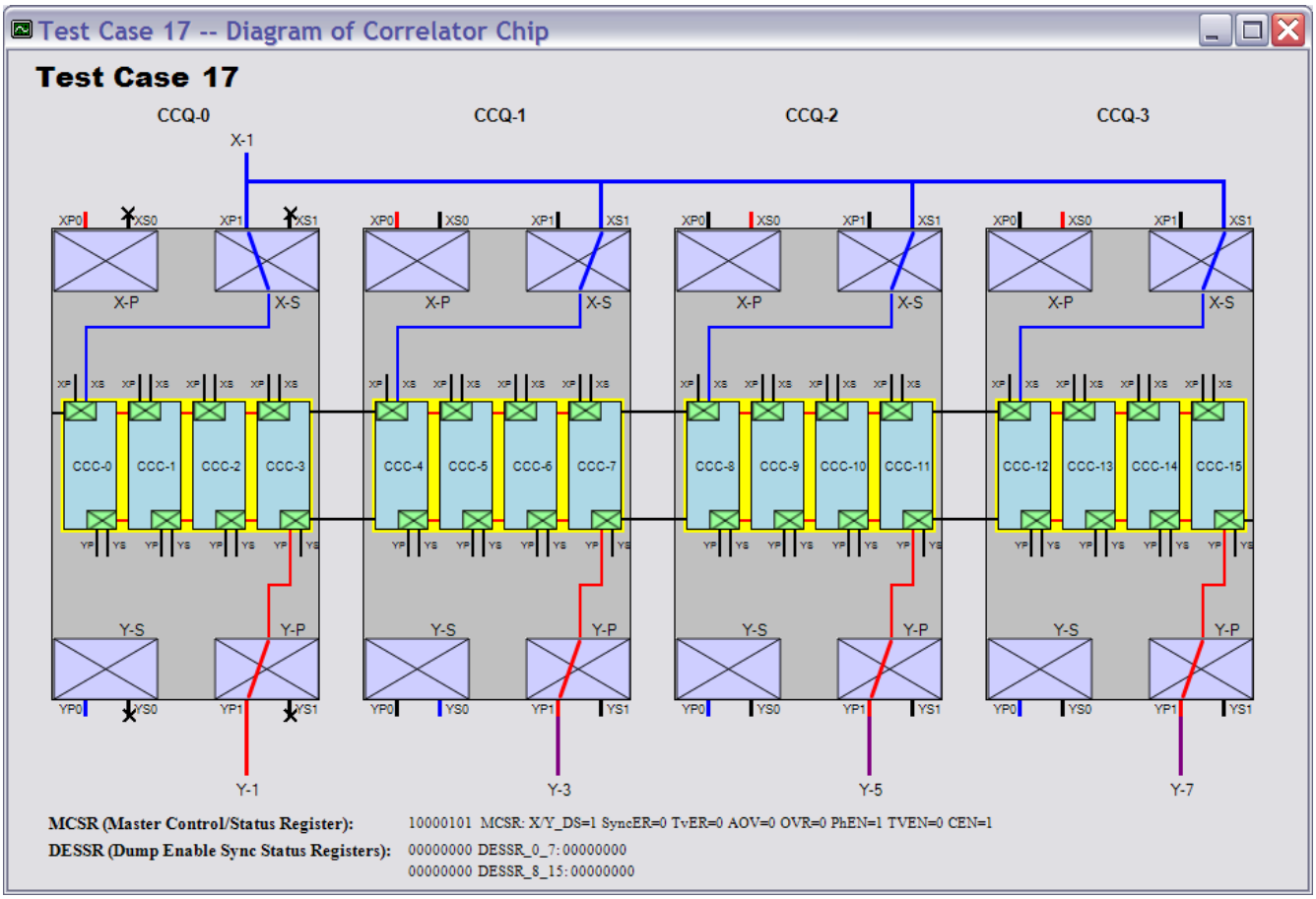
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
```

4.18 Test case 17




```
// ********************************************************************************************************
// File name : tc17_jul27-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc17_mar15-04_v1.v     Mar 15, 2004    1                creation
// tc17_aug24-04_v2.v     Aug 24, 2004    2                modify for new test bench, modify data ID and
//                                                         X-TIMESTAMP, 2 dump
// tc17_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
// tc17_jul27-05_v3.v     Jul 27, 2005    3                increase dump time to multiple of 12000
// ********************************************************************************************************
//
//
// ********************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 17;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 17:
# ver: v3,05jul27
# Synopsis:   CONCAT CELLS IN QUADS, MIXED ODD INPUTS, 2 DUMP ON X
# Description:
#              Purpose of Test:      independent X,Y quad switches, data-T.S. capture
#              Concatenation:        1(0:3)1, 1(4:7)3, 1(8:11)5, 1(12:15)7
#              Num of Dumps-input:   2-1, 2-3, 2-5, 2-7
#              Dump Control:         X
#              Num samples/dump-inp: 12000 on active inputs
#               Distinct samp/dump:   12000
#              Active SE clocks:     X-1; Y-1,3,5,7
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   check data ID and timestamp capture
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_2_3_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b______0________1__________1________0__________0__________0__________0__________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________1________1__________0__________0__________1__________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;
y_se_clk_mem[tc] = 8'b_____1_______0_______1_______0_______1_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;
y_dvalid_mem[tc] = 8'b_____1_______0_______1_______0_______1_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 'haaaaaaaa;
ts_1 = 'h55555555;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 21;
y_sbid_mem[tc] = 10;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

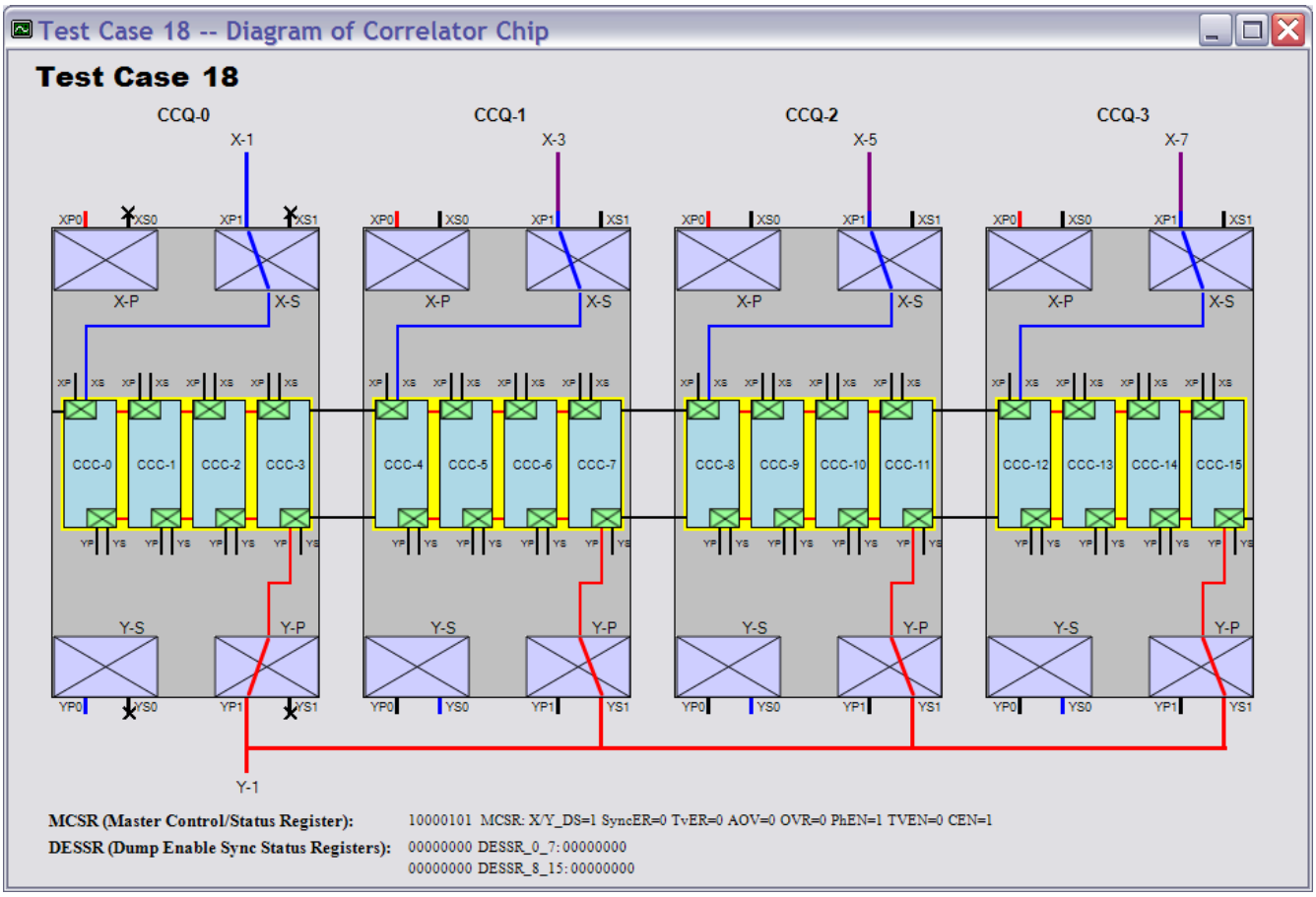
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 12000;

//-----------------------------------------------------------------------------------------------------------
```

4.19 Test case 18




```
// ********************************************************************************************************************
// File name : tc18_jul27-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc18_mar15-04_v1.v     Mar 15, 2004    1                creation
// tc18_aug24-04_v2.v     Aug 24, 2004    2                modify for new test bench, modify data ID and
//                                                         X-TIMESTAMP, 2 dump
// tc18_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
// tc18_jul27-05_v3.v     Jul 27, 2005    3                increase dump interval to multiple of 12000
//
// ********************************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                  BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 18;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 18:
# ver: v3,05jul27
# Synopsis:   CONCAT CELLS IN QUADS, MIXED ODD INPUTS, 2 DUMP ON X
# Description:
#              Purpose of Test:      independent X,Y quad switches, data-T.S. capture
#              Concatenation:        1(0:3)1, 3(4:7)1, 5(8:11)1, 7(12:15)1
#              Num of Dumps-input:   2-1, 2-3, 2-5, 2-7
#              Dump Control:         X
#              Num samples/dump-inp: 10000 on active inputs
#              Distinct samp/dump:   10000
#              Active SE clocks:     X-1,3,5,7; Y-1
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   check data ID and timestamp capture
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         -----------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         -----------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______1______0;
cccscr_2_3_mem[tc] = 8'b___0______1______0______0______0______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______1______0;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______1______0;
cccscr_10_11_mem[tc] = 8'b_0______1______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______1______0;
cccscr_14_15_mem[tc] = 8'b_0______1______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________0________0_________1_________1_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______0_______1_______0_______1_______0_______1_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______0_______1_______0_______1_______0_______1_______0;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 'h55555555;
ts_1 = 'haaaaaaaa;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 21;
y_sbid_mem[tc] = 10;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

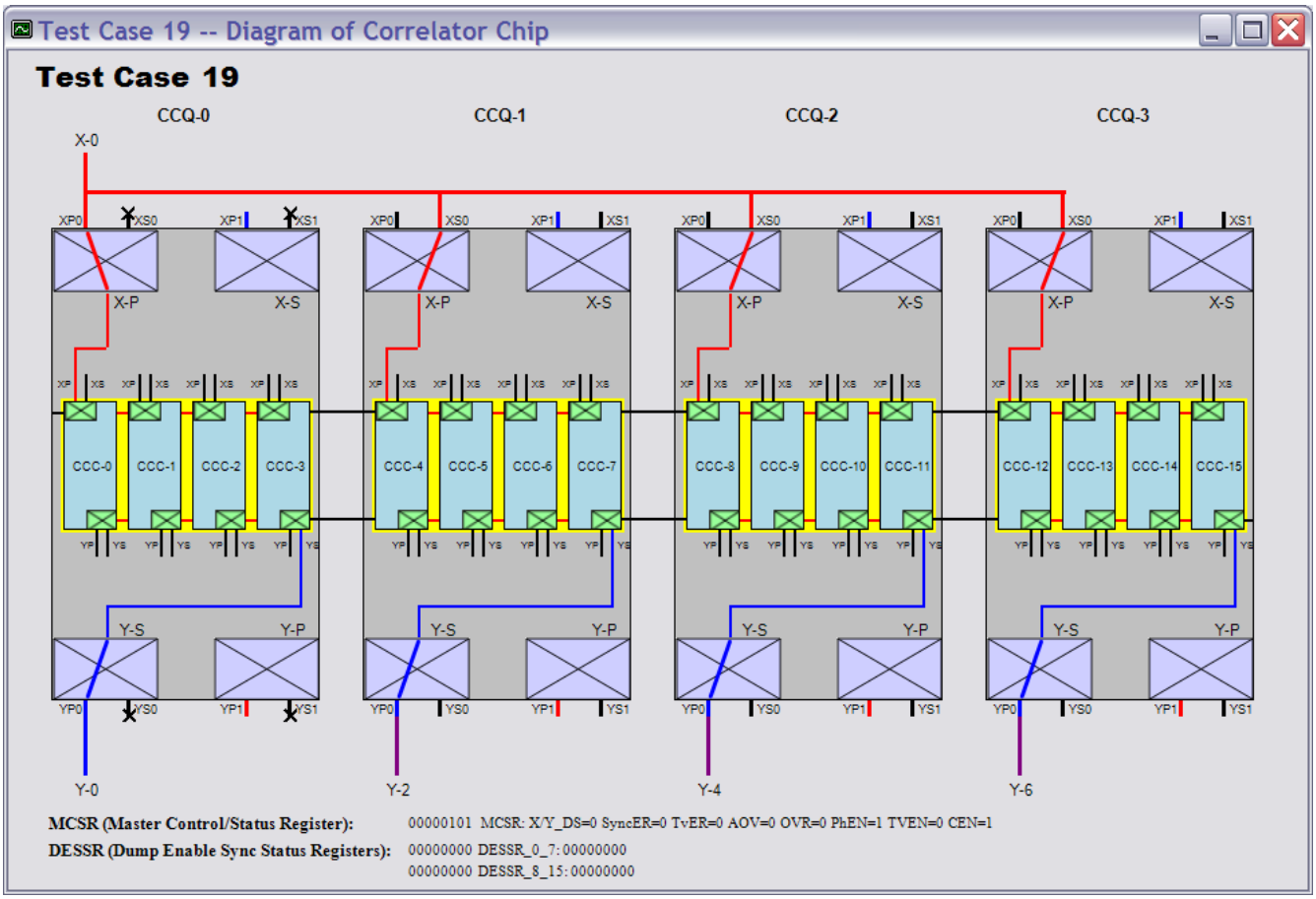
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 12000;

//-----------------------------------------------------------------------------------------------------------
```

4.20 Test case 19




```
// ********************************************************************************************************
// File name : tc19_jun14-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc19_mar15-04_v1.v     Mar 15, 2004    1                creation
// tc19_aug24-04_v2.v     Aug 24, 2004    2                modify for new test bench, modify data ID and
//                                                         Y-TIMESTAMP, 2 dump on Y
// tc19_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//tc19_jun14-05_v3.v      Jun 14, 2005    3                increase dump interval
//
// ********************************************************************************************************
//
//
// ********************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 19;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 19:
# ver: v3,05jun14
# Synopsis:   CONCAT CELLS IN QUADS, MIXED EVEN INPUTS, 2 DUMP ON Y
# Description:
#              Purpose of Test:      independent X,Y quad switches, data-T.S. capture
#              Concatenation:        0(0:3)0, 0(4:7)2, 0(8:11)4, 0(12:15)6
#              Num of Dumps-input:   2-0, 2-2, 2-4, 2-6
#              Dump Control:         Y
#              Num samples/dump-inp: 12000 on active inputs
#              Distinct samp/dump:   12000
#              Active SE clocks:     X-0; Y-0,2,4,6
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   check data ID and timestamp capture
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b______0_______0_________0_______1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________1_________0_________0_________0_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

ts_fillsync = 2'b01;
ts_0 = 'haaaaaaaa;
ts_1 = 'h55555555;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 170;
y_station_id_mem[tc] = 85;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 10;
y_sbid_mem[tc] = 21;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

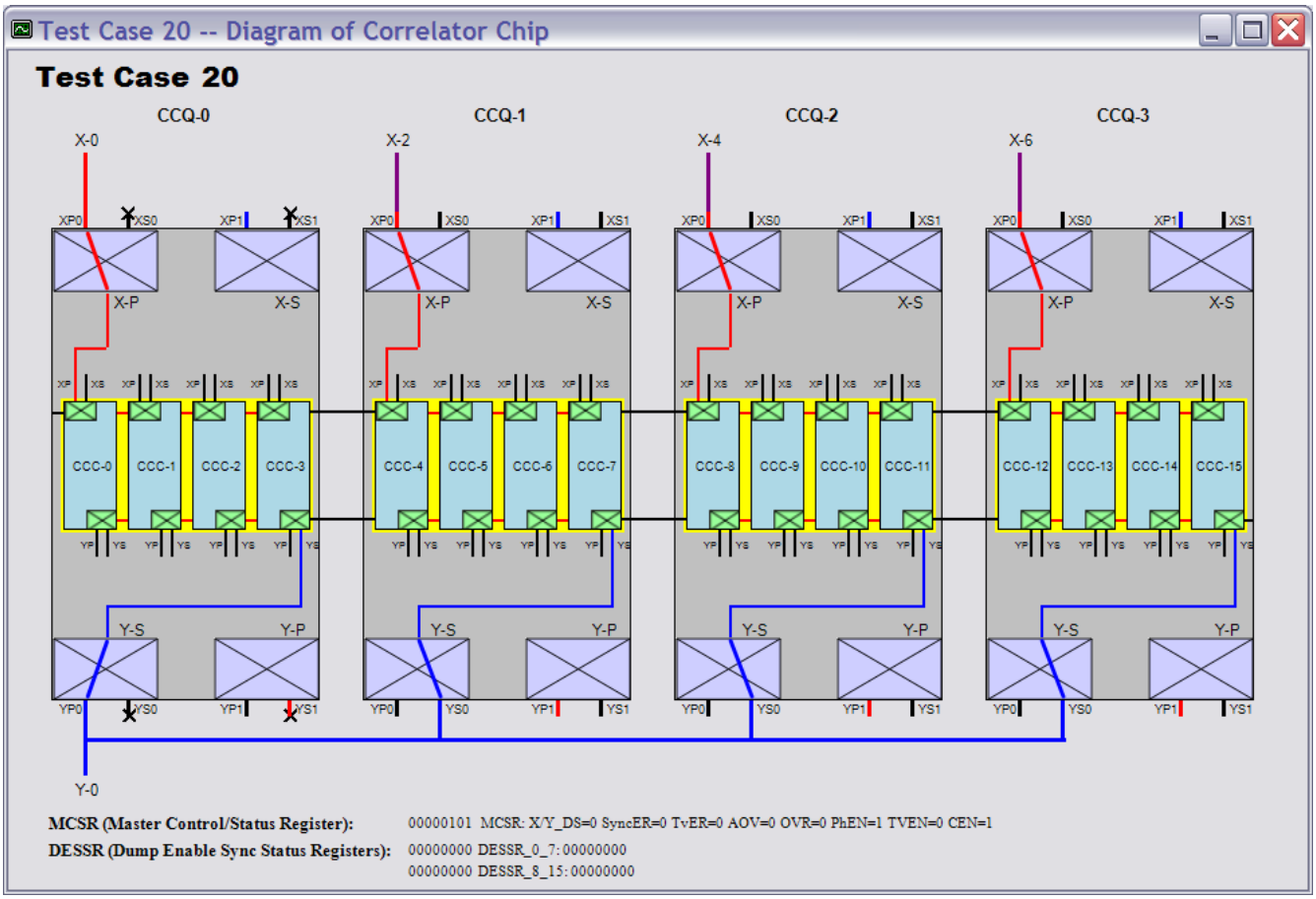
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 12000;

//-----------------------------------------------------------------------------------------------------------
```

4.21 Test case 20




```
// ********************************************************************************************************
// File name : tc20_jun14-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc20_mar15-04_v1.v     Mar 15, 2004    1                creation
// tc20_aug24-04_v2.v     Aug 24, 2004    2                modify for new test bench, modify data ID and
//                                                         Y-TIMESTAMP, 2 dump on Y
// tc20_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc20_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
//
// ********************************************************************************************************
//
//
// ********************************************************************************************************
```

```
// ***************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***************************************************************************************************************
// ***************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 20;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 20:
# ver: v3,05jun14
# Synopsis:   CONCAT CELLS IN QUADS, MIXED EVEN INPUTS, 2 DUMP ON Y
# Description:
#              Purpose of Test:      independent X,Y quad switches, data-T.S. capture
#              Concatenation:        0(0:3)0, 2(4:7)0, 4(8:11)0, 6(12:15)0
#              Num of Dumps-input:   2-0, 2-2, 2-4, 2-6
#              Dump Control:         Y
#              Num samples/dump-inp: 12000 on active inputs
#              Distinct samp/dump:   12000
#              Active SE clocks:     X-0,2,4,6; Y-0
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   check data ID and timestamp capture
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b______0________1__________0________0_________1_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1__________0________0_________0_________1_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {70'bx,2'b0};
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

ts_fillsync = 2'b01;
ts_0 = 'h55555555;
ts_1 = 'haaaaaaaa;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 170;
y_station_id_mem[tc] = 85;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 10;
y_sbid_mem[tc] = 21;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

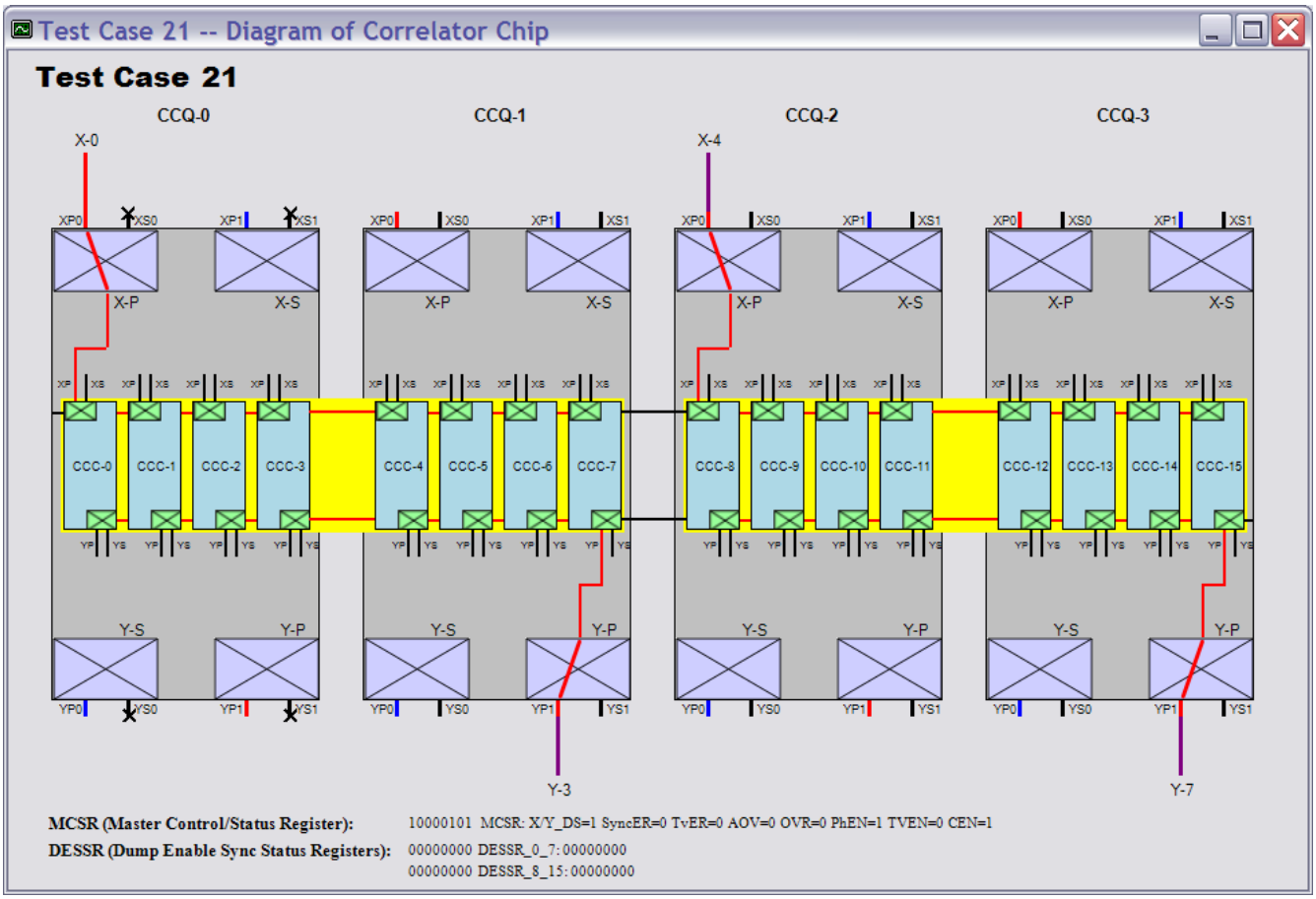
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 12000;

//-----------------------------------------------------------------------------------------------------------
```

4.22 Test case 21




```
// ********************************************************************************************************
// File name : tc21_jun14-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc21_mar15-04_v1.v     Mar 15, 2004    1                creation
// tc21_jul25-04_v2.v     Jul 25, 2004    2                modify for new test bench, multi-dump
// tc21_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc21_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
//
//
// ********************************************************************************************************
//
//
// ********************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 21;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 21:
# ver: v3,05jun14
# Synopsis:   CONCAT 2 QUADS, MIX INP, MULT DUMP ON X, MOD 2 CLK ON 1 QUAD PAIR
# Description:
#              Purpose of Test:      quad concatentation
#              Concatenation:        0(0:7)3, 4(8:15)7
#              Num of Dumps-input:   2-0, 2-3, 4-4, 4-7
#              Dump Control:         X
#              Num samples/dump-inp: 24000-0, 24000-3, 12000-4, 12000-7
#              Distinct samp/dump:   12000-0, 12000-3, 12000-4, 12000-7
#              Active SE clocks:     X-0,4; Y-3,7
#              Modulo se_clk gen:    MOD 2 CLK on X-0, Y-3
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b______0________0__________0________0__________0__________0__________0__________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0__________0__________0__________0__________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______1_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b_____1_______0_______0_______0_______1_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______1;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______1_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______1_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b_____1_______0_______0_______0_______1_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};


// these are dummies
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 12000;
dump_period_mem3[tc] = 12000;
dump_period_mem4[tc] = 6000;
dump_period_mem7[tc] = 6000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

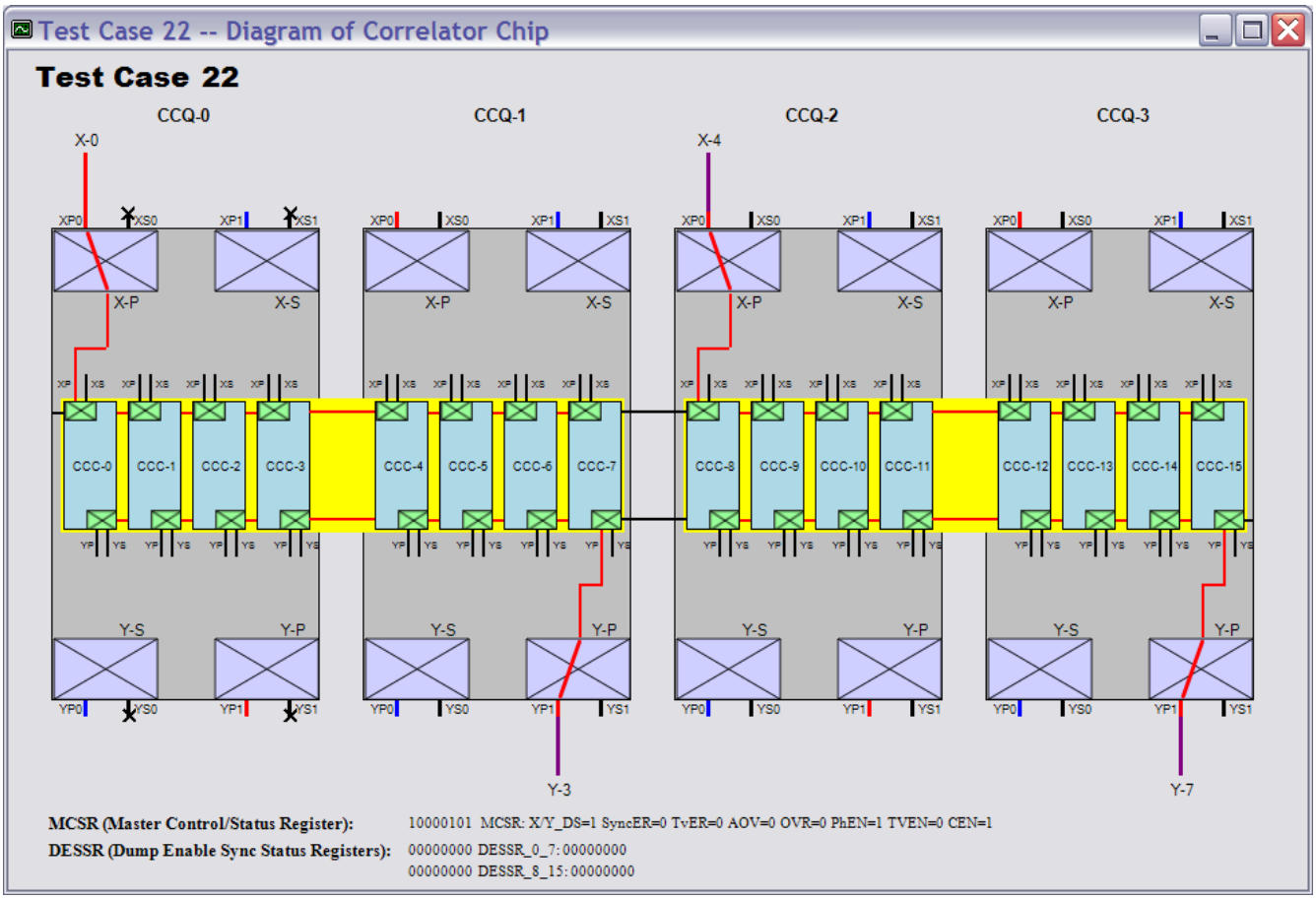
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 24000;

//-----------------------------------------------------------------------------------------------------------
```

4.23 Test case 22




```
// ********************************************************************************************************************
// File name : tc22_jul24-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc22_mar15-04_v1.v     Mar 15, 2004    1                creation
// tc22_jul25-04_v2.v     Jul 25, 2004    2                modify for new test bench, multi-dump
// tc22_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc22_jul24-05_v3.v     Jul 24, 2005    3                change dump interval to 12000 samples
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ***************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***************************************************************************************************************
// ***************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 22;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 22:
# ver: v3,05jul24
# Synopsis:   CONCAT 2 QUADS, MIXED INP, DUMP ON X, MOD 2 CLK ON 1 QUAD PAIR
# Description:
#              Purpose of Test:      quad concatentation
#              Concatenation:        0(0:7)3, 4(8:15)7
#              Num of Dumps-input:   2-0, 2-3, 2-4, 2-7
#              Dump Control:         X
#              Num samples/dump-inp: 24000-0, 24000-3, 24000-4, 24000-7
#              Distinct samp/dump:   24000-0, 24000-3, 12000-4, 12000-7
#              Active SE clocks:     X-0,4; Y-3,7
#              Modulo se_clk gen:    MOD 2 on 4,7
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b______0________0__________0________0__________0__________0__________0__________0;
ccqr_2_3_mem[tc] = 8'b______0________0__________0________0__________0__________0__________0__________0;

//-------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______1_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b_____1_______0_______0_______0_______1_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______1_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__1_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______1_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b_____1_______0_______0_______0_______1_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 12000;
dump_period_mem3[tc] = 12000;
dump_period_mem4[tc] = 12000;
dump_period_mem7[tc] = 12000;

//-------------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 24000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------
```

4.24 Test case 23

This region is intentionally left blank.

```
// ********************************************************************************************************
// File name : tc23_aug07-05_v4.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V4
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc23_mar08-04_v1.v     Mar 08, 2004    1                creation
// tc23_aug25-04_v2.v     Aug 25, 2004    2                modify for new test bench
// tc23_aug26-04_v2.v     Aug 26, 2004    2                fix error in description
// tc23_apr22-05_v3.v     Apr 22, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc23_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
// tc23_aug07-05_v4.v     Aug  7, 2005    4                apply uniform dump interval, SE_CLK on inp. 7
//
// ********************************************************************************************************
```

```
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 23;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 23:
# ver: v4,05aug07
# Synopsis:   MIX OF DEFINED AND UNDEFINED CELL INPUTS, DUMP ON X
# Description:
#              Purpose of Test:      mixed concat.
#              Concatenation:        7(0:2)0, 3(6:7)3, 4(8)4
#              Num of Dumps-input:   2-7, 2-3 , 2-4
#              Dump Control:         X
#              Num samples/dump-inp: 12000-3, 12000-4, 12000-5, 12000-7
#              Distinct samp/dump:   12000-3, 12000-4, 12000-5, 12000-7
#              Active SE clocks:     0, 3, 4, 5, 7
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   apply Y-X offset on DUMP_SYNC.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______1______0;
cccscr_2_3_mem[tc] = 8'b___0______0______1______1______1______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______1______0______0______0______0______0______0;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___0______0______1______1______1______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______0______0______0______0______0______1______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_14_15_mem[tc] = 8'b_0______0______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                          B7       B6         B5       B4         B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________1__________0________1_________0_________0_________1_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1__________1________1_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______0_______1_______1_______1_______0_______0_______1;
y_se_clk_mem[tc] = 8'b_____1_______0_______1_______1_______1_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______1_______1_______1_______0_______0_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______1_______1_______1_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 2;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

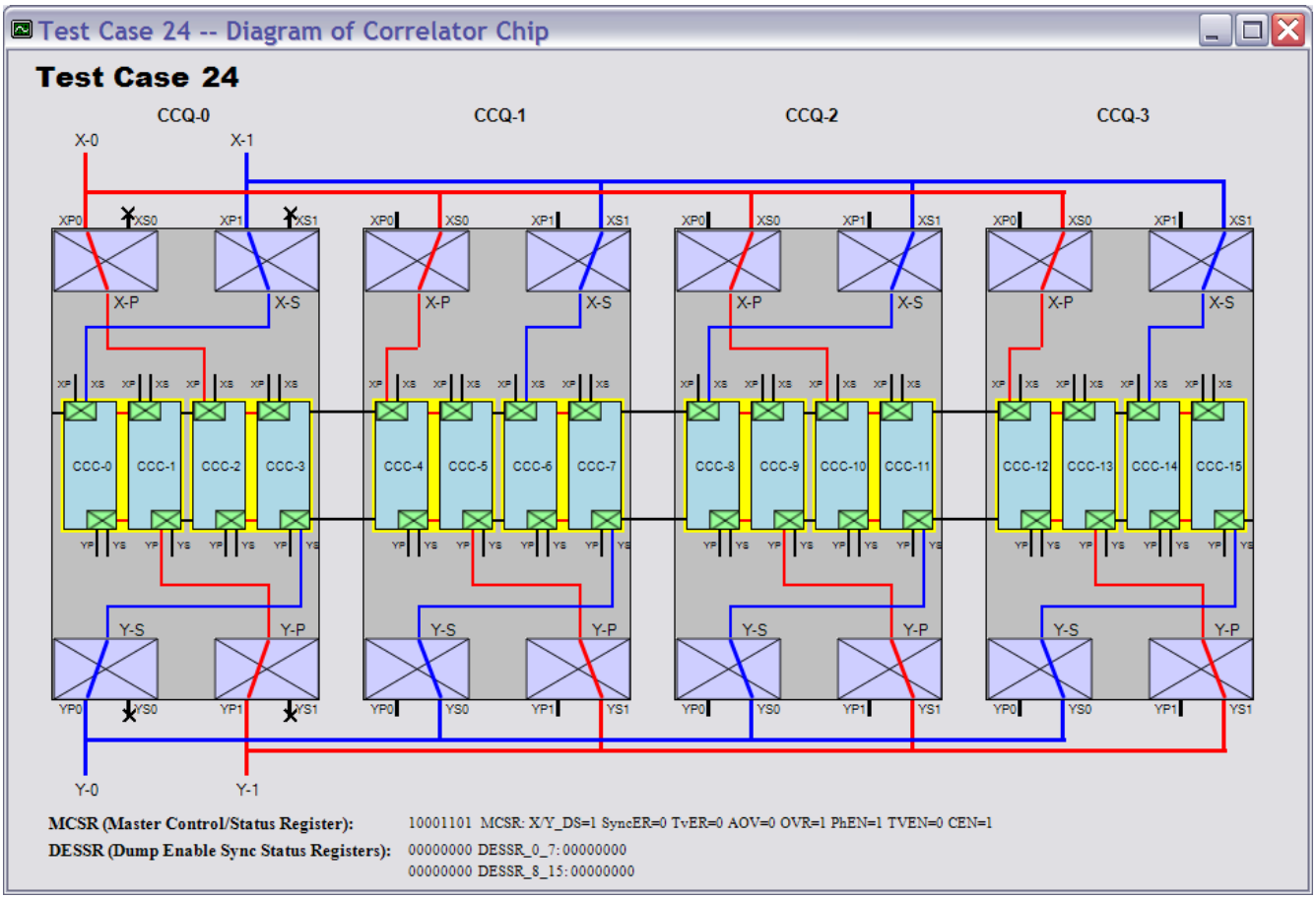
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 12000;

//-----------------------------------------------------------------------------------------------------------
```

4.25 Test case 24




```
// ********************************************************************************************************************
// File name : tc24_apr23-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc24_mar13-04_v1.v     Apr 29, 2004    1                creation
// tc24_jul27-04_v2.v     Jul 27, 2004    2                modify for new test bench
// tc24_nov17-04_v2.v     Nov 17, 2004    2                decrease dump rate
// tc24_apr23-05_v3.v     Apr 23, 2005    3                upgrade to new test bench
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 24;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 24:
# ver: v3,05apr25
# Synopsis:   INP 0,1 ACTIVE, 4 POL PRODUCTS IN CELL PAIRS, 10 DUMPS ON X
# Description:
#              Purpose of Test:     all pol products; mult dump in cell pairs;
#                                SE CLK on inactive inp
#              Concatenation:       1(0:1)1, 0(2:3)0, 0(4:5)1, 1(6:7)0, 1(8:9)1,
#                                0(10:11)0, 0(12:13)1, 1(14:15)0
#              Num of Dumps-input:   10-0, 10-1
#              Dump Control:         X
#              Num samples/dump-inp: 8000-0, 8000-1
#              Distinct samp/dump:   8000
#              Active SE clocks:     0,1,2,3
#              Modulo se_clk gen:    mod2 on inputs 2,3
#              Compare result with:  compare between cells with like products
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_2_3_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_4_5_mem[tc] = 8'b___0______1______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___1______0______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_10_11_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_12_13_mem[tc] = 8'b_0______1______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______1______0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1__________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1__________1________1_________1_________1_________1_________1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______1_______1_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______1_______1_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits
```

```
// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 4000;
dump_period_mem1[tc] = 4000;
dump_period_mem2[tc] = 4000;
dump_period_mem3[tc] = 4000;
dump_period_mem4[tc] = 4000;
dump_period_mem5[tc] = 4000;
dump_period_mem6[tc] = 4000;
dump_period_mem7[tc] = 4000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

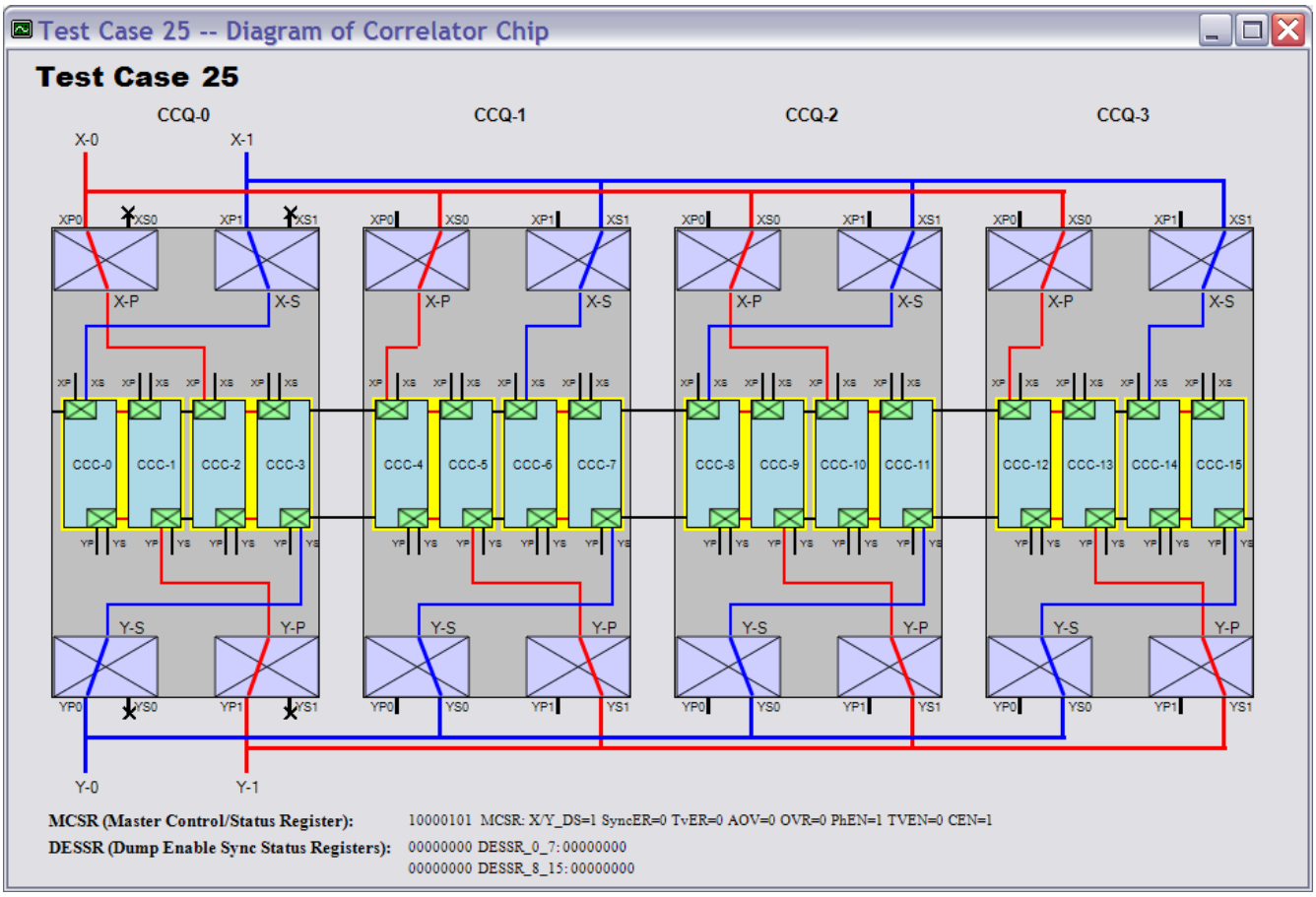
test_duration_mem[tc] = 40000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// *****************************************************************************************************************
```

4.26 Test case 25




```
// ********************************************************************************************************************
// File name : tc25_jun14-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc25_mar13-04_v1.v     Apr 29, 2004    1                creation
// tc25_jul27-04_v2.v     Jul 27, 2004    2                new test bench, SE_CLK mod 2
// tc25_apr23-05_v3.v     Apr 23, 2005    3                upgrade to new test bench
// tc25_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 25;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 25:
# ver: v3,05jun14
# Synopsis:   INP 0,1 ACTIVE, 4 POL PRODUCTS IN CELL PAIRS, 10 DUMPS ON X
# Description:
#              Purpose of Test:     all pol products; mult dump in cell pairs;
#                                SE CLK on inactive inp
#              Concatenation:       1(0:1)1, 0(2:3)0, 0(4:5)1, 1(6:7)0, 1(8:9)1,
#                                0(10:11)0, 0(12:13)1, 1(14:15)0
#              Num of Dumps-input:   10-0, 10-1
#              Dump Control:         X
#              Num samples/dump-inp: 12000-0, 12000-1
#              Distinct samp/dump:   6000
#              Active SE clocks:     0,1
#              Modulo se_clk gen:    mod 2 on 0,1
#              Compare result with:  compare between cells with like products
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____1_____0;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_4_5_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____1_____0;
cccscr_8_9_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____1;
cccscr_12_13_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____1_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5        B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______1;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits
```

```
// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

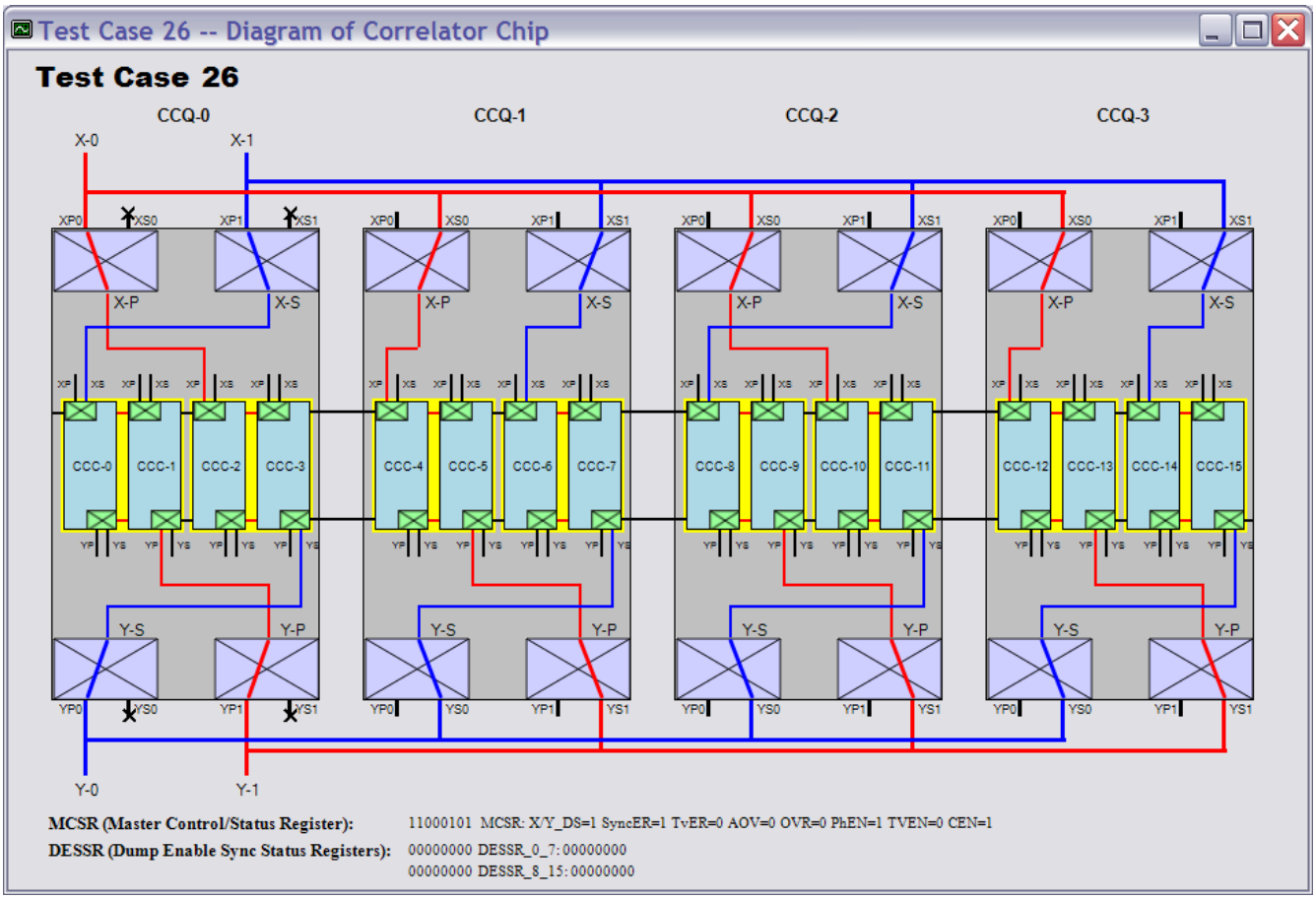
test_duration_mem[tc] = 60000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// *****************************************************************************************************************
```

4.27 Test case 26




```
// ********************************************************************************************************************
// File name : tc26_jun14-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc26_aug22-04_v1.v     Aug 22, 2004    1                creation
// tc26_apr23-05_v2.v     Apr 23, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc26_jun14-05_v2.v     Jun 14, 2005    2                increase dump interval
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 26;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 26:
# ver: v2,05jun14
# Synopsis:   INP 0,1 ACTIVE, X,Y-TIMESTAMP errors, 2 DUMPS ON X
# Description:
#              Purpose of Test:     timestamp sync. error detection
#              Concatenation:       1(0:1)1, 0(2:3)0, 0(4:5)1, 1(6:7)0, 1(8:9)1,
#                                0(10:11)0, 0(12:13)1, 1(14:15)0
#              Num of Dumps-input:   2-0, 2-1
#              Dump Control:         X
#              Num samples/dump-inp: 12000-0, 12000-1
#              Distinct samp/dump:   12000-0, 12000-1
#              Active SE clocks:     0,1,2,3
#              Modulo se_clk gen:    none
#              Compare result with:  check X,Y-TIMESTAMP
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_2_3_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_4_5_mem[tc] = 8'b___0______1______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___1______0______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_10_11_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_12_13_mem[tc] = 8'b_0______1______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______1______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) on off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______1_______1_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______1_______1_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b11;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b11;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

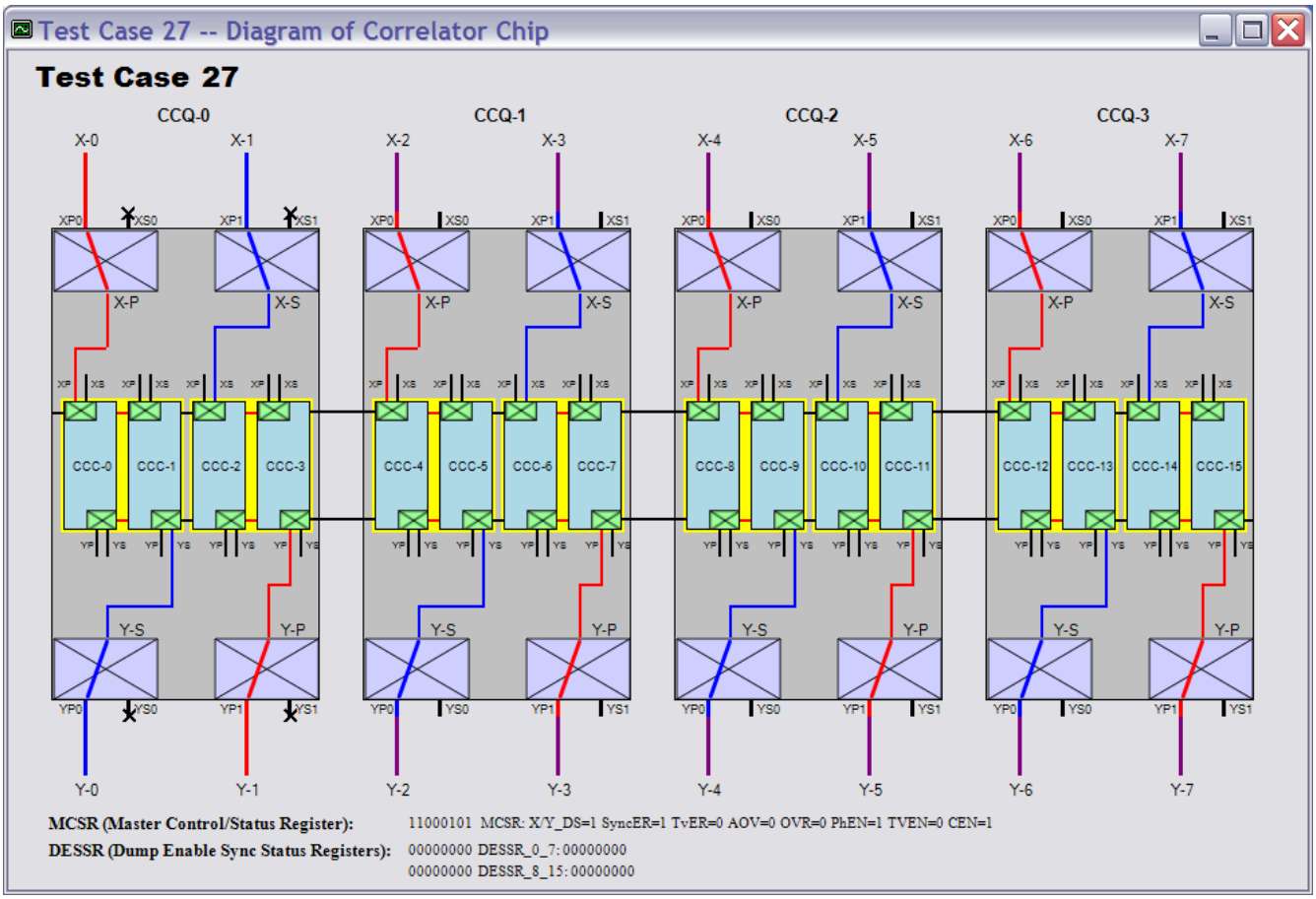
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 12000;

//-----------------------------------------------------------------------------------------------------------
```

4.28 Test case 27




```
// ********************************************************************************************************
// File name : tc27_apr23-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME               DATE            VERSION          COMMENTS
// tc27_aug24-04_v1.v      Aug 24, 2004    1                creation
// tc27_aug28-04_v1.v      Aug 28, 2004    1                add all inputs, SE_CLK disabled on odd
// tc27_apr23-05_v2.v      Apr 23, 2005    2                upgrade to new test bench
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 27;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 27:
# ver: v2,05apr23
# Synopsis:   CONCAT IN PAIRS, ALL INP, DISABLE ODD Y-SE_CLK 1, 1 DUMP ON X
# Description:
#              Purpose of Test:      missing SE_CLK detection
#              Concatenation:        0(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)6, 7(14:15)7
#              Num of Dumps-input:   1 on all
#              Dump Control:         X
#              Num samples/dump-inp: 10000 on all
#              Distinct samp/dump:   10000 on all
#              Active SE clocks:     X: 0,1,2,3,4,5,6,7; Y: 0,2,4,6
#              Modulo se_clk gen:    none
#              Compare result with:  check SE_CLK error reporting
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______0______0______0______0______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______0______0______0______0______1______0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits
```

```
// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

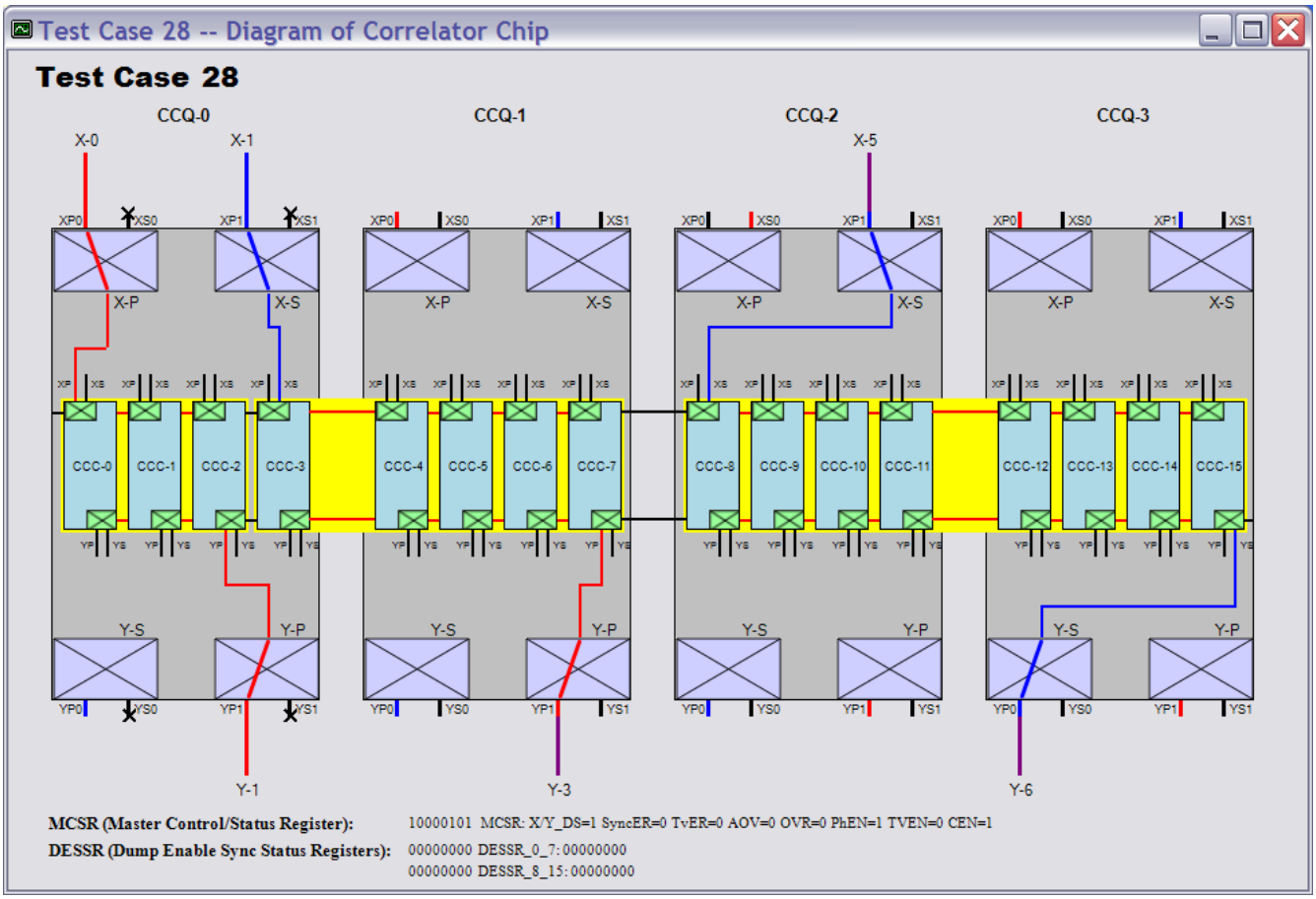
test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***************************************************************************************************************
```

4.29 Test case 28




```
// ****************************************************************************************************************
// File name : tc28_jun14-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ****************************************************************************************************************
// ************************    REVISION HISTORY    ****************************************************************
// ****************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc28_may30-04_v1.v     May 30, 2004    1                creation
// tc28_aug26-04_v2.v     Aug 26, 2004    2                modify for new test bench
// tc28_apr23-05_v3.v     Apr 23, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc28_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
//
//
// ****************************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                  BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 28;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 28:
# ver: v3,05jun14
# Synopsis:   3, 5, 8 CELL CONCAT WITH 2 DUMPS ON X
# Description:
#              Purpose of Test:      mixed length cell chains
#              Concatenation:        0(0:2)1, 1(3:7)3, 5(8:15)6
#              Num of Dumps-input:   2-0, 2-1, 2-5
#              Dump Control:         X
#              Num samples/dump-inp: 12000-0, 12000-1, 12000-5
#              Distinct samp/dump:   12000-0, 12000-1, 12000-5
#              Active SE clocks:     0, 1, 3, 5, 6
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:
#
# ------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______0______1______0______0______1______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______1______0;
cccscr_10_11_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______0______0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________1;

//-----------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______1_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b_____0_______1_______0_______0_______1_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______1_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b_____0_______1_______0_______0_______1_______0_______1_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 6000;
dump_period_mem3[tc] = 6000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 6000;
dump_period_mem7[tc] = 6000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

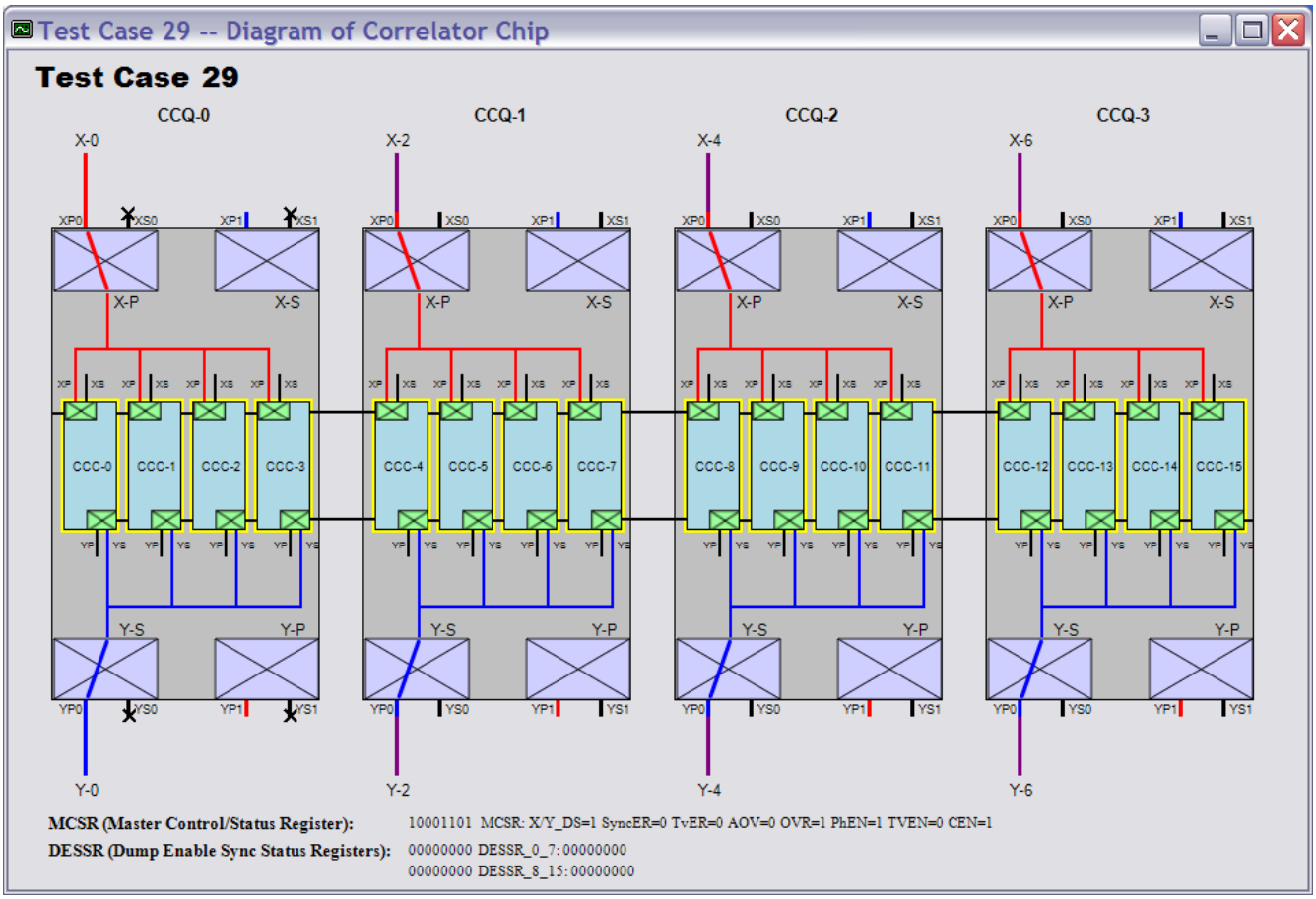
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 12000;

//-----------------------------------------------------------------------------------------------------------
```

4.30 Test case 29




```
// ********************************************************************************************************************
// File name : tc29_apr11-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc29_aug22-04_v1.v     Aug 22, 2004    1                creation
// tc29_apr11-05_v2.v     Apr 11, 2004    2                add verification mode option;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 29;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 29:
# ver: v2,04apr11
# Synopsis:   INDEPENDENT CELLS, 4 INPUTS, 4 DUMP RATES ON X
# Description:
#              Purpose of Test:      check overrun behaviour
#              Concatenation:        0(0)0 , 0(1)0, 0(2)0, 0(3)0, 2(4)2, 2(5)2, 2(6)2,
#                                 2(7)2 , 4(8)4, 4(9)4, 4(10)4, 4(11)4, 6(12)6, 6(13)6,
#                                 6(14)6 , 6(15)6
#              Num of Dumps-input:   16-0, 8-2, 4-4, 2-6
#              Dump Control:         X
#              Num. of samples/dump: 3000-1, 6000-2, 12000-4, 24000-6
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with:  expected, check overrun bit
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_2_3_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_4_5_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_6_7_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_8_9_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_10_11_mem[tc] = 8'b_1______0______0______1______1______0______0______1;
cccscr_12_13_mem[tc] = 8'b_1______0______0______1______1______0______0______1;
cccscr_14_15_mem[tc] = 8'b_1______0______0______1______1______0______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______1_______0_______1_______0_______1_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//            000 = First dump of data into LTA. Just save data in LTA bin.
//            001 = Add to existing LTA data and save in LTA.
//            010 = Last dump: add data to LTA, and flag data as ready for readout.
```

```
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 1500;
dump_period_mem1[tc] = 1500;
dump_period_mem2[tc] = 3000;
dump_period_mem3[tc] = 3000;
dump_period_mem4[tc] = 6000;
dump_period_mem5[tc] = 6000;
dump_period_mem6[tc] = 12000;
dump_period_mem7[tc] = 12000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
```

```
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION

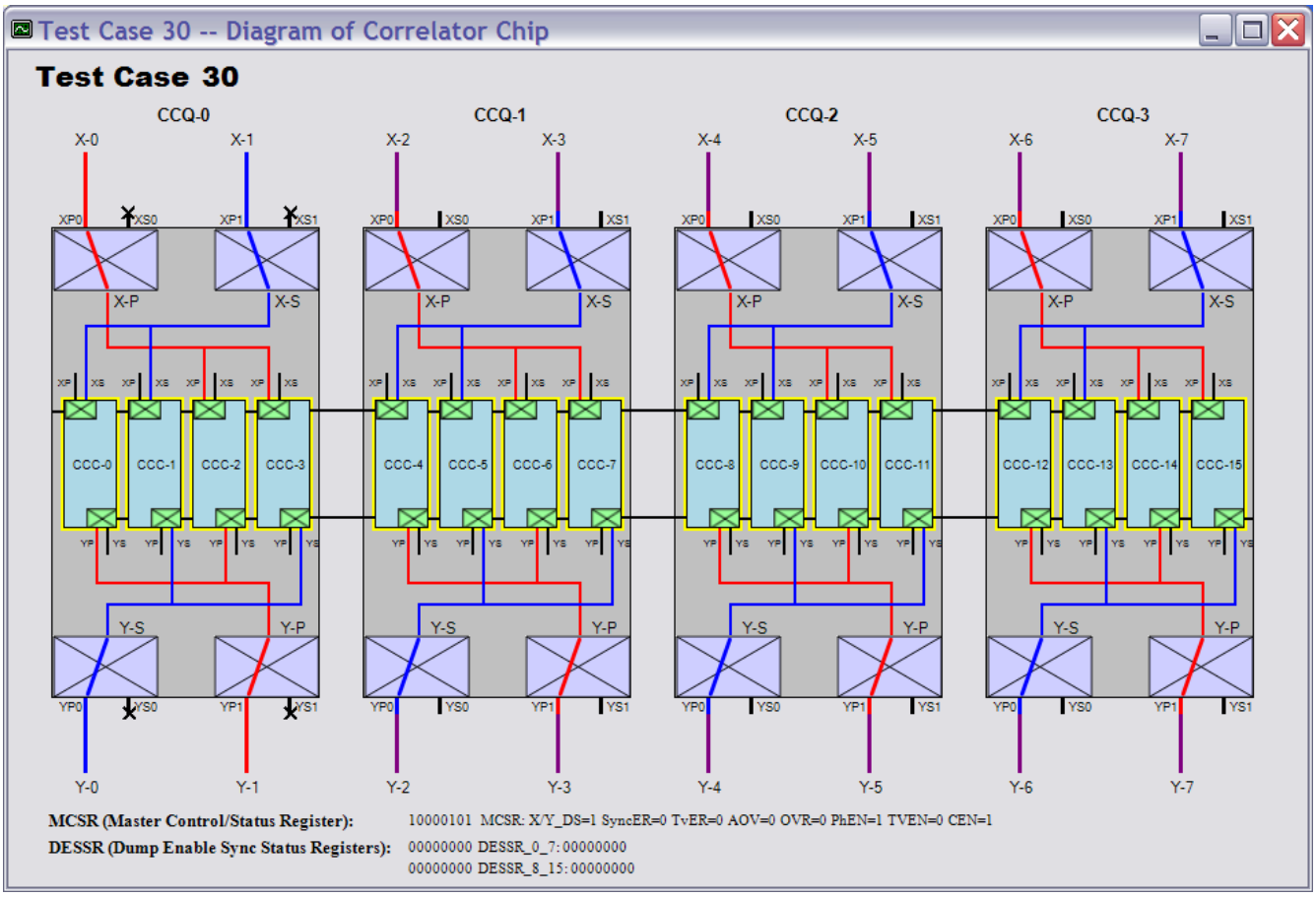
test_duration_mem[tc] = 24000;

//------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ********************************************************************************************************
```

4.31 Test case 30




```
// ********************************************************************************************************************
// File name : tc30_jun14-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc30_may14-04_v1.v     May 14, 2004    1                creation
// tc30_jul26-04_v2.v     Jul 26, 2004    2                modify for new test bench, increase dump count
// tc30_apr23-05_v3.v     Apr 23, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc30_jun14-05_v3.v     Jun 14, 2005    3                increase dump interval
//
//
// ********************************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                    BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 30;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 30:
# ver: v3,05jun14
# Synopsis:   NO CONCAT., ALL POLN, 8 INPUTS, 4 DUMP RATES ON X
# Description:
#              Purpose of Test:      Normal dump and readout with multiple
#                                 dump rates and four pol'n
#              Concatenation:        1(0)1, 1(1)0, 0(2)1, 0(3)0, 3(4)3, 3(5)2, 2(6)3,
#                                 2(7)2, 5(8)5, 5(9)4, 4(10)5, 4(11)4, 7(12)7, 7(13)6,
#                                 6(14)7, 6(15)6
#              Num of Dumps-input:   16-0, 16-1, 8-2, 8-3, 4-4, 4-5, 2-6, 2-7
#              Dump Control:         X
#              Num samples/dump-inp: 12000-0, 12000-1, 24000-2, 24000-3, 48000-4, 48000-5,
#                                 96000-6, 96000-7
#              Distinct samp/dump:  12000-0, 12000-1, 24000-2, 24000-3, 48000-4, 48000-5,
#                                 96000-6, 96000-7
#              Active SE clocks:     0, 1, 2, 3, 4, 5, 6 , 7
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   no data synch errors generated;
#                                 no Y-X offset on DUMP_SYNC.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_4_5_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_8_9_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;
cccscr_12_13_mem[tc] = 8'b_1_____0_____1_____0_____0_____1_____1_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------------
//                            B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid

// these are dummies since X controls dumping

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 6000;
dump_period_mem1[tc] = 6000;
dump_period_mem2[tc] = 12000;
dump_period_mem3[tc] = 12000;
```

```
dump_period_mem4[tc] = 24000;
dump_period_mem5[tc] = 24000;
dump_period_mem6[tc] = 48000;
dump_period_mem7[tc] = 48000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

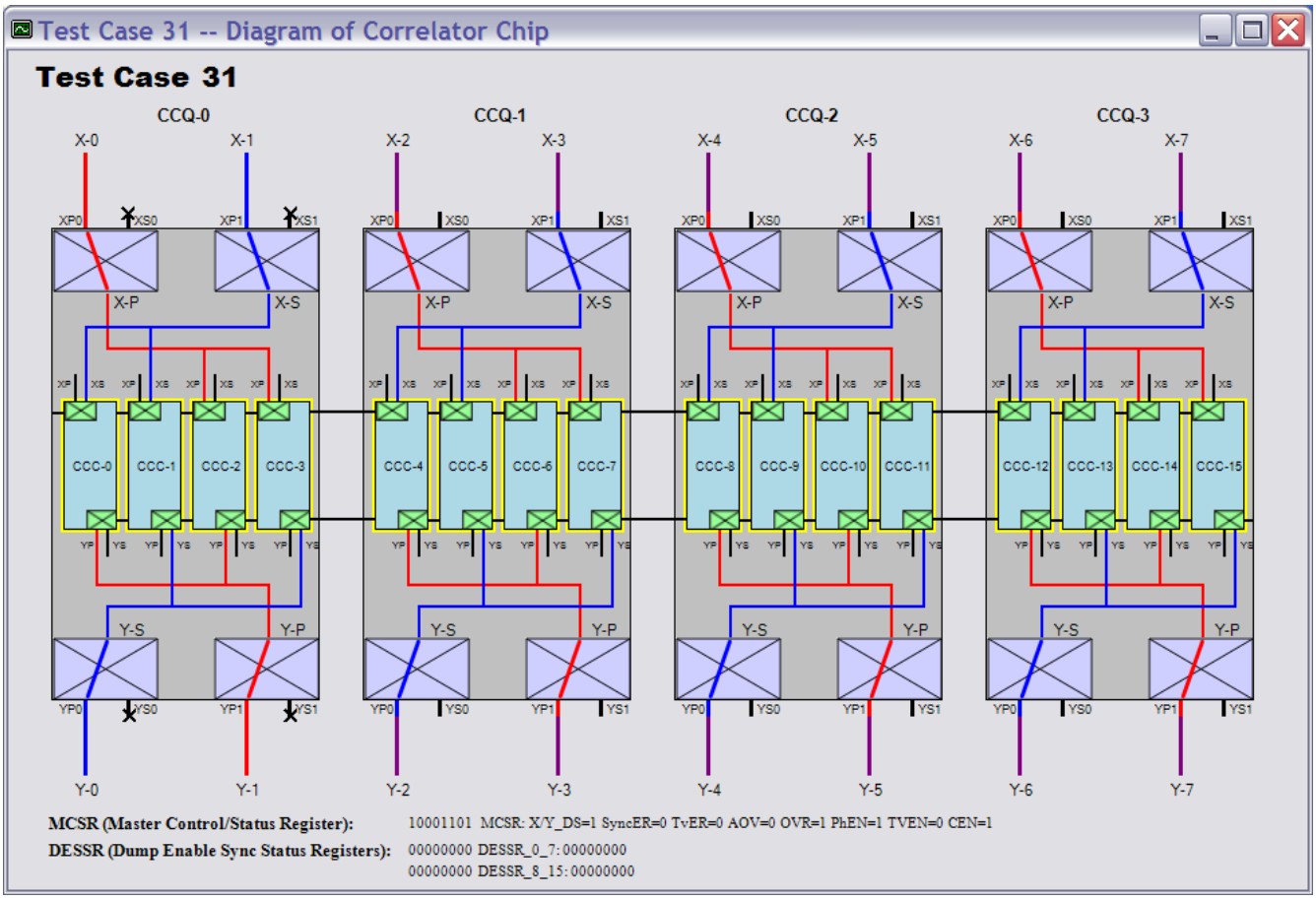
// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 96000;
```

4.32 Test case 31




```
// ********************************************************************************************************************
// File name : tc31_apr23-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME               DATE            VERSION          COMMENTS
// tc31_may15-04_v1.v      May 15, 2004    1                creation
// tc31_jul26-04_v2.v      Jul 26, 2004    2                modify for new test bench, increase dump count
//                                                          and decrease dump size to overrun
// tc31_sep16-04_v2.v      Sep 16, 2004    2                fix description typos
// tc31_apr23-05_v3.v      Apr 23, 2005    3                upgrade to new test bench
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 31;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 31:
# ver: v3,05apr23
# Synopsis:   NO CONCAT., ALL POLN, 8 INPUTS, 4 DUMP RATES ON X
# Description:
#              Purpose of Test:      Apply different dump rates between 4 inputs, multiple
#                           dumps and quad concatenation, overruns
#              Concatenation:        1(0)1, 1(1)0, 0(2)1, 0(3)0, 3(4)3, 3(5)2, 2(6)3,
#                                 2(7)2, 5(8)5, 5(9)4, 4(10)5, 4(11)4, 7(12)7, 7(13)6,
#                                 6(14)7, 6(15)6
#              Num of Dumps-input:   32-0, 32-1, 16-2, 16-3, 8-4, 8-5, 4-6, 4-7
#              Dump Control:         X
#              Num samples/dump-inp: 4000-0, 4000-1, 8000-2, 8000-3, 16000-4, 16000-5,
#                                 32000-6, 32000-7
#              Distinct samp/dump:   4000-0, 4000-1, 8000-2, 8000-3, 16000-4, 16000-5,
#                                 32000-6, 32000-7
#              Active SE clocks:     0, 1, 2, 3, 4, 5, 6 , 7
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:   no data synch errors generated;
#                                 no Y-X offset on DUMP_SYNC.
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_4_5_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_8_9_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;
cccscr_12_13_mem[tc] = 8'b_1_____0_____1_____0_____0_____1_____1_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_____0_______0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_____0_______0_________0________0_________0_________0_________0_________0;
```

```
//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits
```

```
// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid

// these are dummies since X controls dumping

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 2000;
dump_period_mem1[tc] = 2000;
dump_period_mem2[tc] = 4000;
dump_period_mem3[tc] = 4000;
dump_period_mem4[tc] = 8000;
dump_period_mem5[tc] = 8000;
dump_period_mem6[tc] = 16000;
dump_period_mem7[tc] = 16000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.
```

```
sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

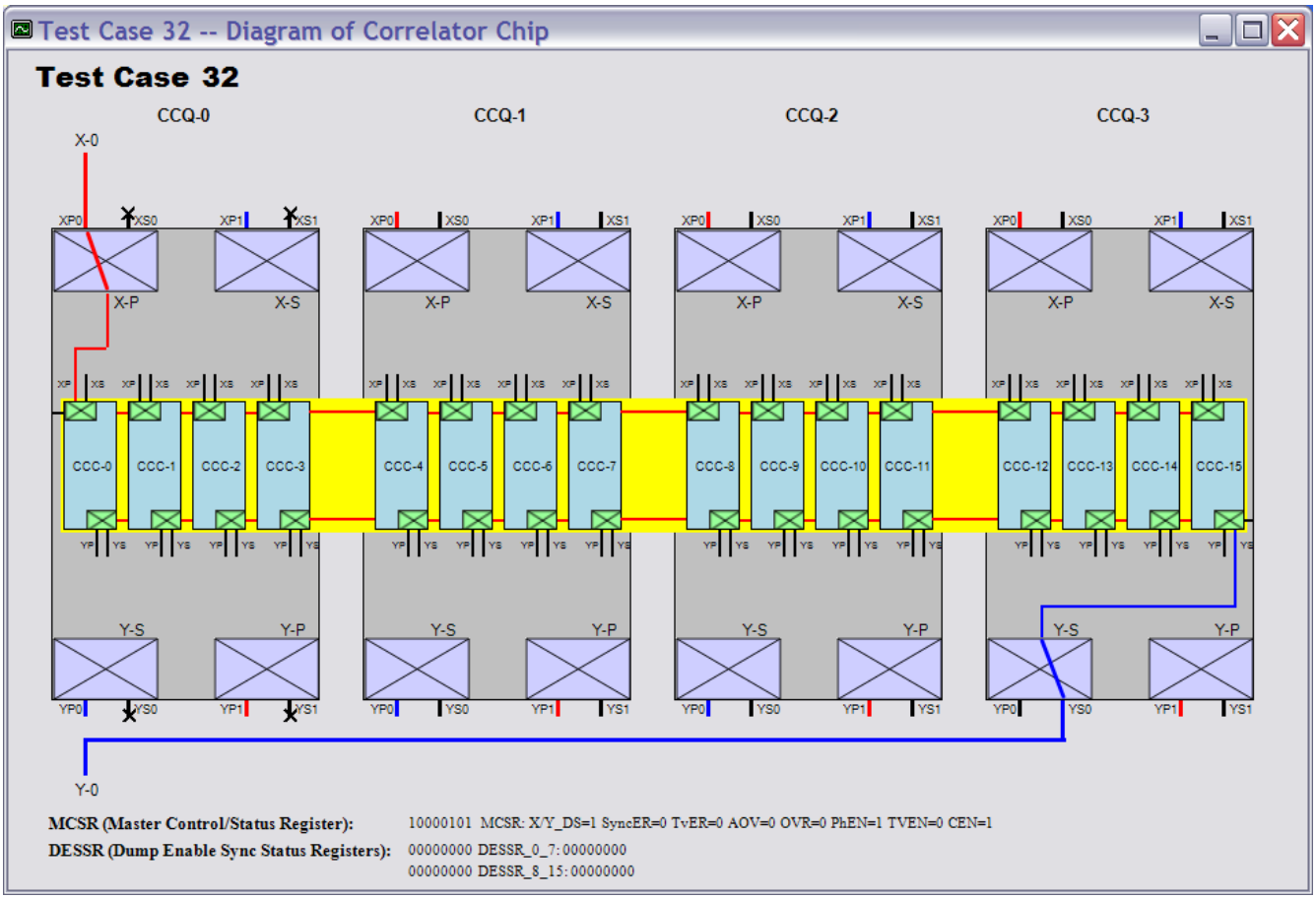
test_duration_mem[tc] = 64000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// *****************************************************************************************************************
```

4.33 Test case 32




```
// ********************************************************************************************************
// File name : tc32_apr19-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3 --- PRELIMINARY
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc32_jul04-04_v1.v     Jul 04, 2004    1 - PRELIM       creation
// tc32_aug22-04_v2.v     Aug 22, 2004    2                update for new test bench
// tc32_apr04-05_v3.v     Apr 04, 2005    3                first test bench release; reduce dump interval;
//                                                         increase X-Y SCHID offset
// tc32_apr19-05_v3.v     Apr 19, 2005    3                add verification only option
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 32;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 32:
# ver: v3,05apr19
# Synopsis: CONCATENATE ALL, X DUMP, INPUT 0 DATA ONLY
# Description:
#              Purpose of Test:      Cell concatenation and correlation;
#                                  short run for bit exact model comparison.
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 10000-0
#              Distinct samp/dump:   10000-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for CCinput0
#              Additional details:   negative Y-SCHID offset
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_6_7_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_10_11_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = -15;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = -30;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

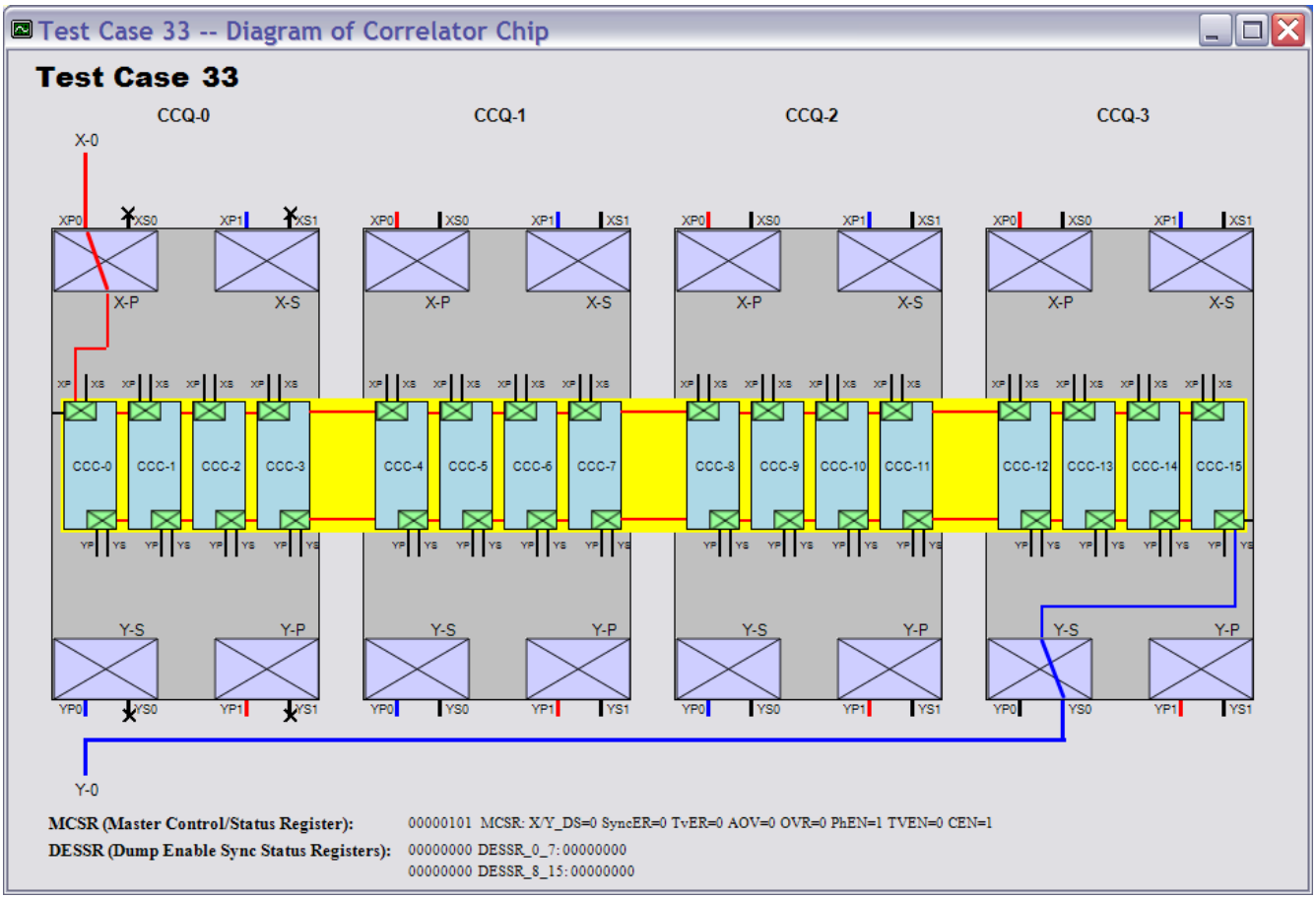
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
```

4.34 Test case 33




```
// ********************************************************************************************************************
// File name : tc33_apr19-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3 --- PRELIMINARY
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME               DATE            VERSION          COMMENTS
// tc33_jul05-04_v1.v      Jul 05, 2004    1 - PRELIM       creation
// tc33_aug22-04_v2.v      Aug 22, 2004    2                update for new test bench
// tc33_aug31-04_v2.v      Aug 31, 2004    2                change description of test vectors
// tc33_apr04-05_v3.v      Apr 04, 2005    3                first test bench release; reduce dump interval;
//                                                          increase X-Y SCHID offset
// tc33_apr19-05_v3.v      Apr 19, 2005    3                add verification only mode option
//
// ********************************************************************************************************************
//
```

```
//
// ***************************************************************************************************************
// ***************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***************************************************************************************************************
// ***************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 33;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 33:
# ver: v3,05apr19
# Synopsis: CONCATENATE ALL, Y DUMP, INPUT 0 DATA ONLY, MOD 2 SE CLK, SHORT RUN
# Description:
#              Purpose of Test:    Apply mod 2 clock; test cell concatenation and correlation;
#                                short run for bit exact model comparison.
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         Y
#              Num samples/dump-inp: 20000-0
#              Distinct samp/dump:   10000-0
#              Active SE clocks:     input 0 mod 2 SE_CLK
#              Modulo se_clk gen:    none
#              Compare result with:  behavioural simulation test vectors for CCinput0
#              Additional details:   positive Y-SCHID offset
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_6_7_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_10_11_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______1;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {70'bx,2'b0};
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 15;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 30;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

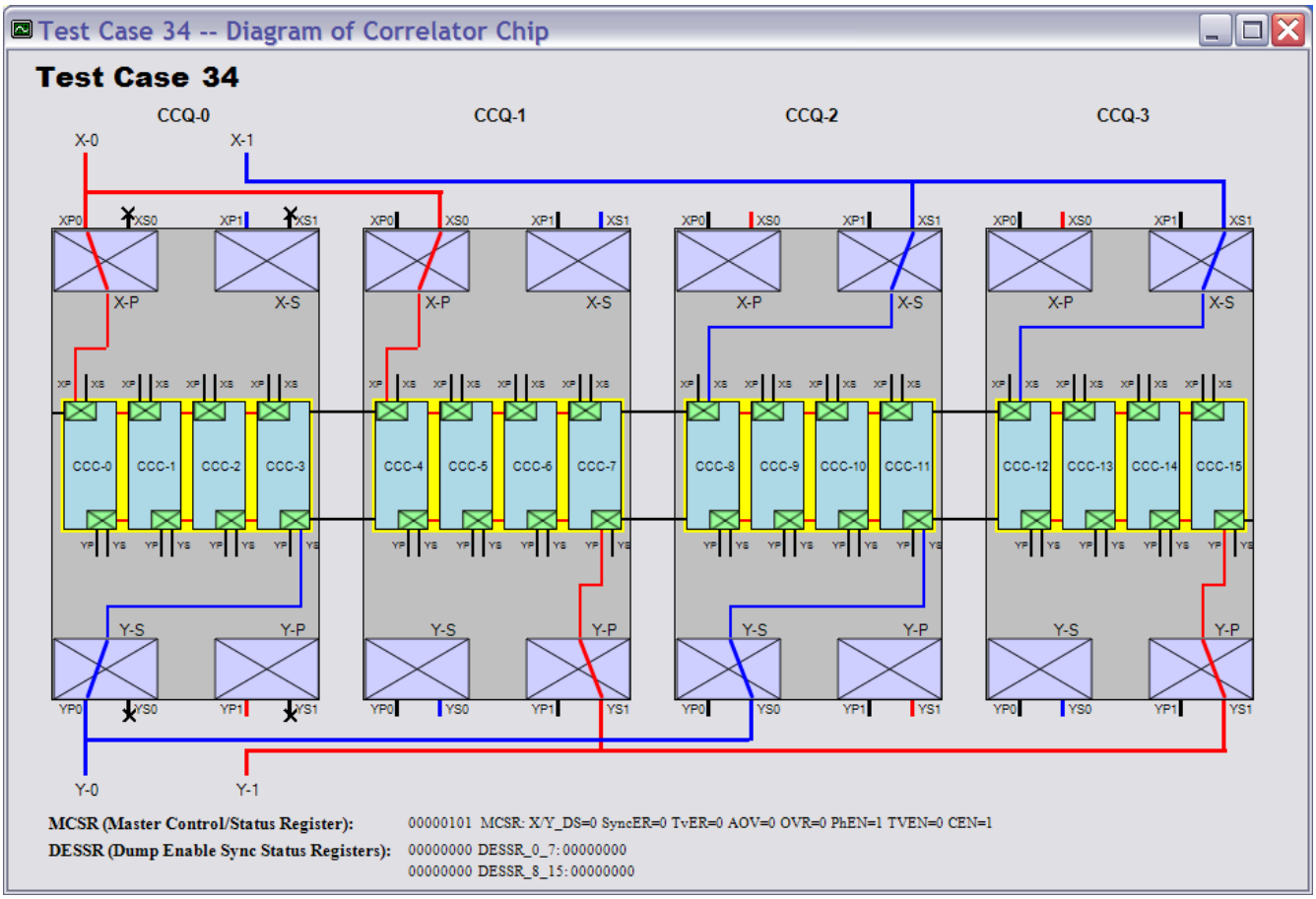
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------
```

4.35 Test case 34




```
// ********************************************************************************************************************
// File name : tc34_apr23-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc34_jul17-04_v1.v     Jul 17, 2004    1                creation
// tc34_apr23-05_v2.v     Apr 23, 2005    2                upgrade to new test bench
//
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 34;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 34:
# ver: v2,05apr23
# Synopsis:  CONCAT AS QUADS, INP 0,1 ALL POLN PRODUCTS, 1 DUMP ON Y
# Description:
#              Purpose of Test:      all poln products with quad concatenation
#              Concatenation:        0(0:3)0, 0(4:7)1, 1(8:11)0, 1(12:15)1
#              Num of Dumps-input:   1-0, 1-1
#              Dump Control:         Y
#              Num samples/dump-inp: 40000-0, 40000-1
#              Distinct samp/dump:   40000-0, 40000-1
#              Active SE clocks:     0, 1
#              Modulo se_clk gen:    none
#              Compare results with: behavioural simulation test vectors for data set DS4
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____0_______0_______0_____0_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7       B6         B5        B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------//
shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits
```

```
// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 20000;
dump_period_mem1[tc] = 20000;
dump_period_mem2[tc] = 20000;
dump_period_mem3[tc] = 20000;
dump_period_mem4[tc] = 20000;
dump_period_mem5[tc] = 20000;
dump_period_mem6[tc] = 20000;
dump_period_mem7[tc] = 20000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;
```

```
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 170;
y_station_id_mem[tc] = 85;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

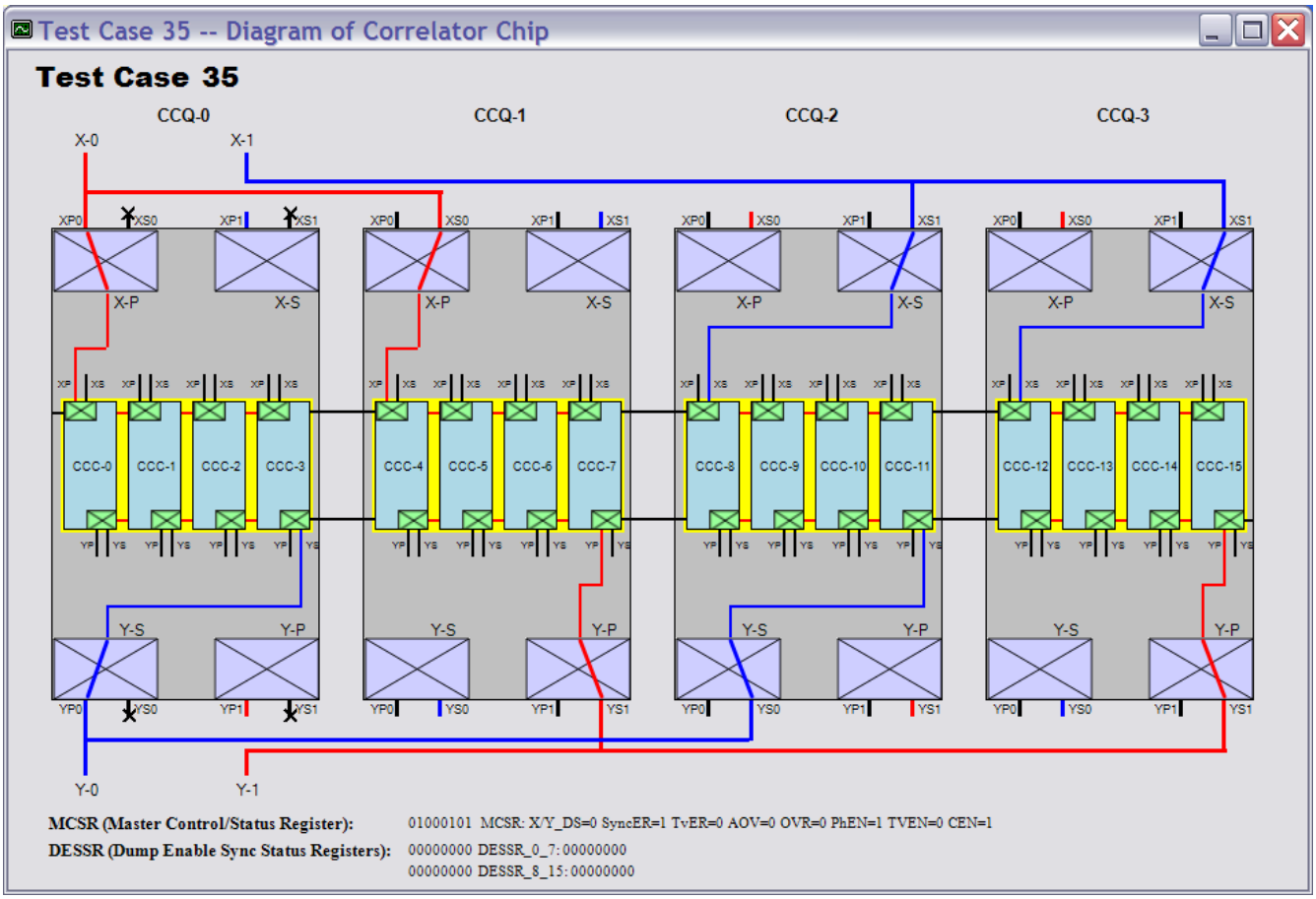
test_duration_mem[tc] = 20000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***************************************************************************************************************
```

4.36 Test case 35




```
// ********************************************************************************************************************
// File name : tc35_apr23-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc35_aug28-04_v1.v     Aug 28, 2004    1                creation
// tc35_apr23-05_v2.v     Apr 23, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 35;
test_description[tc] =
"# ---------------------------------------------------------------------------
# TEST 35:
# ver: v2,05apr23
# Synopsis:  CONCAT AS QUADS, INP 0,1 ALL POLN PRODUCTS, 1 DUMP ON Y
# Description:
#              Purpose of Test:      missing SE_CLK detection
#              Concatenation:        0(0:3)0, 0(4:7)1, 1(8:11)0, 1(12:15)1
#              Num of Dumps-input:   1-0, 1-1
#              Dump Control:         Y
#              Num samples/dump-inp: 10000-0, 10000-1
#              Distinct samp/dump:   10000-0, 10000-1
#              Active SE clocks:     0,2,4,6
#              Modulo se_clk gen:    none
#              Compare result with:  0x0 correlation, SE_CLK error report in XY-STATUS
#              Additional details:
#
# ---------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5        B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______1_______0_______1_______0_______1_______0_______1;
y_se_clk_mem[tc] = 8'b____0_______1_______0_______1_______0_______1_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
```

```
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 170;
y_station_id_mem[tc] = 85;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;
```

4.37 Test case 36

This area is intentionally left blank.

```
// ********************************************************************************************************
// File name : tc36_apr23-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc36_aug19-04_v1.v     Aug 19, 2004    1                creation
// tc36_dec07-04_v2.v     Dec  7, 2004    2                test Quad 0, secondary input
// tc36_apr23-05_v3.v     Apr 23, 2005    3                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 36;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 36:
# ver: v3,05apr36
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, 1 DUMP ON Y
# Description:
#              Purpose of Test:      check sync. error reporting
#              Concatenation:        6(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)0, 7(14:15)7
#              Num of Dumps-input:   1 on all
#              Dump Control:         Y
#              Num samples/dump-inp: 10000 on all
#              Distinct samp/dump:   10000 on all
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with:  check error reporting on DUMP_EN, TIMESTAMP;
#                                    chain 1 identical to 7
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____0_______0_______0_____0_____0_____1_____0_____1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____1_____0;
cccscr_4_5_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____1_____0;
cccscr_8_9_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____1_____0;
cccscr_12_13_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____1_____0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0_______0_________0________0_________0_________0_________0_________1;
ccqr_2_3_mem[tc] = 8'b_______0_______1_________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

fill_sync = 2'b11;                          // create data sync. errors on odd DUMP_EN lines
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

ts_fillsync = 2'b11;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;
```

```
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

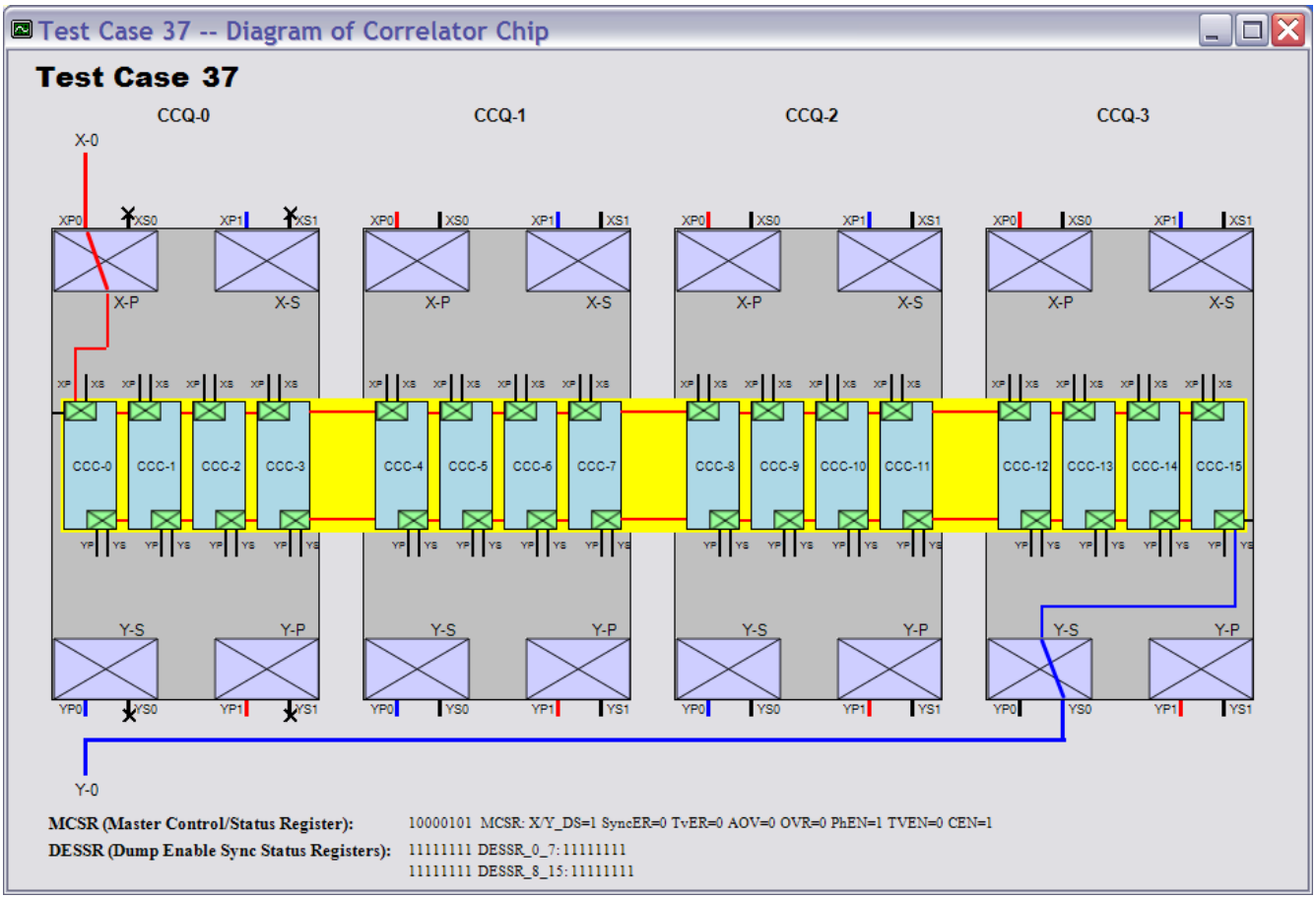
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;
```

4.38 Test case 37




```
// ********************************************************************************************************
// File name : tc37_apr23-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc37_aug24-04_v1.v     Aug 28, 2004    1                creation
// tc37_apr23-05_v2.v     Apr 23, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 37;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 37:
# ver: v2,05apr37
# Synopsis: CONCATENATE ALL, X DUMP, INPUT 0 DATA ONLY, X-Y OFFSET ON DUMP_EN
# Description:
#              Purpose of Test:   check reporting of X-Y offset on dump enable
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 10000-0
#              Distinct samp/dump:   10000-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  check X-Y dump offset bits in DESSR and XY-STATUS
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0_____0_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4         B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
```

```
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 15;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = -20;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

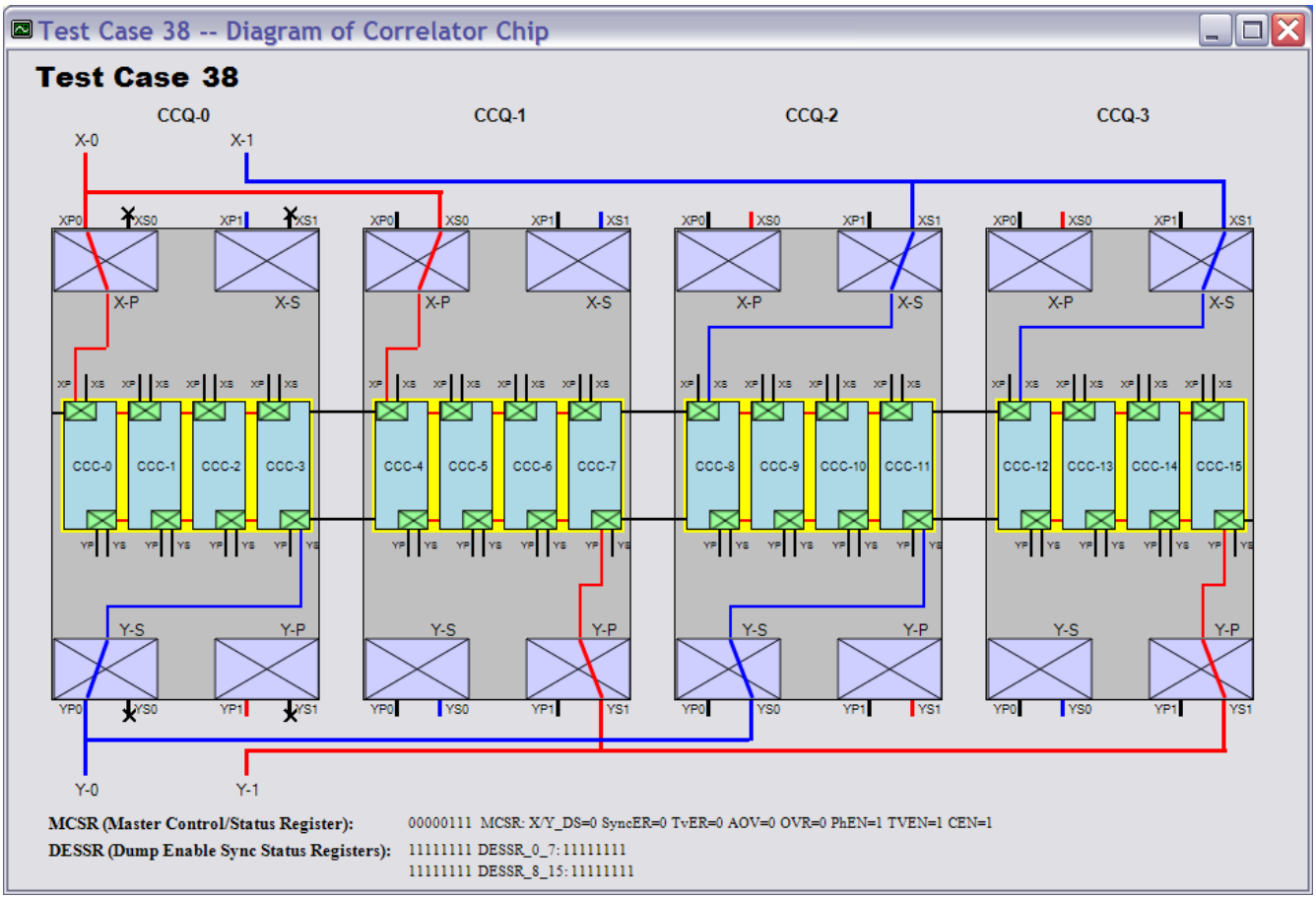
x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;
```

4.39 Test case 38




```
// ********************************************************************************************************
// File name : tc38_apr23-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc38_aug23-04_v1.v     Aug 23, 2004    1                creation
// tc38_apr23-05_v2.v     Apr 23, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 38;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 38:
# ver: v2,05apr23
# Synopsis:    Test vectors enabled
# Description:
#              Purpose of Test:      off line error checking
#              Concatenation:        0(0:3)0, 0(4:7)1, 1(8:11)0, 1(12:15)1
#              Num of Dumps-input:   1-0, 1-1
#              Dump Control:         Y
#              Num samples/dump-inp: 10000-0, 10000-1
#              Distinct samp/dump:   10000-0, 10000-1
#              Active SE clocks:     0, 1
#              Modulo se_clk gen:    none
#              Compare result with:  error detection
#              Additional details:
#
# ------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____0_______0_______0_____0_____0_____1_____1_____1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
```

```
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 170;
y_station_id_mem[tc] = 85;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

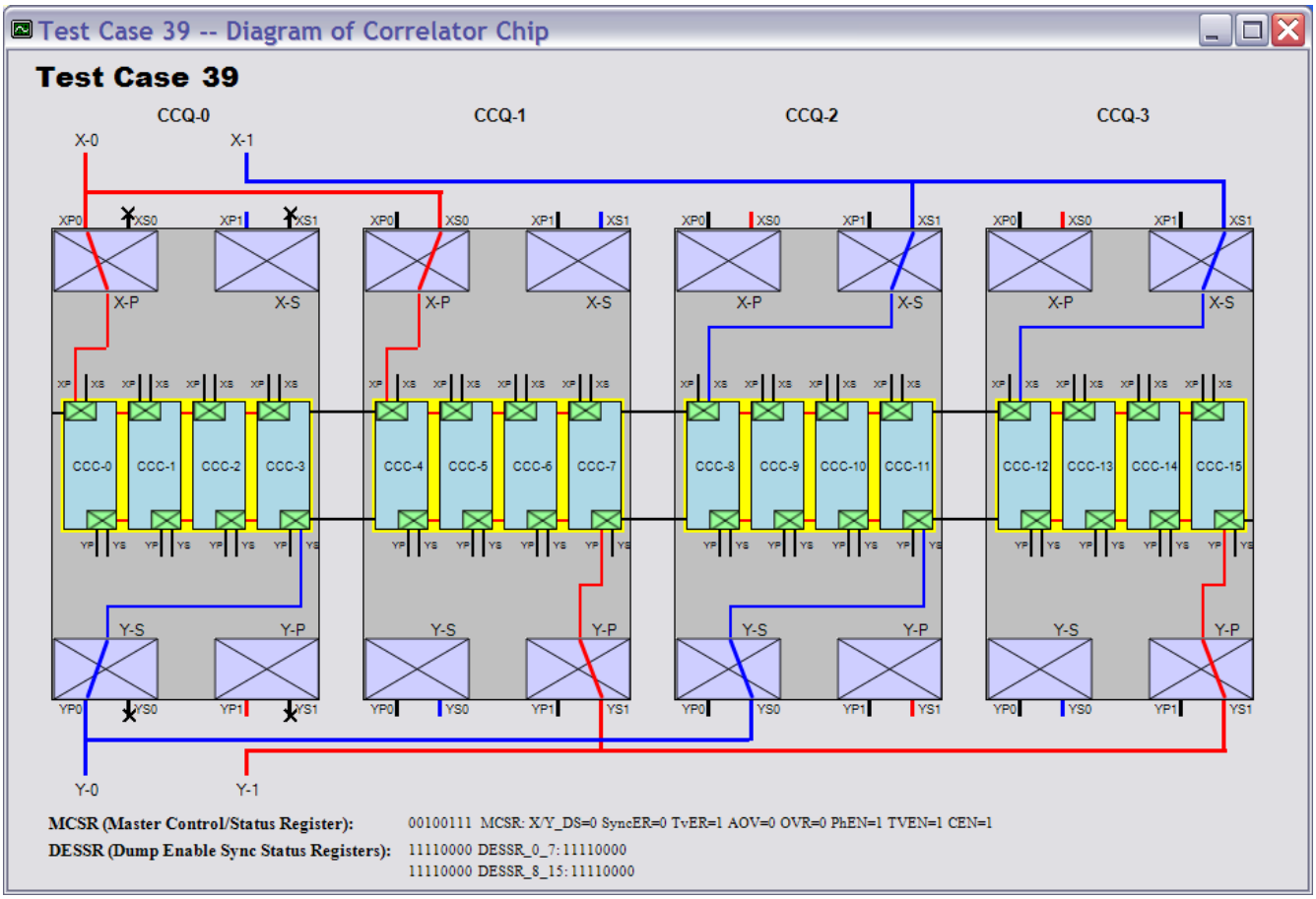
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------
```

4.40 Test case 39




```
// ********************************************************************************************************
// File name : tc39_apr23-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc39_aug23-04_v1.v     Aug 23, 2004    1                creation
// tc39_apr23-05_v2.v     Apr 23, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************
//
```

```
//
// ***************************************************************************************************************
// ***************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***************************************************************************************************************
// ***************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 39;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 39:
# ver: v2,05apr23
# Synopsis:      Test vectors enabled
# Description:
#              Purpose of Test:      off line error checking
#              Concatenation:        0(0:3)0, 0(4:7)1, 1(8:11)0, 1(12:15)1
#              Num of Dumps-input:   1-0, 1-1
#              Dump Control:         Y
#              Num samples/dump-inp: 10000-0, 10000-1
#              Distinct samp/dump:   10000-0, 10000-1
#              Active SE clocks:     0, 1
#              Modulo se_clk gen:    none
#              Compare result with:  error detection
#              Additional details:   input sync. errors applied
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____0_______0_______0_____0_____0_____1_____1_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                     CCCSCR Input Selection:
//                     00 = adjacent CCC
//                     01 = primary CCC input
//                     10 = secondary CCC input
//                     11 = undefined: no lag frames produced
//
//                        CCC0,1 switch config register: (same for CCC2...CCC15)
//                      |--------- CCC1 ----------|---------- CCC0 -----------|
//                      ------------------------------------------------------
//                      Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                      ------------------------------------------------------
//                       B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCQR Input Selection:
//                           0 = primary
//                           1 = secondary
//
//                      CCQ0,1 switch config register
//                      ---------------------------------------------------------------------------
//                      Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                      ---------------------------------------------------------------------------
//                          B7       B6         B5       B4         B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
```

```
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 100;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 170;
y_station_id_mem[tc] = 85;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

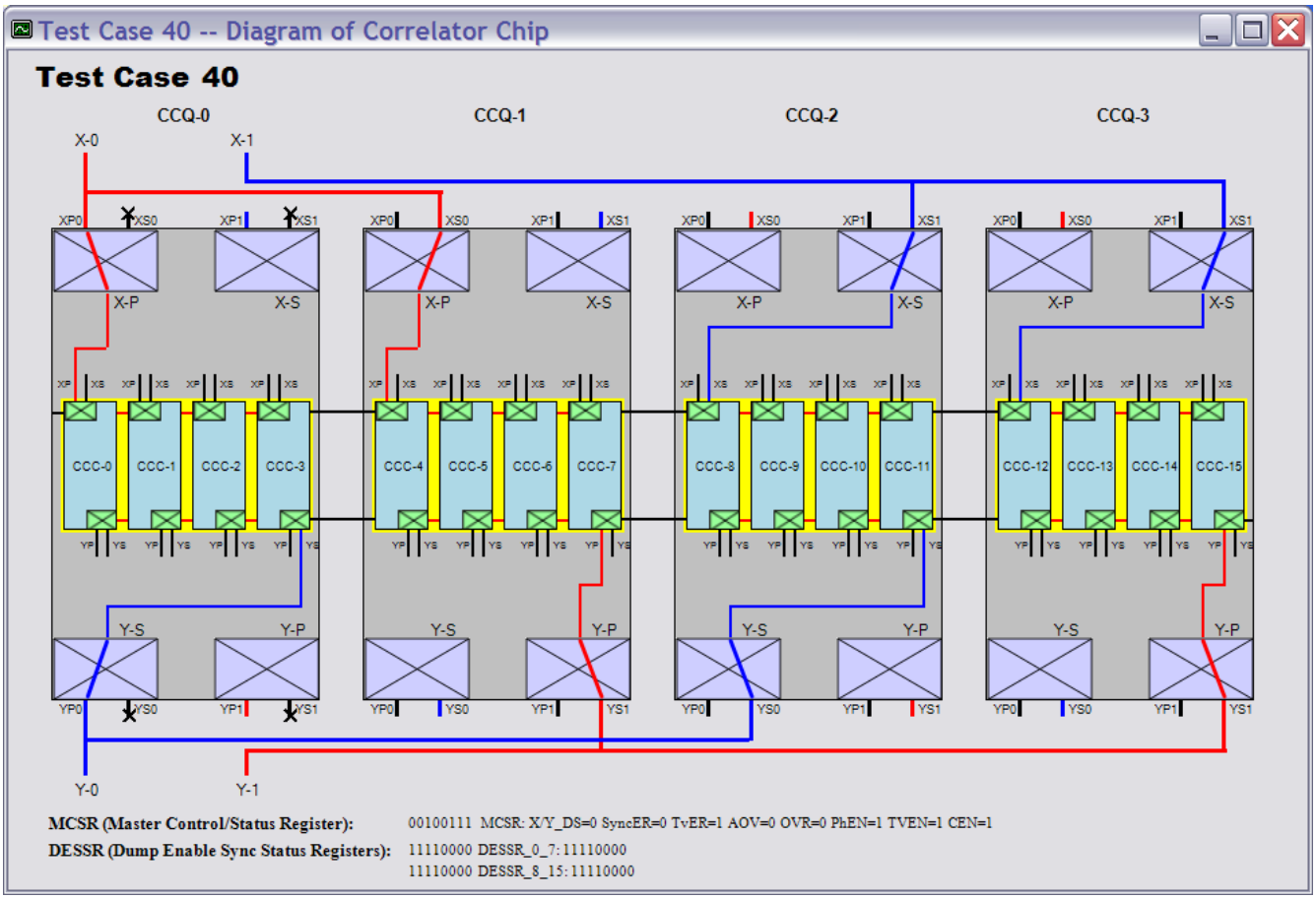
x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;
```

4.41 Test case 40




```
// ********************************************************************************************************
// File name : tc40_apr23-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc40_aug23-04_v1.v     Aug 23, 2004    1                creation
// tc40_apr23-05_v2.v     Apr 23, 2005    2                upgrade to new test bench
//
//
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 40;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 40:
# ver: v2,05apr23
# Synopsis:      Test vectors enabled
# Description:
#              Purpose of Test:      off line error checking
#              Concatenation:        0(0:3)0, 0(4:7)1, 1(8:11)0, 1(12:15)1
#              Num of Dumps-input:   1-0, 1-1
#              Dump Control:         Y
#              Num samples/dump-inp: 10000-0, 10000-1
#              Distinct samp/dump:   10000-0, 10000-1
#              Active SE clocks:     0, 1
#              Modulo se_clk gen:    none
#              Compare result with:  error detection
#              Additional details:   input sync. errors applied
#
# ------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____0_______0_______0_____0_____0_____1_____1_____1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
```

```
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
y_dumpen2_mem[tc] = {70'bx,2'b0};
y_dumpen3_mem[tc] = {70'bx,2'b0};
y_dumpen4_mem[tc] = {70'bx,2'b0};
y_dumpen5_mem[tc] = {70'bx,2'b0};
y_dumpen6_mem[tc] = {70'bx,2'b0};
y_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 2000;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 170;
y_station_id_mem[tc] = 85;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

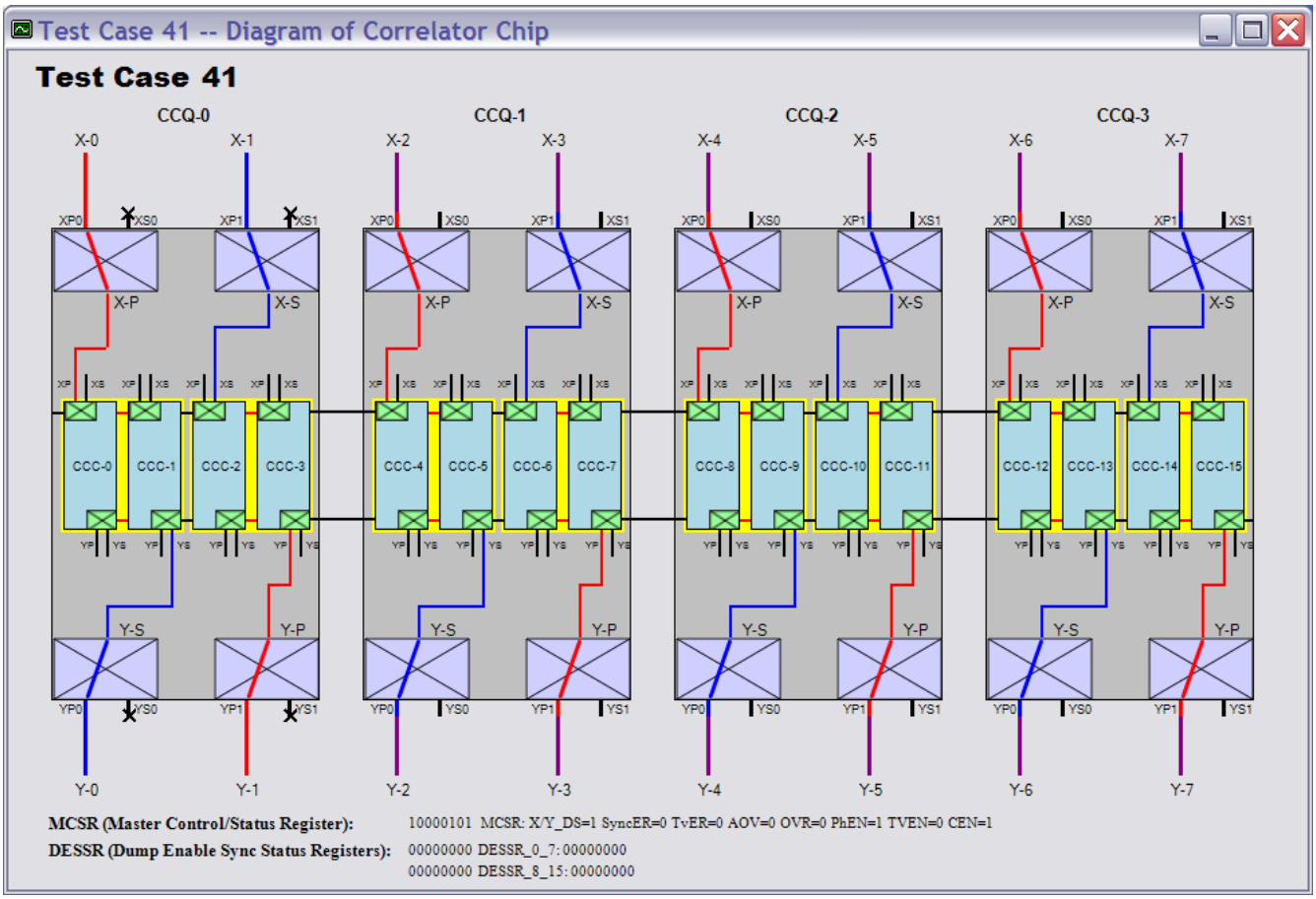
test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// *****************************************************************************************************************
```

4.42 Test case 41




```
// ********************************************************************************************************
// File name : tc41_apr26-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc41_aug19-04_v1.v     Aug 19, 2004    1                creation
// tc41_aug24-04_v1.v     Aug 24, 2004    1                modify data ID
// tc41_apr24-05_v2.v     Apr 24, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc41_apr26-05_v2.v     Apr 26, 2005    2                add unused dump commands
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 41;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 41:
# ver: v2,05apr26
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, 1 DUMP ON X
# Description:
#              Purpose of Test:      dump control data capture, data ID capture
#              Concatenation:        0(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)6, 7(14:15)7
#              Num of Dumps-input:   1 on all
#              Dump Control:         X
#              Num samples/dump-inp: 10000 on all
#              Distinct samp/dump:   10000 on all
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with:  check dump cmmd, RECIRC_BLK, Phase Bin, data ID
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______0______0______0______0______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______0______0______0______0______1______0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.

dump_command = 3'b001;                      // 3 bits:
phase_bin = 21841;                          // 16 bits
recirc_blk = 81;                            // 8 bits
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21843;                          // 16 bits
recirc_blk = 83;                            // 8 bits
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b101;                      // 3 bits:
phase_bin = 21845;                          // 16 bits
recirc_blk = 85;                            // 8 bits
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b111;                      // 3 bits:
phase_bin = 21847;                          // 16 bits
recirc_blk = 87;                            // 8 bits
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

fill_sync = 2'b01;
dump_command = 3'b000;                      // 3 bits:
phase_bin = 21840;                          // 16 bits
recirc_blk = 80;                            // 8 bits
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 21842;                          // 16 bits
recirc_blk = 82;                            // 8 bits
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b110;                      // 3 bits:
phase_bin = 21844;                          // 16 bits
recirc_blk = 84;                            // 8 bits
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21846;                          // 16 bits
recirc_blk = 86;                            // 8 bits
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
```

```
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.
```

```
x_station_id_mem[tc] = 0;
y_station_id_mem[tc] = 255;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 24;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

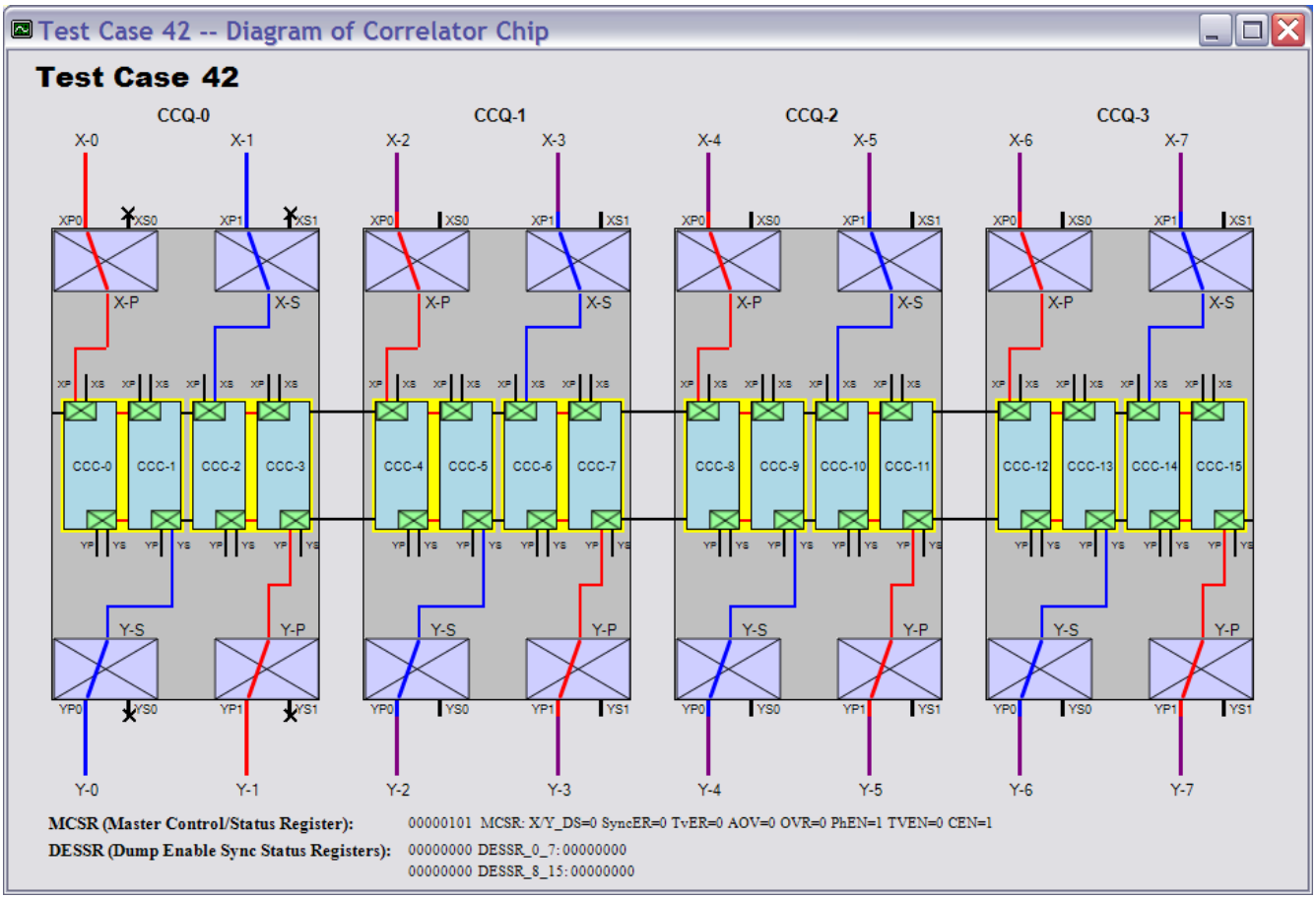
test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.43 Test case 42




```
// ********************************************************************************************************
// File name : tc42_apr26-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc42_aug19-04_v1.v     Aug 19, 2004    1                creation
// tc42_aug25-04_v1.v     Aug 25, 2004    1                increase dts_length to accomodate extended holdoff
// tc42_apr24-05_v2.v     Apr 24, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
// tc42_apr26-05_v2.v     Apr 26, 2005    2                add unused dump commands
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 42;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 42:
# ver: v2,05apr26
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, 1 DUMP ON Y
# Description:
#              Purpose of Test:      dump control data capture, hold off
#              Concatenation:        0(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)6, 7(14:15)7
#              Num of Dumps-input:   1 on all
#              Dump Control:         Y
#              Num samples/dump-inp: 10000 on all
#              Distinct samp/dump:   10000 on all
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with:  check dump cmmd, RECIRC_BLK, Phase Bin, hold off
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______0______0______0______0______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______0______0______0______0______1______0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

dts_len_mem[tc]= 312;  // increase size of dts_len to 312 bits total.

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

fill3 = 3'b111;                              // 3 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.

// Holdoff held high to disable correlation until lag shift registers
// fill up with data. Maximum holdoff size is 280 bits in this test case.

dump_command = 3'b101;                      // 3 bits:
phase_bin = 43691;                          // 16 bits
recirc_blk = 81;                            // 8 bits
holdoff = 'h1fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 257 bits
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 43693;                          // 16 bits
recirc_blk = 83;                            // 8 bits
holdoff = 'h7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 259 bits
y_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 43695;                          // 16 bits
recirc_blk = 85;                            // 8 bits
holdoff = 'h1fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 261 bits
y_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b000;                      // 3 bits:
phase_bin = 43697;                          // 16 bits
recirc_blk = 87;                            // 8 bits
holdoff = 'h7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 263 bits
y_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

fill_sync = 2'b01;
dump_command = 3'b011;                      // 3 bits:
phase_bin = 43690;                          // 16 bits
recirc_blk = 80;                            // 8 bits
holdoff = 'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 256 bits
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b110;                      // 3 bits:
phase_bin = 43692;                          // 16 bits
recirc_blk = 82;                            // 8 bits
holdoff = 'h3fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 258 bits
y_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b111;                      // 3 bits:
phase_bin = 43694;                          // 16 bits
recirc_blk = 84;                            // 8 bits
holdoff = 'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 260 bits
y_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b001;                      // 3 bits:
phase_bin = 43696;                          // 16 bits
recirc_blk = 86;                            // 8 bits
holdoff = 'h3fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 262 bits
y_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.
```

```
// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
```

```
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

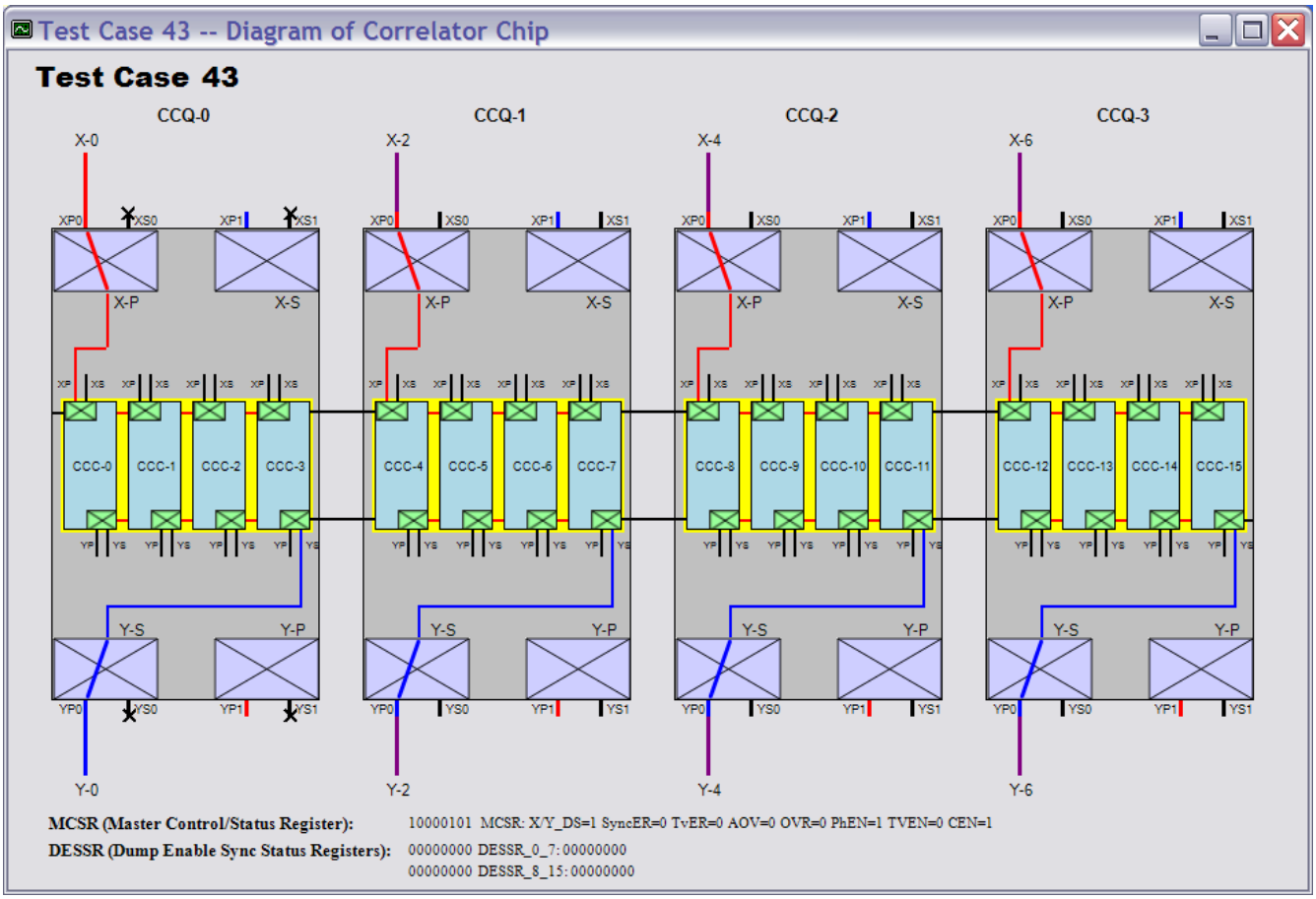
test_duration_mem[tc] = 5000;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// *************************************************************************************************************
```

4.44 Test case 43




```
// ********************************************************************************************************************
// File name : tc44_apr24-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc44_aug24-04_v1.v     Aug 28, 2004    1                creation
// tc44_aug31-04_v1.v     Aug 31, 2004    1                bug fix: interval < test duration
// tc44_apr24-05_v2.v     Apr 24, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench; change configuration
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 44;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 44:
# ver: v2,05apr24
# Synopsis:    CONCAT ALL, X DUMP, INPUT 0 DATA ONLY, PHASE ROTATION DISABLED
# Description:
#              Purpose of Test:      phase rotation enable
#              Concatenation:        1(0)1, 0(1)0, 1(2)1, 0(3)0, 2(4)2, 2(5)2, 3(6)3,
#                                 3(7)3, 5(8)5, 5(9)5, 4(10)4, 4(11)4, 6(12)6, 7(13)7,
#                                 6(14)6, 7(15)7
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 40000-0
#              Distinct samp/dump:   40000-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  check correlation in output lag frames
#              Additional details:   disable phase rotation (PhEN=0)
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0_____0_____0_____0_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____1_____0;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____1_____0;
cccscr_4_5_mem[tc] = 8'b___1_____0_____0_____1_____1_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_8_9_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____1_____1_____0_____0_____1;
cccscr_12_13_mem[tc] = 8'b_0_____1_____1_____0_____1_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_0_____1_____1_____0_____1_____0_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 20000;
dump_period_mem1[tc] = 20000;
dump_period_mem2[tc] = 20000;
dump_period_mem3[tc] = 20000;
dump_period_mem4[tc] = 20000;
dump_period_mem5[tc] = 20000;
dump_period_mem6[tc] = 20000;
```

```
dump_period_mem7[tc] = 20000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 20000;
```

4.45 Test case 44

This region is intentionally left blank.

```
// ********************************************************************************************************
// File name : tc45_apr24-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ************************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc45_aug28-04_v1.v     Aug 28, 2004    1                creation
// tc45_apr24-05_v2.v     Apr 24, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 45;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 45:
# ver: v2,05apr24
# Synopsis:    CORRELATION DISABLED
# Description:
#              Purpose of Test:      correlation enable
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 10000-0
#              Distinct samp/dump:   10000-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  output lag frames are not generated
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1_________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
```

```
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

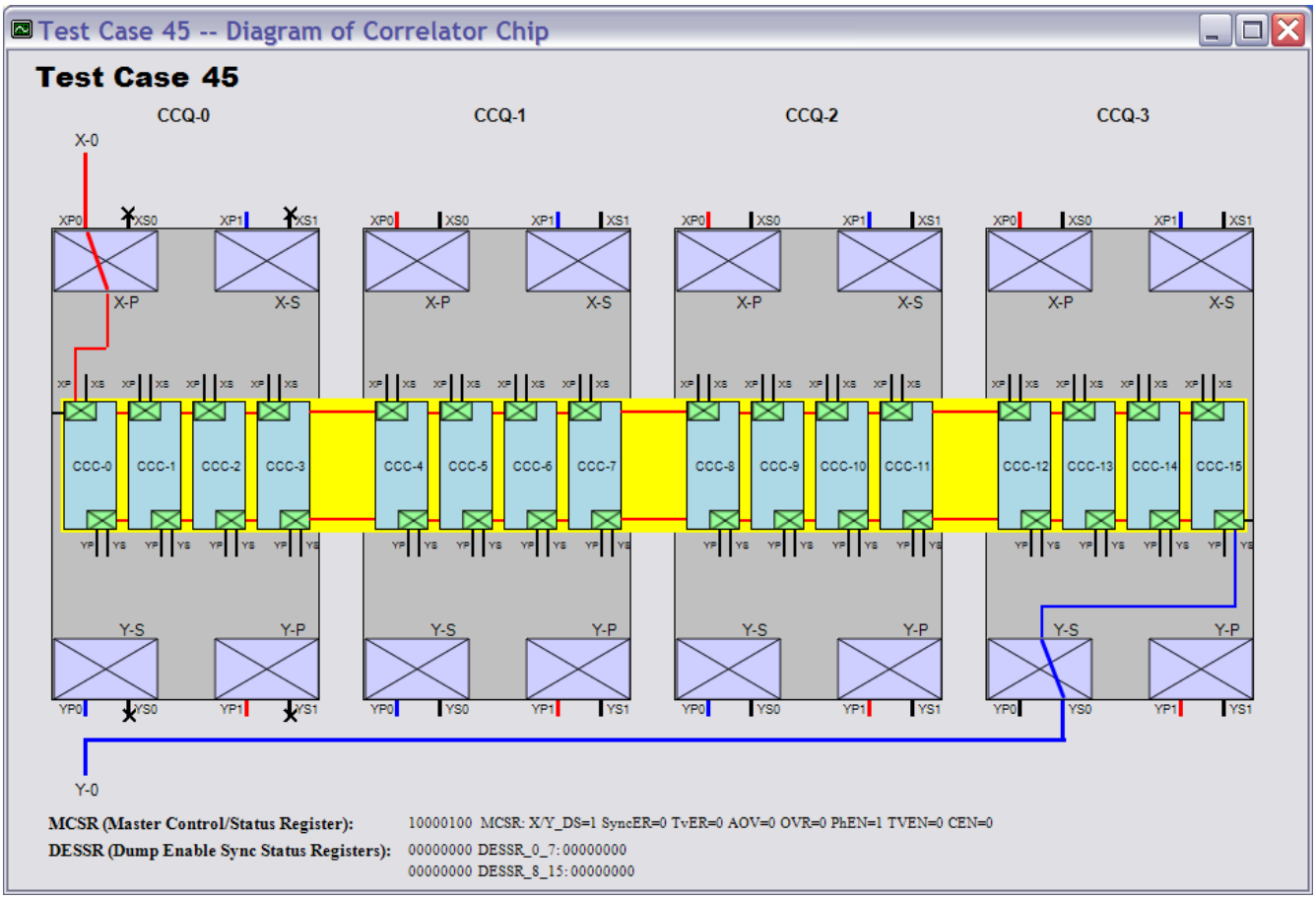
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;
```

4.46 Test case 45




```
// ********************************************************************************************************************
// File name : tc45_apr24-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc45_aug28-04_v1.v     Aug 28, 2004    1                creation
// tc45_apr24-05_v2.v     Apr 24, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 45;
test_description[tc] =
"# ---------------------------------------------------------------------------
# TEST 45:
# ver: v2,05apr24
# Synopsis:    CORRELATION DISABLED
# Description:
#              Purpose of Test:      correlation enable
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 10000-0
#              Distinct samp/dump:   10000-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  output lag frames are not generated
#              Additional details:
#
# ---------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0_____0_____0_____1_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
```

```
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

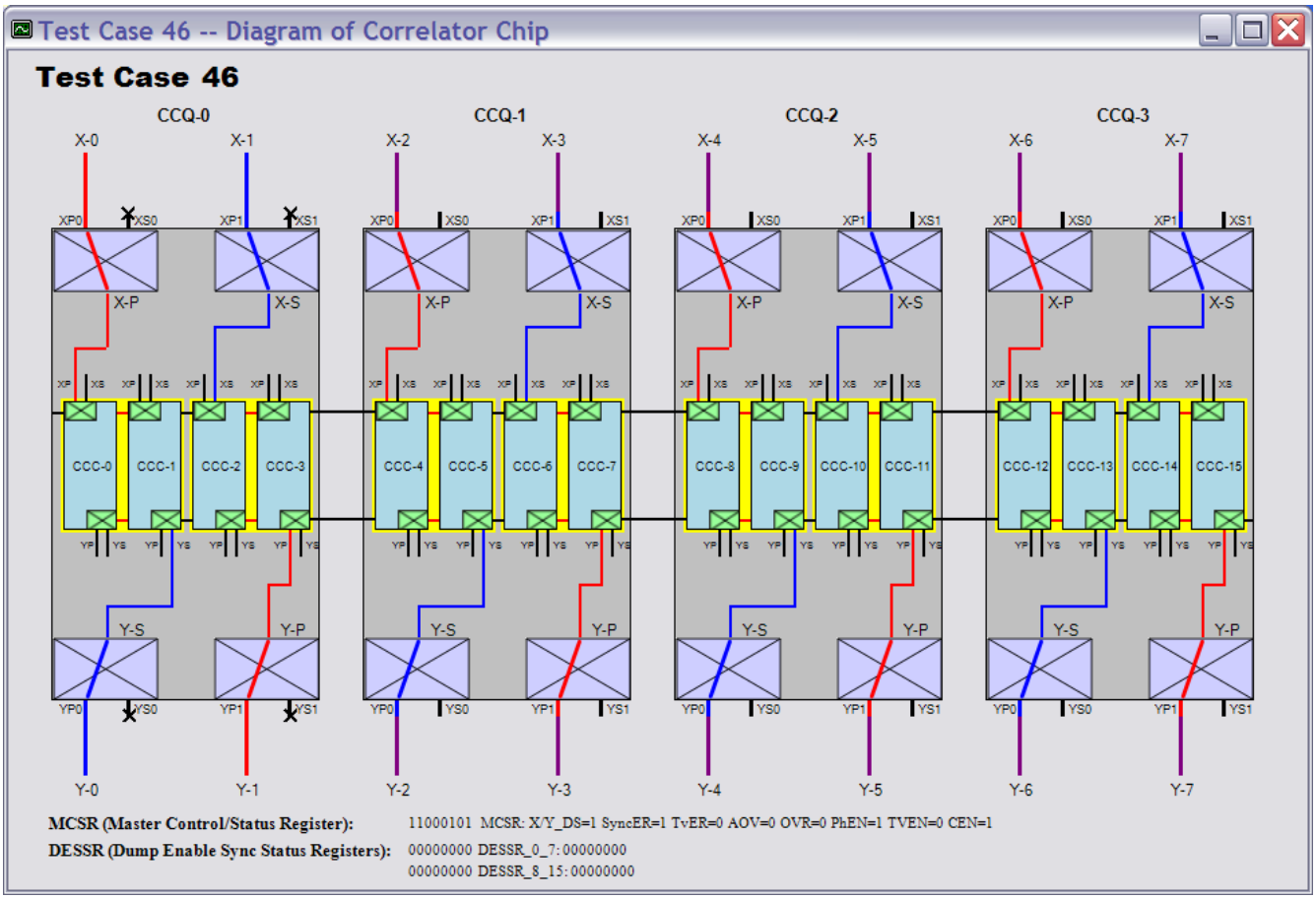
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 5000;
```

4.47 Test case 46




```
// ********************************************************************************************************
// File name : tc46_apr24-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc46_aug22-04_v1.v     Aug 22, 2004    1                creation
// tc46_apr24-05_v2.v     Apr 24, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 46;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 46:
# ver: v2,05apr24
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, 1 DUMP ON X
# Description:
#              Purpose of Test:      dump enable error and data capture
#              Concatenation:        0(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)6, 7(14:15)7
#              Num of Dumps-input:   1 on all
#              Dump Control:         X
#              Num samples/dump-inp: 10000 on all
#              Distinct samp/dump:   10000 on all
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with:  RECIRC_BLK, Phase Bin, error reporting on DUMP_EN
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______0______0______0______0______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______0______0______0______0______1______0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b11;                          // 2 bits, bad fill sync

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.

dump_command = 3'b001;                      // 3 bits:
phase_bin = 21841;                          // 16 bits
recirc_blk = 171;                           // 8 bits
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21843;                          // 16 bits
recirc_blk = 173;                           // 8 bits
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 21845;                          // 16 bits
recirc_blk = 175;                           // 8 bits
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21847;                          // 16 bits
recirc_blk = 177;                           // 8 bits
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

fill_sync = 2'b01; // good fill sync
dump_command = 3'b000;                      // 3 bits:
phase_bin = 21840;                          // 16 bits
recirc_blk = 170;                           // 8 bits
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 21842;                          // 16 bits
recirc_blk = 172;                           // 8 bits
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 21844;                          // 16 bits
recirc_blk = 174;                           // 8 bits
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21846;                          // 16 bits
recirc_blk = 176;                           // 8 bits
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
```

```
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.
```

```
x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

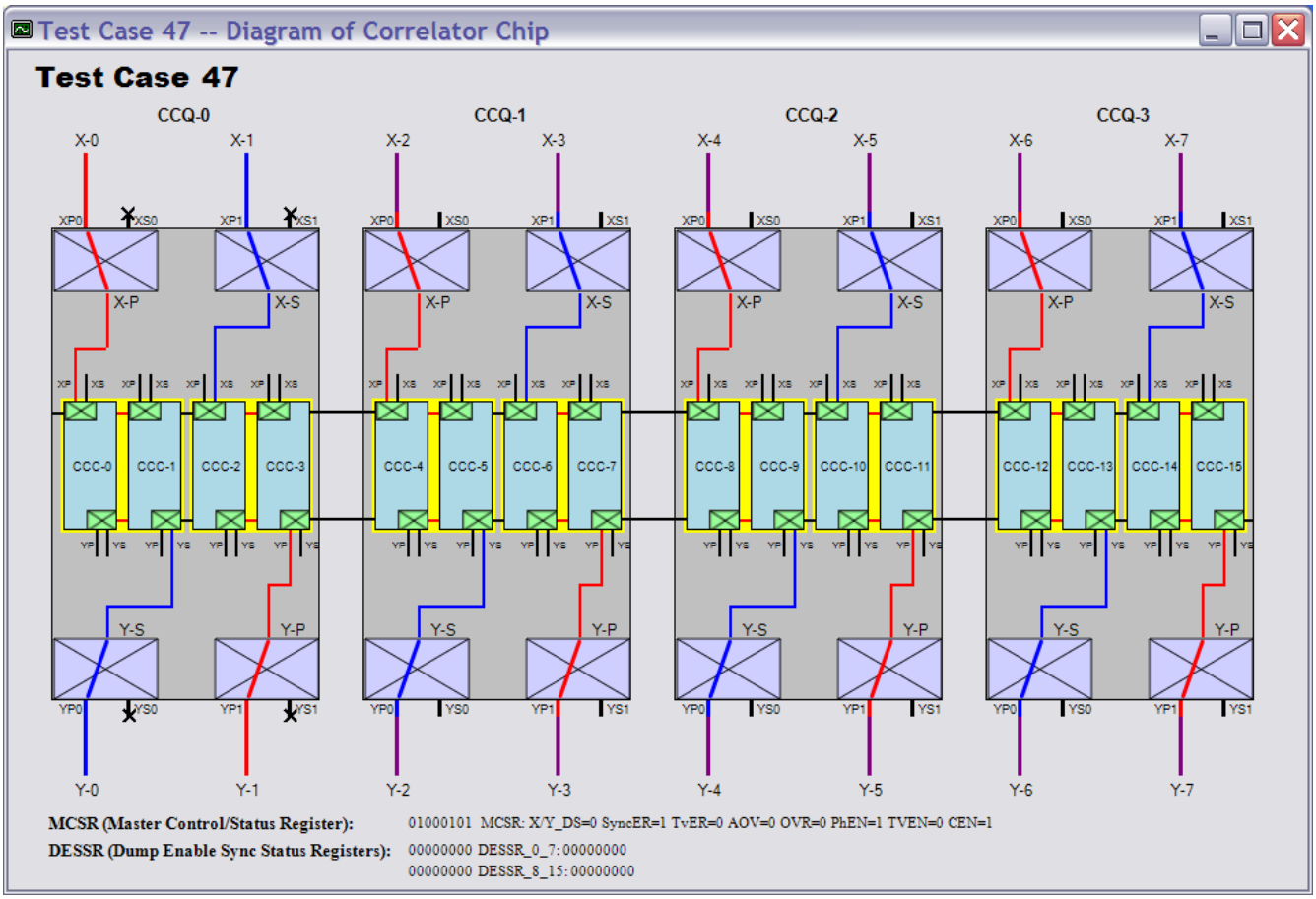
test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.48 Test case 47




```
// ********************************************************************************************************************
// File name : tc47_apr24-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME               DATE            VERSION          COMMENTS
// tc47_aug22-04_v1.v      Aug 22, 2004    1                creation
// tc47_apr24-05_v2.v      Apr 24, 2005    2                add verification mode and split acc. options;
//                                                          upgrade to new test bench
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 47;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 47:
# ver: v2,05apr24
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, 1 DUMP ON Y
# Description:
#              Purpose of Test:      dump enable error and data capture
#              Concatenation:        0(0:1)0, 1(2:3)1, 2(4:5)2, 3(6:7)3, 4(8:9)4,
#                                 5(10:11)5, 6(12:13)6, 7(14:15)7
#              Num of Dumps-input:   1 on all
#              Dump Control:         Y
#              Num samples/dump-inp: 10000 on all
#              Distinct samp/dump:   10000 on all
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with:  RECIRC_BLK, Phase Bin, error reporting on DUMP_EN
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______0______0______0______0______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______0______0______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______0______0______0______0______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______0______0______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______0______0______0______0______1______0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
```

```
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits, good fill sync

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.

dump_command = 3'b001;                      // 3 bits:
phase_bin = 21841;                          // 16 bits
recirc_blk = 171;                           // 8 bits
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21843;                          // 16 bits
recirc_blk = 173;                           // 8 bits
y_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 21845;                          // 16 bits
recirc_blk = 175;                           // 8 bits
y_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21847;                          // 16 bits
recirc_blk = 177;                           // 8 bits
y_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

fill_sync = 2'b11; // bad fill sync
dump_command = 3'b000;                      // 3 bits:
phase_bin = 21840;                          // 16 bits
recirc_blk = 170;                           // 8 bits
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 21842;                          // 16 bits
recirc_blk = 172;                           // 8 bits
y_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 21844;                          // 16 bits
recirc_blk = 174;                           // 8 bits
y_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21846;                          // 16 bits
recirc_blk = 176;                           // 8 bits
y_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
```

```
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 5000;
dump_period_mem1[tc] = 5000;
dump_period_mem2[tc] = 5000;
dump_period_mem3[tc] = 5000;
dump_period_mem4[tc] = 5000;
dump_period_mem5[tc] = 5000;
dump_period_mem6[tc] = 5000;
dump_period_mem7[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.
```

```
x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

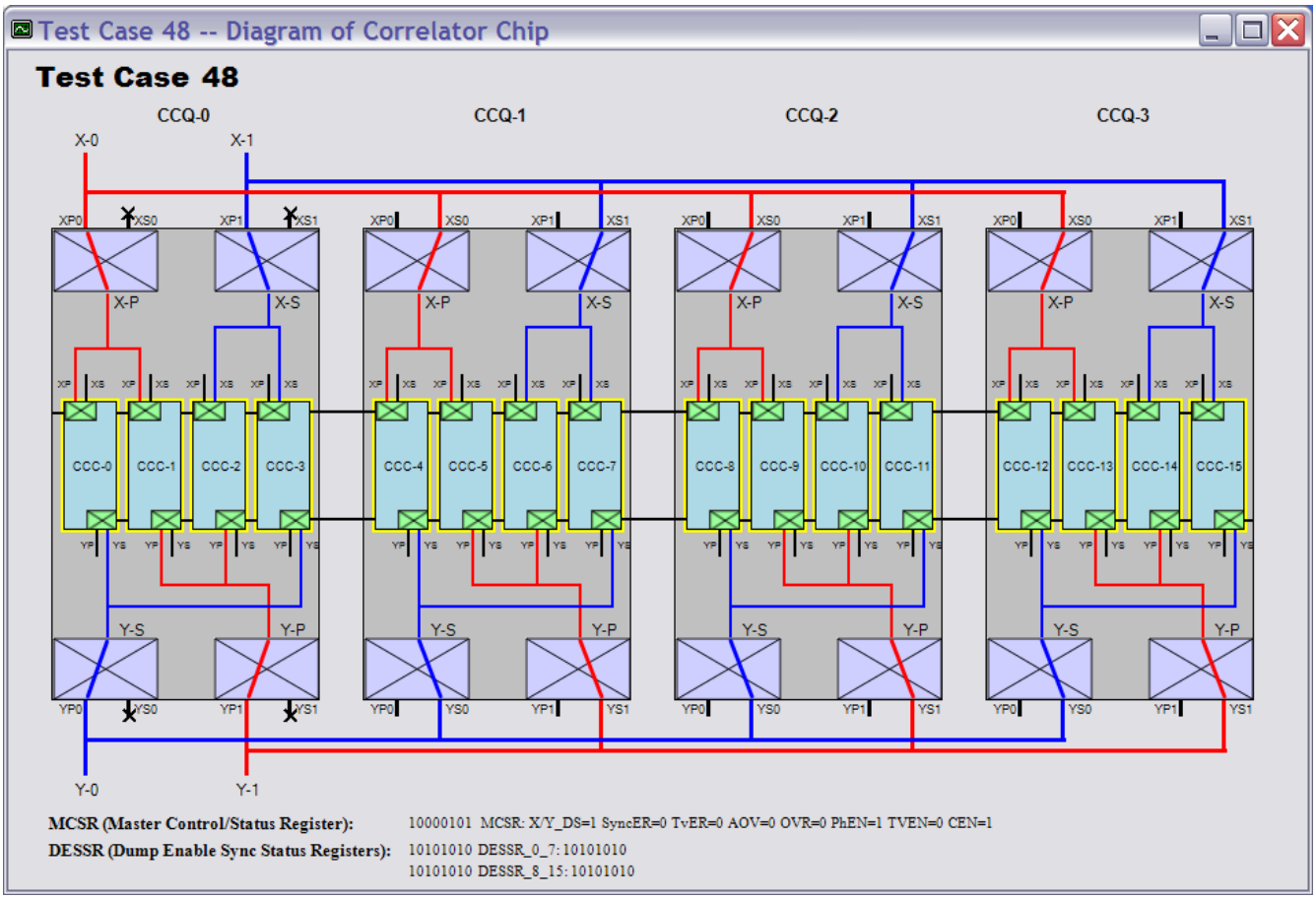
test_duration_mem[tc] = 5000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.49 Test case 48




```
// ********************************************************************************************************************
// File name : tc48_aug07-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V3
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc48_aug24-04_v1.v     Aug 28, 2004    1                creation
// tc48_apr16-05_v2.v     Apr 16, 2005    2                decrease run time;
//                                                         add verifcation mode option
//
// tc48_jul27-05_v2.v     Jul 27, 2005    2               decrease X-Y dump time differential
// tc48_aug07-05_v3.v     Aug  7, 2005    3               make X-Y dump times harmonically related
//
// ********************************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                    BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 48;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 48:
# ver: v3,05aug07
# Synopsis:    INDEP CELLS, 4 POLS, X DUMP, UNMATCHED X-Y DUMP RATE
# Description:
#              Purpose of Test:      reporting X-Y offset on dump enable
#              Concatenation:        0(0)0 0(1)1 1(2)1 1(3)0 0(4)0 0(5)1 1(6)1 1(7)0
#                                 0(8)0 0(9)1 1(10)1 1(11)0 0(12)0 0(13)1 1(14)1 1(15)0
#              Num of Dumps-inp.:    2-0, 1-1
#              Dump Control:         X
#              Num samples/dump-inp: 500-0, 1000-1
#              Distinct samp/dump:   500-0, 1000-1
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  check offset reporting in DESSR
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____1_____0_____1_____1_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_4_5_mem[tc] = 8'b___0_____1_____0_____1_____1_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_8_9_mem[tc] = 8'b___0_____1_____0_____1_____1_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_1_____0_____1_____0_____0_____1_____1_____0;
cccscr_12_13_mem[tc] = 8'b_0_____1_____0_____1_____1_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_1_____0_____1_____0_____0_____1_____1_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4         B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits


// Dump control bits:
```

```
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
// these are dummies since only input 0 is valid
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 250;
dump_period_mem1[tc] = 500;
dump_period_mem2[tc] = 500;
dump_period_mem3[tc] = 500;
dump_period_mem4[tc] = 500;
dump_period_mem5[tc] = 500;
dump_period_mem6[tc] = 500;
dump_period_mem7[tc] = 500;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

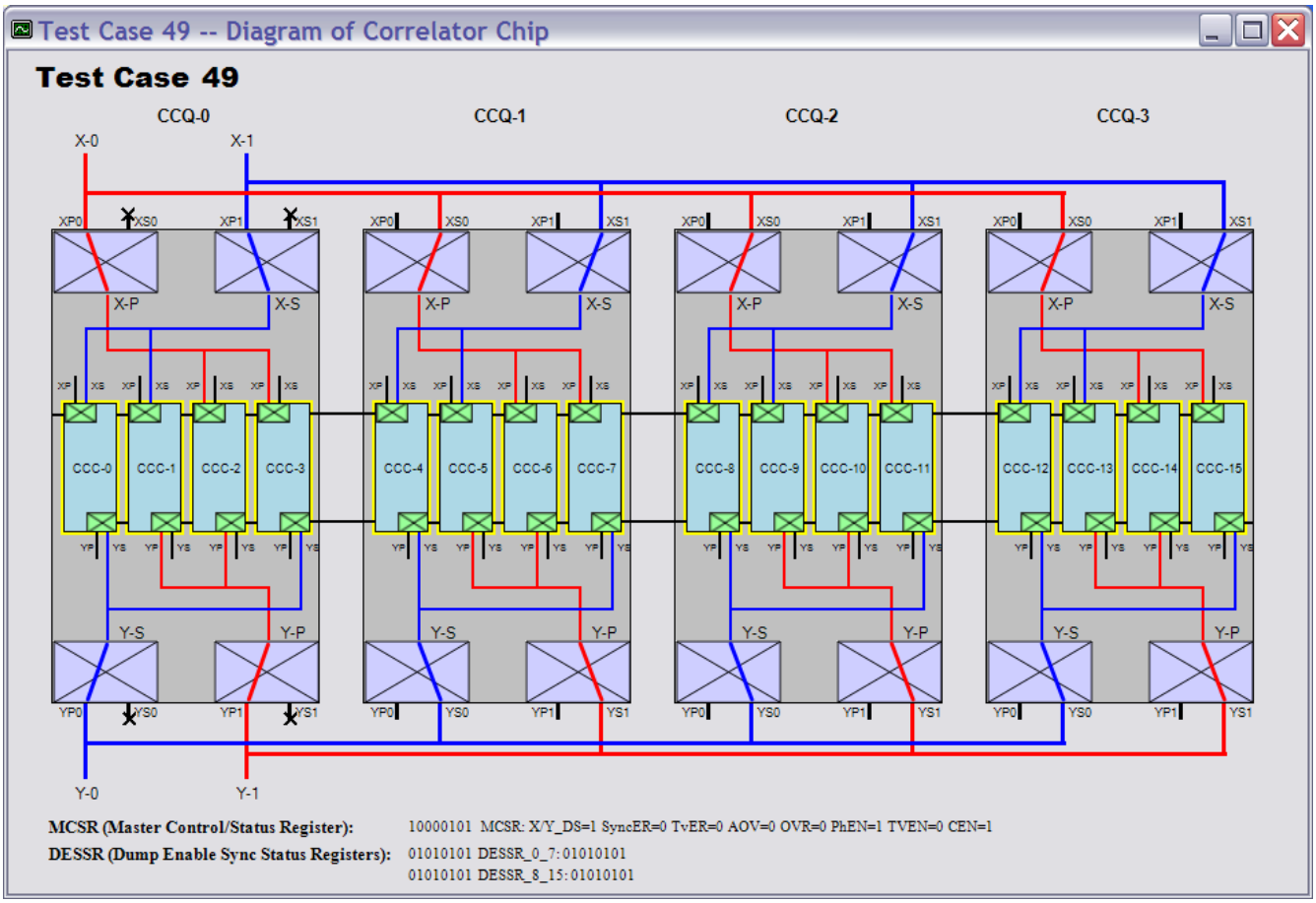
test_duration_mem[tc] = 500;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***************************************************************************************************
```

4.50 Test case 49




```
// ********************************************************************************************************************
// File name : tc49_aug07-05_v3.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc49_aug24-04_v1.v     Aug 28, 2004    1                creation
// tc49_apr16-05_v2.v     Apr 16, 2005                     decrease run time; add verifcation mode option
// tc49_jul27-05_v2.v     Jul 27, 2005    2                decrease X-Y dump time differential
// tc49_aug07-05_v3.v     Aug  7, 2005    3                make X-Y dump time harmonically related
//
// ********************************************************************************************************************
//
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 49;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 49:
# ver: v3,05aug07
# Synopsis:    INDEP CELLS, 4 POLS, X DUMP, UNMATCHED X-Y DUMP RATE
# Description:
#              Purpose of Test:      reporting X-Y offset on dump enable
#              Concatenation:        1(0)0 1(1)1 0(2)1 0(3)0 1(4)0 1(5)1 0(6)1 0(7)0
#                                 1(8)0 1(9)1 0(10)1 0(11)0 1(12)0 1(13)1 0(14)1 0(15)0
#              Num of Dumps-inp.:    2-0, 1-1
#              Dump Control:         X
#              Num samples/dump-inp: 500-0, 1000-1
#              Distinct samp/dump:   500-0, 1000-1
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  check offset reporting in DESSR
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0_____0_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____1_____1_____0_____1_____0_____1_____0;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_4_5_mem[tc] = 8'b___0_____1_____1_____0_____1_____0_____1_____0;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_8_9_mem[tc] = 8'b___0_____1_____1_____0_____1_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;
cccscr_12_13_mem[tc] = 8'b_0_____1_____1_____0_____1_____0_____1_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//                 modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
```

```
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
// these are dummies since only input 0 is valid
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 250;
dump_period_mem1[tc] = 500;
dump_period_mem2[tc] = 500;
dump_period_mem3[tc] = 500;
dump_period_mem4[tc] = 500;
dump_period_mem5[tc] = 500;
dump_period_mem6[tc] = 500;
dump_period_mem7[tc] = 500;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

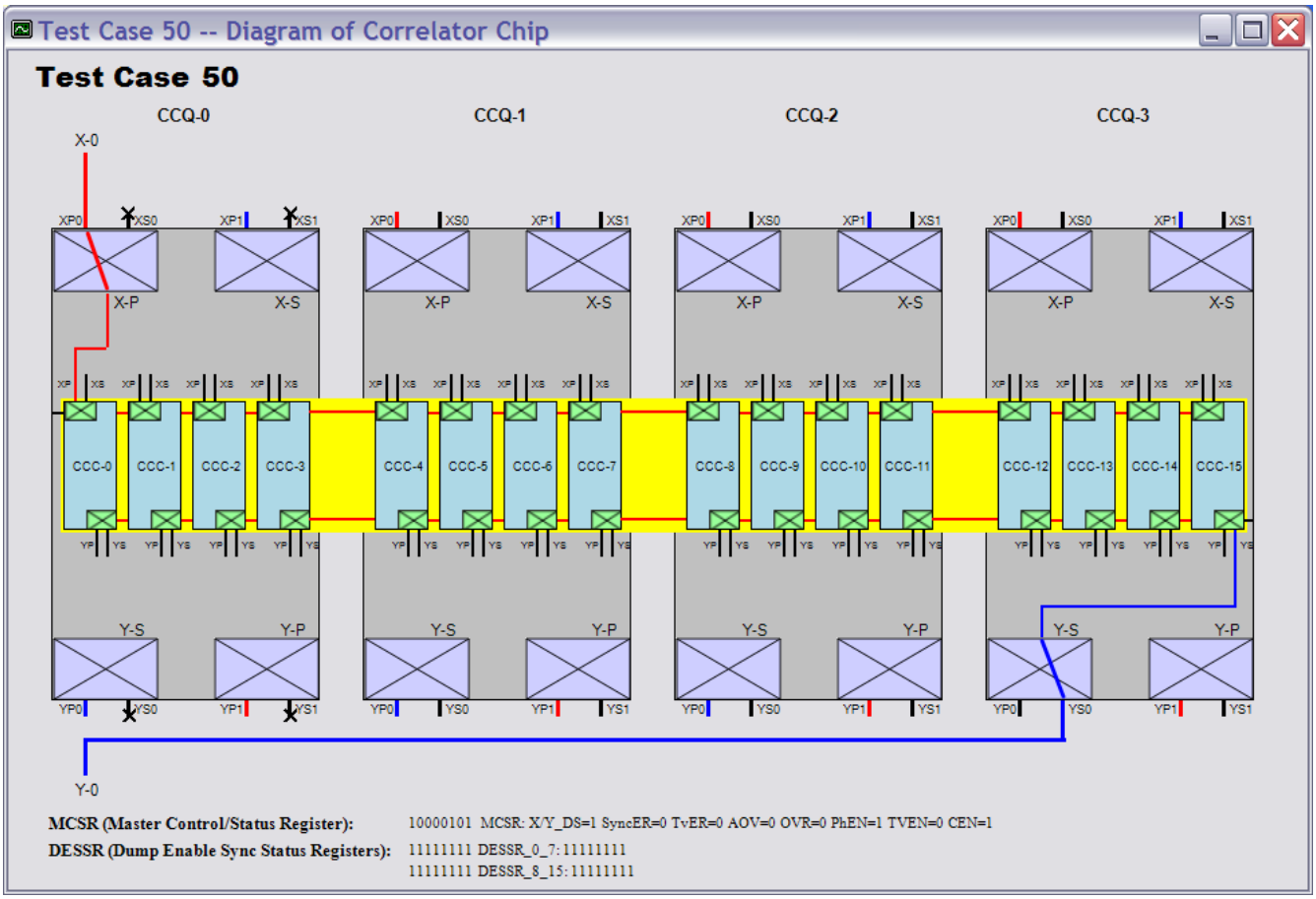
test_duration_mem[tc] = 500;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***************************************************************************************************
```

4.51 Test case 50




```
// ********************************************************************************************************************
// File name : tc50_apr24-05_v1.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V1
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc50_apr24-05_v1.v     Apr 24, 2005    1                creation (recycle 50)
//
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 50;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 50:
# ver: v1,05apr24
# Synopsis:   NO CONCAT., ALL POLN, 8 INPUTS, SELECTED STREAMS RESET
# Description:
#              Purpose of Test:      detection of missing phase or data streams
#              Concatenation:        1(0)1, 1(1)0, 0(2)1, 0(3)0, 3(4)3, 3(5)2, 2(6)3,
#                                 2(7)2, 5(8)5, 5(9)4, 4(10)5, 4(11)4, 7(12)7, 7(13)6,
#                                 6(14)7, 6(15)6
#              Num of Dumps-input:   1-0, 1-1, 1-2, 1-3, 1-4, 1-5, 1-6, 1-7
#              Dump Control:         X
#              Num samples/dump-inp: 5000-0, 5000-1, 5000-2, 5000-3, 5000-4, 5000-5,
#                                 5000-6, 5000-7
#              Distinct samp/dump:  5000-0, 5000-1, 5000-2, 5000-3, 5000-4, 5000-5,
#                                 5000-6, 5000-7
#              Active SE clocks:     0, 1, 2, 3, 4, 5, 6 , 7
#              Modulo se_clk gen:    none
#              Compare result with:  expected error reporting
#              Additional details:
#
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_4_5_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_8_9_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;
cccscr_12_13_mem[tc] = 8'b_1_____0_____1_____0_____0_____1_____1_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// x_dead_mem[tc] -- Bits that are reset (0) in this 89-bit register for each testcase will cause
// the corresponding X-inputs to the chip to be forced low.  If a bit is set (1),
// the corresponding X-input to the chip is unaffected.  Default: all bits set (1).
//             Bit assignments are as follows:
//             =============                 ================
//             Bit locations                 Signals affected
//             =============                 ================
//             [7:0]                         DUMP_EN[7:0]
//             [15:8]                        DVALID[7:0]
//             [23:16]                       SE_CLK[7:0]
//             [55:24]                       PHASE-[0...7][3:0] (e.g. [27:24] = PHASE-0[3:0]
//             [87:56]                       SDATA-[0...7][3:0] (e.g. [59:56] = SDATA-1[3:0]
//             [88]                          TIMESTAMP.
//
//             y_dead_mem[tc] -- Same as x_dead_mem, only for Y inputs to the chip.

x_dead_mem[tc]= 'h1aaaaaaaaaaaaaaaaffffff;
y_dead_mem[tc]= 'h155555555555555555ffffff;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};
```

```
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 2500;
dump_period_mem1[tc] = 2500;
dump_period_mem2[tc] = 2500;
dump_period_mem3[tc] = 2500;
dump_period_mem4[tc] = 2500;
dump_period_mem5[tc] = 2500;
dump_period_mem6[tc] = 2500;
dump_period_mem7[tc] = 2500;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

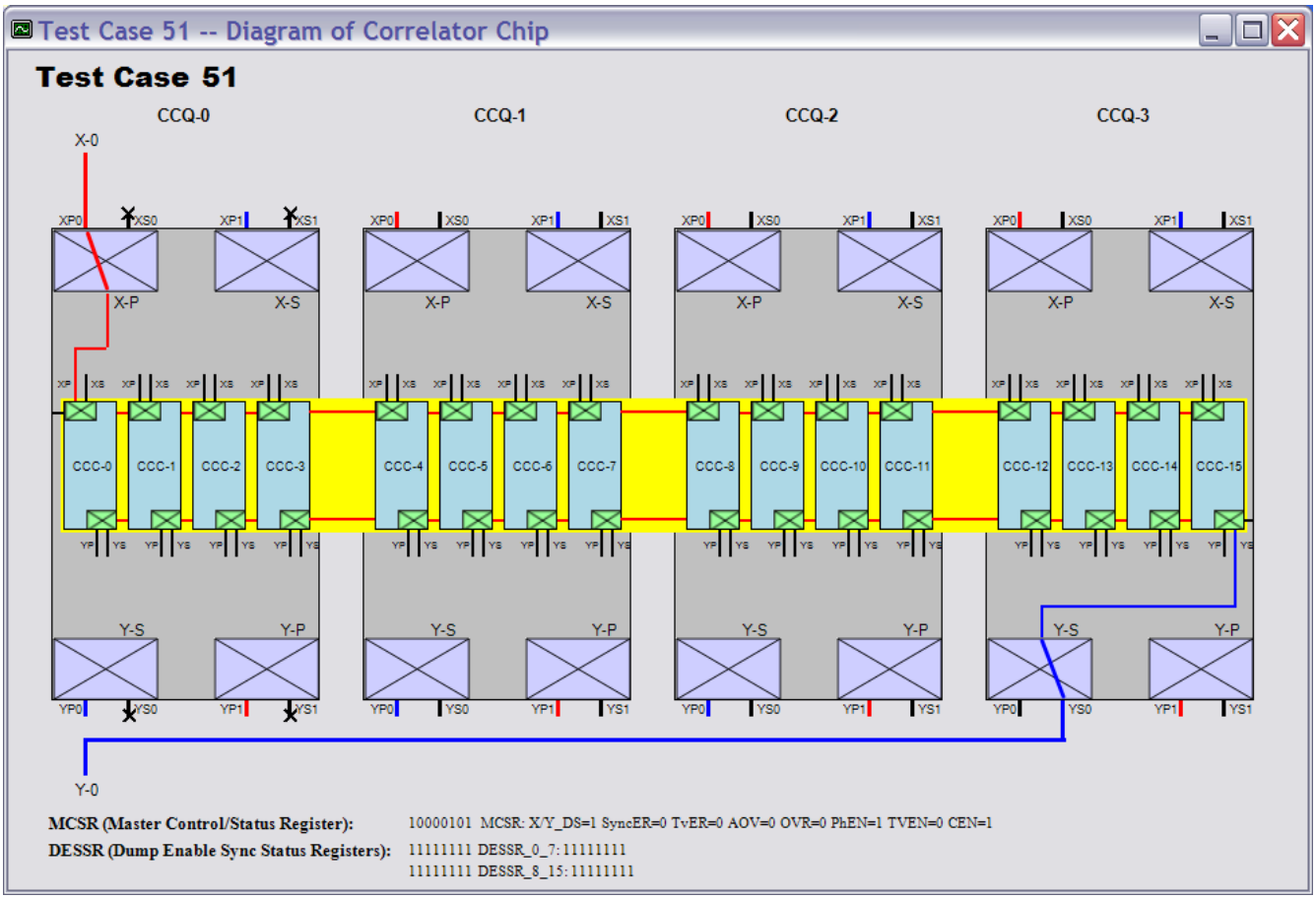
test_duration_mem[tc] = 2500;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.52 Test case 51




```
// ********************************************************************************************************************
// File name : tc51_apr24-05_v1.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V1
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc51_apr24-05_v1.v     Apr 24, 2005    1                creation (recycle 51)
//
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 51;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 51:
# ver: v1,05apr24
# Synopsis:   NO CONCAT., ALL POLN, 8 INPUTS, SELECTED STREAMS RESET
# Description:
#              Purpose of Test:      detection of missing phase or data streams
#              Concatenation:        1(0)1, 1(1)0, 0(2)1, 0(3)0, 3(4)3, 3(5)2, 2(6)3,
#                                 2(7)2, 5(8)5, 5(9)4, 4(10)5, 4(11)4, 7(12)7, 7(13)6,
#                                 6(14)7, 6(15)6
#              Num of Dumps-input:   1-0, 1-1, 1-2, 1-3, 1-4, 1-5, 1-6, 1-7
#              Dump Control:         X
#              Num samples/dump-inp: 5000-0, 5000-1, 5000-2, 5000-3, 5000-4, 5000-5,
#                                 5000-6, 5000-7
#              Distinct samp/dump:  5000-0, 5000-1, 5000-2, 5000-3, 5000-4, 5000-5,
#                                 5000-6, 5000-7
#              Active SE clocks:     0, 1, 2, 3, 4, 5, 6 , 7
#              Modulo se_clk gen:    none
#              Compare result with:  expected error reporting
#              Additional details:
#
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_4_5_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____0_____1;
cccscr_8_9_mem[tc] = 8'b___1_____0_____1_____0_____0_____1_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;
cccscr_12_13_mem[tc] = 8'b_1_____0_____1_____0_____0_____1_____1_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____1_____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0__________0__________0__________0__________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0__________0__________0__________0__________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// x_dead_mem[tc] -- Bits that are reset (0) in this 89-bit register for each testcase will cause
// the corresponding X-inputs to the chip to be forced low.  If a bit is set (1),
// the corresponding X-input to the chip is unaffected.  Default: all bits set (1).
//             Bit assignments are as follows:
//             =============                 ================
//             Bit locations                 Signals affected
//             =============                 ================
//             [7:0]                         DUMP_EN[7:0]
//             [15:8]                        DVALID[7:0]
//             [23:16]                       SE_CLK[7:0]
//             [55:24]                       PHASE-[0...7][3:0] (e.g. [27:24] = PHASE-0[3:0]
//             [87:56]                       SDATA-[0...7][3:0] (e.g. [59:56] = SDATA-1[3:0]
//             [88]                          TIMESTAMP.
//
//             y_dead_mem[tc] -- Same as x_dead_mem, only for Y inputs to the chip.

y_dead_mem[tc]= 'h1aaaaaaaaaaaaaaaaffffff;
x_dead_mem[tc]= 'h155555555555555555ffffff;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};
```

```
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 2500;
dump_period_mem1[tc] = 2500;
dump_period_mem2[tc] = 2500;
dump_period_mem3[tc] = 2500;
dump_period_mem4[tc] = 2500;
dump_period_mem5[tc] = 2500;
dump_period_mem6[tc] = 2500;
dump_period_mem7[tc] = 2500;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

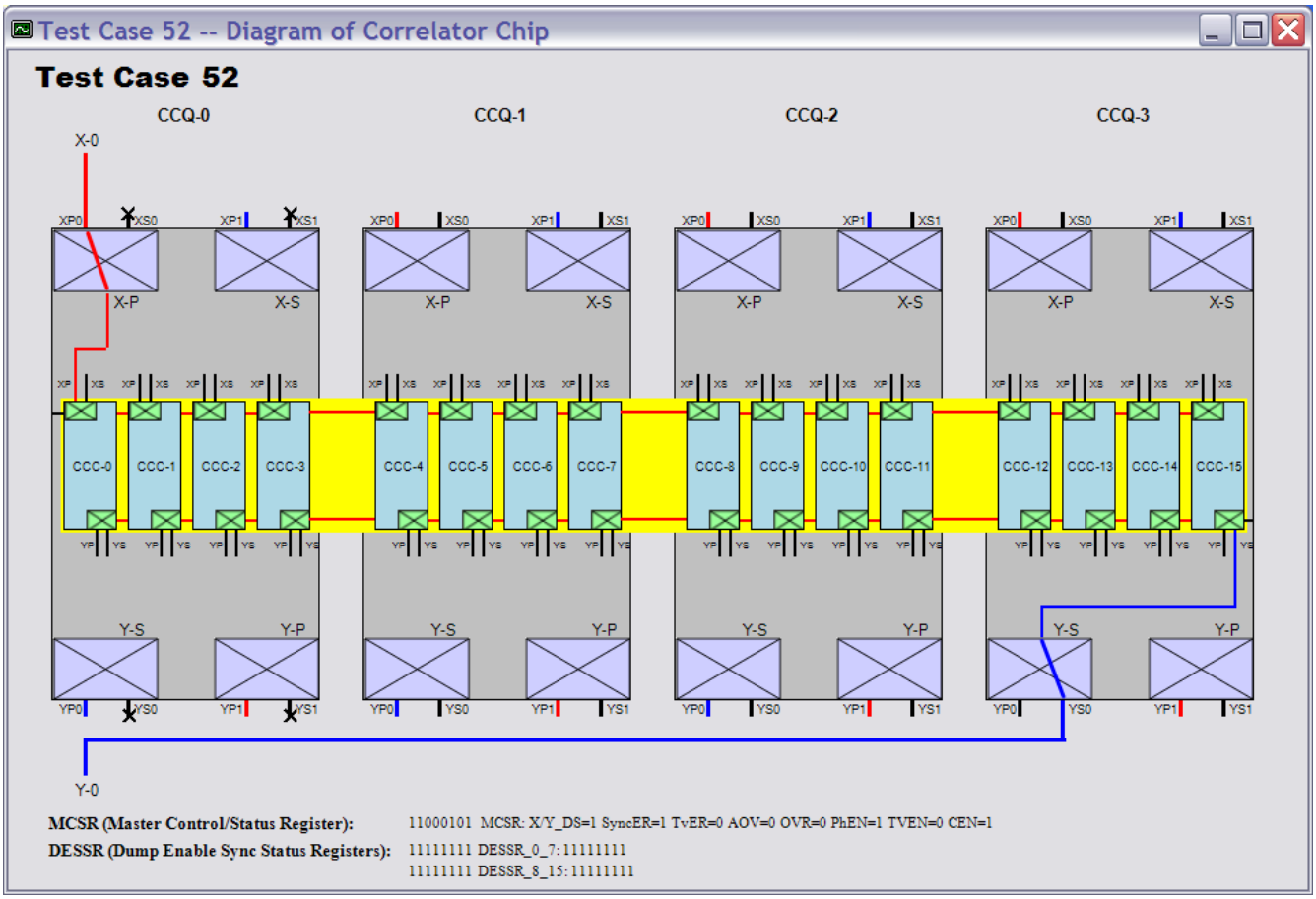
test_duration_mem[tc] = 2500;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.53 Test case 52




```
// ********************************************************************************************************************
// File name : tc52_may03-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc52_aug31-04_v1.v     Aug 31, 2004    1                creation
// tc52_sep01-04_v1.v     Sep 01, 2004    1                increase offset to definite excess
// tc52_may03-05_v2.v     May 03, 2005    2                modify for new test bench; add verification mode
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 52;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 52:
# ver: v2,05may03
# Synopsis:    CONCAT ALL, X DUMP, INP 0, EXCEED POS Y-X OFFSET INP BUFFER LIMIT
# Description:
#              Purpose of Test:      SCHID_SYNC reporting in status register and buffer
#                                 limits
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 500-0
#              Distinct samp/dump:   500-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  check SCHID_SYNC in X|Y-STATUS
#              Additional details:   exceed Y-X offset of input buffer
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         -----------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         -----------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______0______0______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_6_7_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_10_11_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_14_15_mem[tc] = 8'b_1______0______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4         B3        B2         B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 250;
dump_period_mem1[tc] = 250;
dump_period_mem2[tc] = 250;
dump_period_mem3[tc] = 250;
dump_period_mem4[tc] = 250;
dump_period_mem5[tc] = 250;
```

```
dump_period_mem6[tc] = 250;
dump_period_mem7[tc] = 250;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 70;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

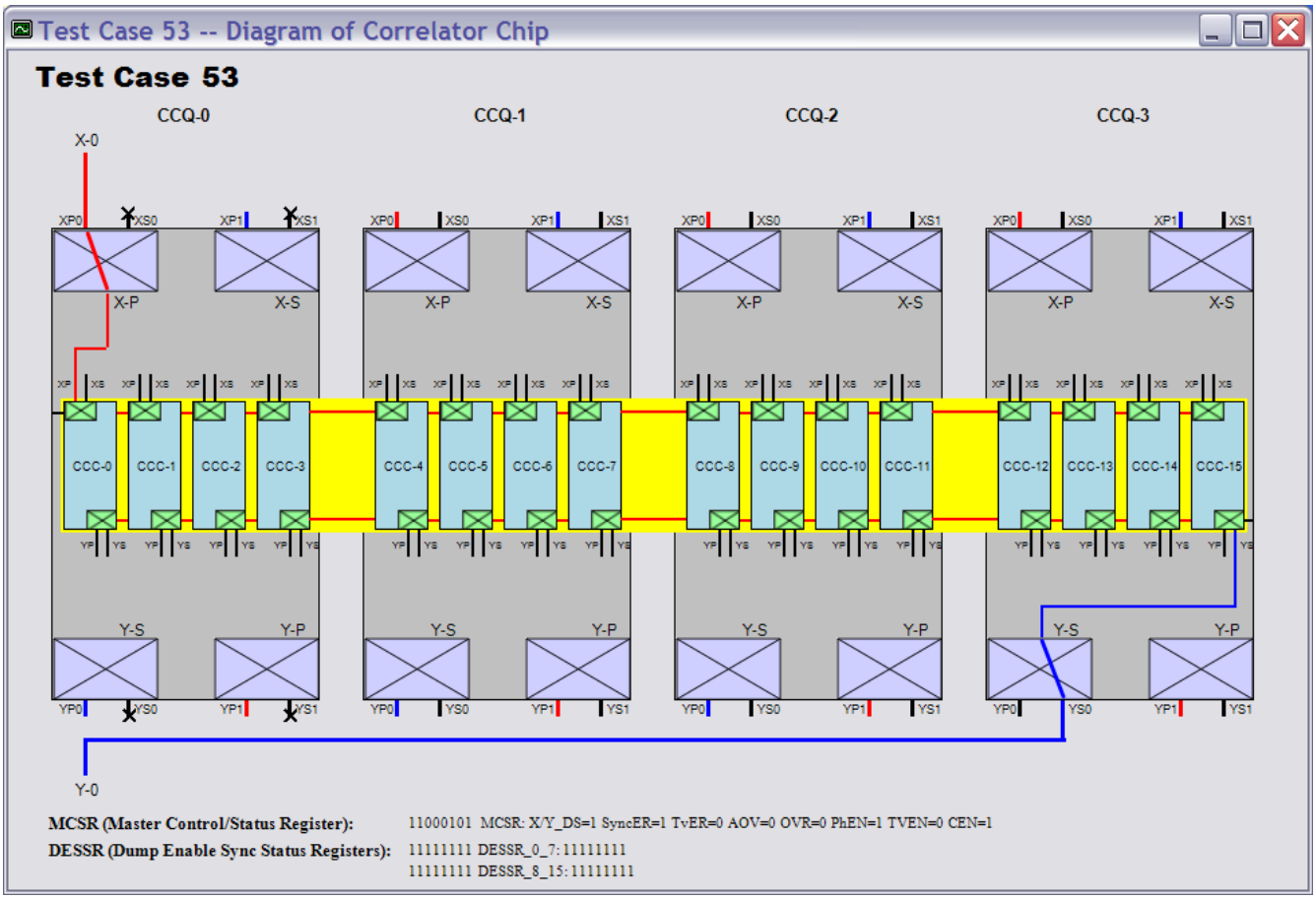
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 250;
```

4.54 Test case 53




```
// ********************************************************************************************************************
// File name : tc53_may03-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc53_sep01-04_v1.v     Sep 1,  2004    1                creation
// tc53_may03-05_v2.v     May 03, 2005    2                modify for new test bench; add verification mode
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 53;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 53:
# ver: v2,05may03
# Synopsis:    CONCAT ALL, X DUMP, INP 0, EXCEED NEG Y-X OFFSET INP BUFFER LIMIT
# Description:
#              Purpose of Test:      SCHID_SYNC reporting in status register and buffer
#                                 limits
#              Concatenation:        0(0:15)0
#              Num of Dumps:         1
#              Dump Control:         X
#              Num samples/dump-inp: 500-0
#              Distinct samp/dump:   500-0
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  check SCHID_SYNC in X|Y-STATUS
#              Additional details:   exceed Y-X offset of input buffer
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_1_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________1_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies since only input 0 is valid
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 250;
dump_period_mem1[tc] = 250;
dump_period_mem2[tc] = 250;
dump_period_mem3[tc] = 250;
dump_period_mem4[tc] = 250;
dump_period_mem5[tc] = 250;
```

```
dump_period_mem6[tc] = 250;
dump_period_mem7[tc] = 250;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = -70;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

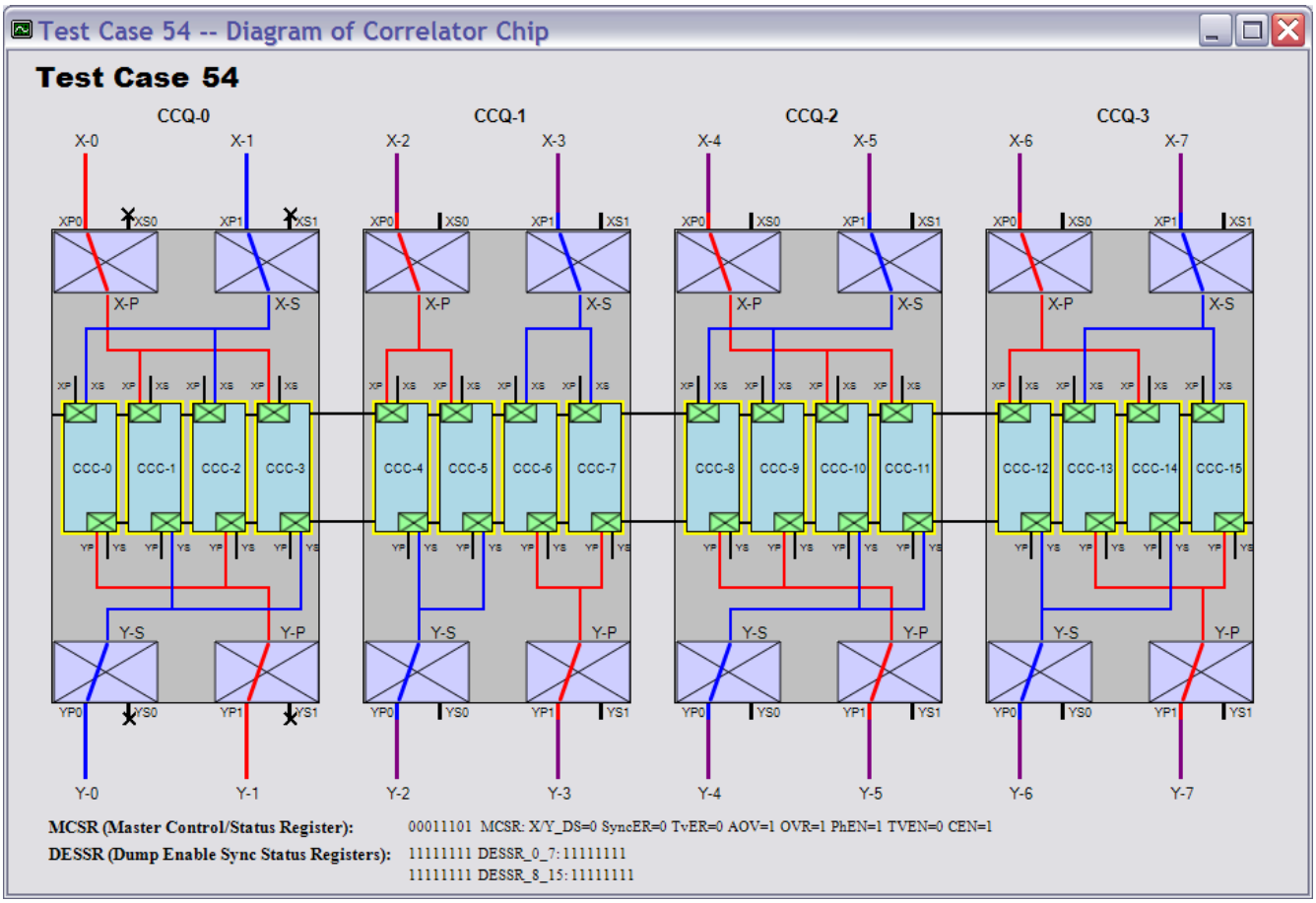
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 250;
```

4.55 Test case 54




```
// ********************************************************************************************************
// File name : tc54_apr14-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc54_nov26-04_v1.v     Nov 26, 2004    1                creation
// tc54_nov27-04_v1.v     Nov 27, 2004    1                modify Y_schid and Y_dump offsets
// tc54_dec07-04_v1.v     Dec  7, 2004    1                typo in description
// tc54_apr13-05_v2.v     Apr 13, 2005    1                reduce run time; no overflow
// tc54_apr14-05_v2.v     Apr 14, 2005    1                reduce run time; increase dump rate; no overflow
// ********************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 54;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 54:
# ver: v2,05apr14
# Synopsis:   VERIFICATION, OUTPUT FRAME GENERATION, MULTIPLE DUMP RATE, OVERRUN
# Description:
#              Purpose of Test:      verification of frame builder, dump overrun
#                                 detection and capture with multiple dump rate
#              Concatenation:        1(0)1, 0(1)0, 1(2)1, 0(3)0, 2(4)2, 2(5)2, 3(6)3,
#                                 3(7)3, 5(8)5, 5(9)5, 4(10)4, 4(11)4, 6(12)6, 7(13)7,
#                                 6(14)6, 7(15)7
#              Num of Dumps-input:   80-0, 40-1, 20-2, 20-3, 20-4, 20-5, 10-6, 10-7
#              Dump Control:         Y
#              Num samples/dump-inp: 1000-0, 2000-1, 4000-2, 4000-3, 4000-4, 4000-5,
#                                 8000-6, 8000-7
#              Distinct samp/dump: 500-0, 666-1, 1000-2, 800-3, 666-4, 571-5,
#                                 1000-6, 888-7
#              Active SE clocks:     all
#              Modulo se_clk gen:    modulo (input# + 2)
#              Compare result with:  previous verification run
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 ----------|
//                        ---------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____1_____0;
cccscr_2_3_mem[tc] = 8'b___1_____0_____0_____1_____0_____1_____1_____0;
cccscr_4_5_mem[tc] = 8'b___1_____0_____0_____1_____1_____0_____0_____1;
cccscr_6_7_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_8_9_mem[tc] = 8'b___0_____1_____1_____0_____0_____1_____1_____0;
cccscr_10_11_mem[tc] = 8'b_1_____0_____0_____1_____1_____0_____0_____1;
cccscr_12_13_mem[tc] = 8'b_0_____1_____1_____0_____1_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_0_____1_____1_____0_____1_____0_____0_____1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-----------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__1_______1_______1_______1_______1_______1_______1_______1;
y_se_clkmod_mem[tc] = 8'b__1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

dts_len_mem[tc]= 312;  // increase size of dts_len to 312 bits total.

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

fill3 = 3'b111;                              // 3 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
```

```
// dump_en is transmitted LSB first.

// Holdoff held high to disable correlation until lag shift registers
// fill up with data. Maximum holdoff size is 280 bits in this test case.

dump_command = 3'b011;                      // 3 bits:
phase_bin = 43691;                          // 16 bits
recirc_blk = 81;                            // 8 bits
holdoff = 'h1fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 257 bits
y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b010;                      // 3 bits:
phase_bin = 43693;                          // 16 bits
recirc_blk = 83;                            // 8 bits
holdoff = 'h7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 259 bits
y_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b110;                      // 3 bits:
phase_bin = 43695;                          // 16 bits
recirc_blk = 85;                            // 8 bits
holdoff = 'h1fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 261 bits
y_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b000;                      // 3 bits:
phase_bin = 43697;                          // 16 bits
recirc_blk = 87;                            // 8 bits
holdoff = 'h7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 263 bits
y_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

fill_sync = 2'b01;
dump_command = 3'b011;                      // 3 bits:
phase_bin = 43690;                          // 16 bits
recirc_blk = 80;                            // 8 bits
holdoff = 'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 256 bits
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b111;                      // 3 bits:
phase_bin = 43692;                          // 16 bits
recirc_blk = 82;                            // 8 bits
holdoff = 'h3fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 258 bits
y_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 43694;                          // 16 bits
recirc_blk = 84;                            // 8 bits
holdoff = 'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 260 bits
y_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b001;                      // 3 bits:
phase_bin = 43696;                          // 16 bits
recirc_blk = 86;                            // 8 bits
holdoff = 'h3fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff; // 262 bits
y_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
// The X timestamp is ignored because Y controls dumping.
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

ts_fillsync = 2'b01;
ts_0 = 'haaaaaaaa;
ts_1 = 'h55555555;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000
```

```
dump_period_mem0[tc] = 500;
dump_period_mem1[tc] = 1000;
dump_period_mem2[tc] = 2000;
dump_period_mem3[tc] = 2000;
dump_period_mem4[tc] = 2000;
dump_period_mem5[tc] = 2000;
dump_period_mem6[tc] = 4000;
dump_period_mem7[tc] = 4000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = -30;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 21;
y_sbid_mem[tc] = 10;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
```

```
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

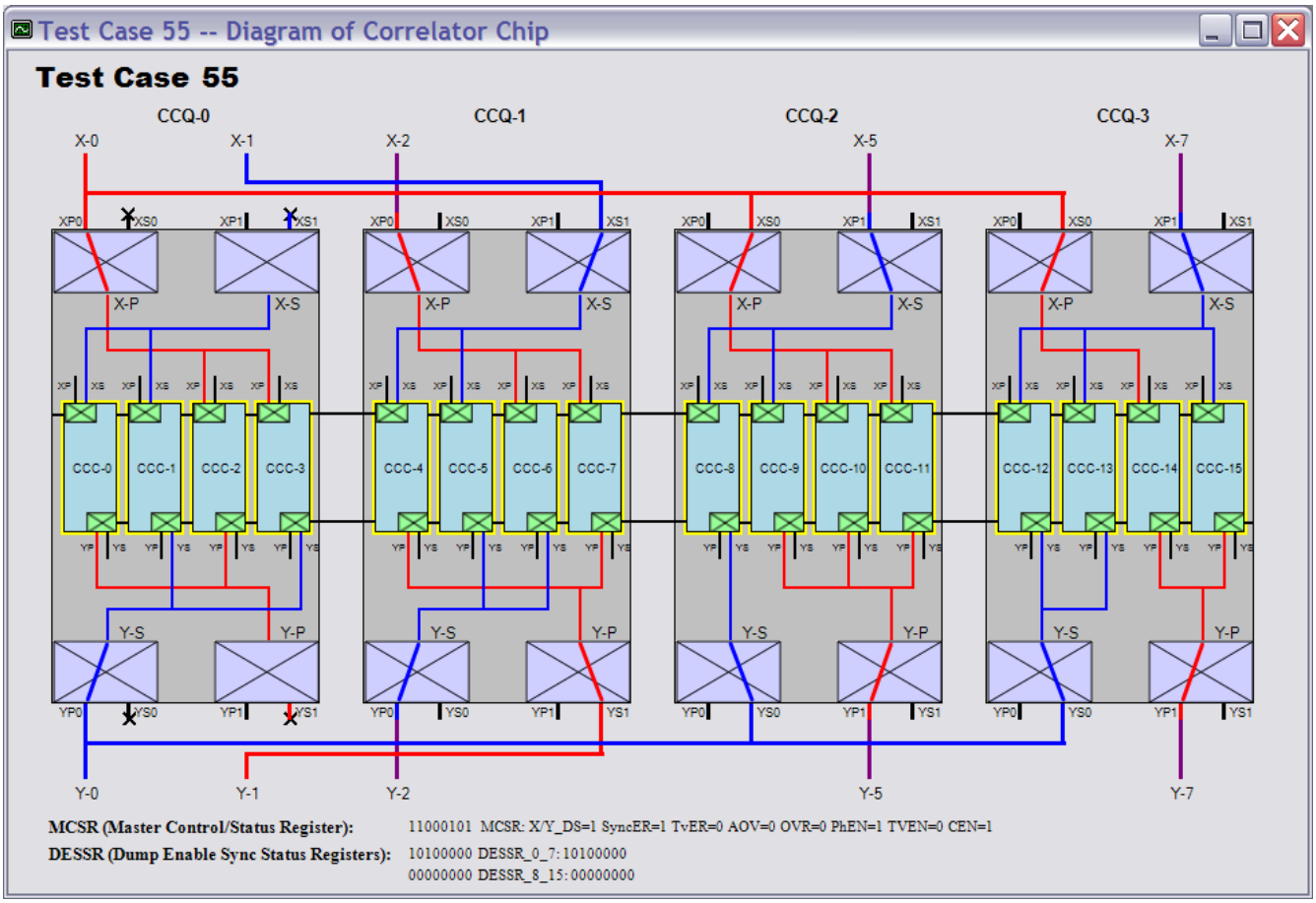
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 40000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);
```

4.56 Test case 55




```
// ********************************************************************************************************************
// File name : tc55_apr25-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                 phone: 250-490-4346
//                 email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc55_nov27-04_v1.v     Nov 27, 2004    1                creation
// tc55_nov29-04_v1.v     Nov 29, 2004    1                change description field, timestamp fill sync.
// tc55_apr13-05_v2.v     Apr 13, 2005    2                shorten dump time to create overrun on all cells
// tc55_apr25-05_v2.v     Apr 25, 2005    2                remove input errors; reduce X-Y DUMP_EN offset
//
// ********************************************************************************************************************
//
```

```
//
// ***********************************************************************************************************
// ***********************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***********************************************************************************************************
// ***********************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 55;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 55:
# ver: v2,05apr25
# Synopsis:   VERIFICATION, OUTPUT FRAME GENERATION, HIGH DUMP RATE, OVERRUN ALL
# Description:
#              Purpose of Test:      verification of frame builder and dump
#                                 overrun detection
#              Concatenation:        7(0)-, 7(1)0, 0(2)-, 0(3)0, 1(4)1, 1(5)2, 2(6)2,
#                                 2(7)1, 5(8)0, 5(9)5, 0(10)5, 0(11)5, 7(12)0, 7(13)0,
#                                 0(14)7, 7(15)7
#              Num of Dumps-input:   20-0, 10-1, 10-2, 20-5, 20-7
#              Dump Control:         X
#              Num samples/dump-inp: 512-0, 1024-1, 1024-2, 512-5, 512-7,
#              Distinct samp/dump:   256-0, 512-1, 512-2, 256-5, 256-7
#              Active SE clocks:     all
#              Modulo se_clk gen:    modulo 2 on all
#              Compare result  with: previous verification run
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         -----------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         -----------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______1______0______0______1______1______0;
cccscr_2_3_mem[tc] = 8'b___1______0______0______1______0______1______0______1;
cccscr_4_5_mem[tc] = 8'b___1______0______1______0______0______1______1______0;
cccscr_6_7_mem[tc] = 8'b___0______1______0______1______1______0______0______1;
cccscr_8_9_mem[tc] = 8'b___0______1______1______0______1______0______1______0;
cccscr_10_11_mem[tc] = 8'b_0______1______0______1______0______1______0______1;
cccscr_12_13_mem[tc] = 8'b_1______0______1______0______1______0______1______0;
cccscr_14_15_mem[tc] = 8'b_0______1______1______0______0______1______0______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________0_________1________0________1_________0_________1_________0_________1;
ccqr_2_3_mem[tc] = 8'b_______0________1_________0________1________0_________1_________0_________1_________0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b____1_______0_______1_______0_______0_______1_______1_______1;
y_se_clk_mem[tc] = 8'b____1_______0_______1_______0_______0_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__1_______0_______1_______0_______0_______1_______1_______1;
y_se_clkmod_mem[tc] = 8'b__1_______0_______1_______0_______0_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______0_______1_______0_______0_______1_______1_______1;
y_dvalid_mem[tc] = 8'b____1_______0_______1_______0_______0_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

dts_len_mem[tc]= 312;  // increase size of dts_len to 312 bits total.

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

fill3 = 3'b111;                              // 3 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.
```

```
// Holdoff held high to disable correlation until lag shift registers
// fill up with data. Maximum holdoff size is 280 bits in this test case.

dump_command = 3'b000;                      // 3 bits:
phase_bin = 21841;                          // 16 bits
recirc_blk = 171;                           // 8 bits
holdoff = 'h1fffffffffffffffffffffffffffffffffffffffffff; // 177 bits
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b101;                      // 3 bits:
phase_bin = 21843;                          // 16 bits
recirc_blk = 173;                           // 8 bits
holdoff = 'h7fffffffffffffffffffffffffffffffffffffffffff; // 175 bits
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b011;                      // 3 bits:
phase_bin = 21845;                          // 16 bits
recirc_blk = 175;                           // 8 bits
holdoff = 'h1fffffffffffffffffffffffffffffffffffffffffff; // 173 bits
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b111;                      // 3 bits:
phase_bin = 21845;                          // 16 bits
recirc_blk = 170;                           // 8 bits
holdoff = 'hffffffffffffffffffffffffffffffffffffffffffff; // 176 bits
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b101;                      // 3 bits:
phase_bin = 21842;                          // 16 bits
recirc_blk = 172;                           // 8 bits
holdoff = 'h3fffffffffffffffffffffffffffffffffffffffffff; // 178 bits
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b110;                      // 3 bits:
phase_bin = 43694;                          // 16 bits
recirc_blk = 84;                            // 8 bits
holdoff = 'hffffffffffffffffffffffffffffffffffffffffffff; // 176 bits
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

dump_command = 3'b001;                      // 3 bits:
phase_bin = 43696;                          // 16 bits
recirc_blk = 86;                            // 8 bits
holdoff = 'h3fffffffffffffffffffffffffffffffffffffffffff; // 174 bits
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

fill_sync = 2'b11; // bad fill sync
dump_command = 3'b010;                      // 3 bits:
phase_bin = 21847;                          // 16 bits
recirc_blk = 177;                           // 8 bits
holdoff = 'h7fffffffffffffffffffffffffffffffffffffffffff; // 175 bits
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 'haaaaaaaa;
ts_1 = 'h55555555;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 256;
```

```
dump_period_mem1[tc] = 512;
dump_period_mem2[tc] = 512;
dump_period_mem3[tc] = 256;
dump_period_mem4[tc] = 256;
dump_period_mem5[tc] = 256;
dump_period_mem6[tc] = 256;
dump_period_mem7[tc] = 256;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 1024;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 32;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 5;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 10;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 170;
y_station_id_mem[tc] = 85;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 10;
y_sbid_mem[tc] = 21;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
```

```
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION

test_duration_mem[tc] = 5120;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.57 Test case 56

```
// ********************************************************************************************************************
// File name : tc56_apr20-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc56_dec09-04_v1.v     Dec  9, 2004    1                creation
// tc56_apr20-05_v2.v     Apr 20, 2005    2                add verification and split acc. modes
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data'; 'x' in place of inpY indicates
// the chip is in autocorrelation mode.
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 56;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 56:
# ver: v2,05apr20
# Synopsis:  AUTOCORRELATION INP 0, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      autocorrelation mode
#              Concatenation:        0(0:15)x
#              Num of Dumps-input:   1-0
#              Dump Control:         X
#              Num samples/dump-inp: 20000-0
#              Distinct samp/dump:   10000-0
#              Active SE clocks:     0
#              Modulo se_clk gen:    2-0
#              Compare result with:  expected general behaviour
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                   MCSR register:
//                   ------------------------------------------------------
//                   X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                   ------------------------------------------------------
//                     B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_2_3_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
// On (1) or off (0).
// Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
// If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
// However, this can be overridden by set_modulo_mode_mem below.
// E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
// This could result in the detection of input synchronization errors since a given se_clk may not
// be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______1;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
```

```
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
```

```
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***************************************************************************************************
```

4.58 Test case 57

```
// ********************************************************************************************************************
// File name : tc57_apr20-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc57_dec09-04_v1.v     Dec  9, 2004    1                creation
// tc57_apr20-05_v2.v     Apr 20, 2005    2                add verification and split acc. modes
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data'; 'x' in place of inpY indicates
// the chip is in autocorrelation mode.
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 57;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 57:
# ver: v2,05apr20
# Synopsis:  AUTOCORRELATION INP 1, SYNC ERRORS, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      autocorrelation mode
#              Concatenation:        1(0:15)x
#              Num of Dumps-input:   1-1
#              Dump Control:         X
#              Num samples/dump-inp: 20000-1
#              Distinct samp/dump:   10000-1
#              Active SE clocks:     1
#              Modulo se_clk gen:    2-1
#              Compare result with:  expected general behaviour
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b__0_____0_____0_____0_____0_____1_____0_____0;
cccscr_2_3_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
// On (1) or off (0).
// Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
// If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
// However, this can be overridden by set_modulo_mode_mem below.
// E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
// This could result in the detection of input synchronization errors since a given se_clk may not
// be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______1_______0;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
```

```
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 130;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
```

```
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.59 Test case 58

```
// ********************************************************************************************************************
// File name : tc58_apr20-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc58_dec09-04_v1.v     Dec  9, 2004    1                creation
// tc58_apr20-05_v2.v     Apr 20, 2005    1                add verification and split acc. modes
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data'; 'x' in place of inpY indicates
// the chip is in autocorrelation mode.
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 58;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 58:
# ver: v2,05apr20
# Synopsis:  AUTOCORRELATION INP 2, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      autocorrelation mode
#              Concatenation:        2(0:15)x
#              Num of Dumps-input:   1-2
#              Dump Control:         X
#              Num samples/dump-inp: 20000-2
#              Distinct samp/dump:   10000-2
#              Active SE clocks:     2
#              Modulo se_clk gen:    2-2
#              Compare result with:  expected general behaviour
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b__0_____0_____0_____0_____1_____0_____0_____0;
cccscr_2_3_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______1_______0_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
// On (1) or off (0).
// Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
// If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
// However, this can be overridden by set_modulo_mode_mem below.
// E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
// This could result in the detection of input synchronization errors since a given se_clk may not
// be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______1_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______1_______0_______0;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
```

```
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
```

```
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.60 Test case 59

```
// ********************************************************************************************************************
// File name : tc59_apr20-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc59_dec09-04_v1.v     Dec  9, 2004    1                creation
// tc59_apr20-05_v2.v     Apr 20, 2005    1                add verification and split acc. modes
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data'; 'x' in place of inpY indicates
// the chip is in autocorrelation mode.
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 59;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 59:
# ver: v2,05apr20
# Synopsis:  AUTOCORRELATION INP 3, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      autocorrelation mode
#              Concatenation:        3(0:15)x
#              Num of Dumps-input:   1-3
#              Dump Control:         X
#              Num samples/dump-inp: 20000-3
#              Distinct samp/dump:   10000-3
#              Active SE clocks:     3
#              Modulo se_clk gen:    2-3
#              Compare result with:  expected general behaviour
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-----------------------------------------------------------------------------------------------------------------

//                   MCSR register:
//                   ------------------------------------------------------
//                   X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                   ------------------------------------------------------
//                     B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b__0_____0_____0_____0_____1_____1_____0_____0;
cccscr_2_3_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______1_______0_______0_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
// On (1) or off (0).
// Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
// If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
// However, this can be overridden by set_modulo_mode_mem below.
// E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
// This could result in the detection of input synchronization errors since a given se_clk may not
// be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______1_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______1_______0_______0_______0;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
```

```
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
```

```
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***************************************************************************************************
```

4.61 Test case 60

```
// ********************************************************************************************************************
// File name : tc60_apr07-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc60_dec09-04_v1.v     Dec  9, 2004    1                creation
// tc60_apr07-05_v2.v     Apr  7, 2005    2                add verification option
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data'; 'x' in place of inpY indicates
// the chip is in autocorrelation mode.
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 60;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 60:
# ver: v2,05apr07
# Synopsis:  AUTOCORRELATION INP 4, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      autocorrelation mode
#              Concatenation:        4(0:15)x
#              Num of Dumps-input:   1-4
#              Dump Control:         X
#              Num samples/dump-inp: 20000-4
#              Distinct samp/dump:   10000-4
#              Active SE clocks:     4
#              Modulo se_clk gen:    2-4
#              Compare result with:  expected general behaviour
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                   MCSR register:
//                   ------------------------------------------------------
//                   X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                   ------------------------------------------------------
//                     B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b__0_____0_____0_____1_____0_____0_____0_____0;
cccscr_2_3_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______1_______0_______0_______0_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
// On (1) or off (0).
// Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
// If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
// However, this can be overridden by set_modulo_mode_mem below.
// E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
// This could result in the detection of input synchronization errors since a given se_clk may not
// be 1 when schid_frame is asserted.
//
//             modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______1_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                       Input line number
//                       ---------------------------------------------------------------
//                          7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                       ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______0_______0_______1_______0_______0_______0_______0;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
```

```
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***************************************************************************************************************
```

4.62 Test case 61

```
// ********************************************************************************************************************
// File name : tc61_aug07-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE           VERSION          COMMENTS
// tc61_dec09-04_v1.v     Dec  9, 2004    1                creation
// tc61_apr20-05_v2.v     Apr 20, 2005    2                add verification and split acc. modes
// tc61_aug07-05_v2.v     Aug  7, 2005    2                fix version date string
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data'; 'x' in place of inpY indicates
// the chip is in autocorrelation mode.
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 61;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 61:
# ver: v2,05aug07
# Synopsis:  AUTOCORRELATION INP 5, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      autocorrelation mode
#              Concatenation:        5(0:15)x
#              Num of Dumps-input:   1-5
#              Dump Control:         X
#              Num samples/dump-inp: 20000-5
#              Distinct samp/dump:   10000-5
#              Active SE clocks:     5
#              Modulo se_clk gen:    2-5
#              Compare result with:  expected general behaviour
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
```

```
//                        CCC0,1 switch config register: (same for CCC2...CCC15)
//                       |--------- CCC1 ----------|---------- CCC0 -----------|
//                       ---------------------------------------------------------
//                       Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                       ---------------------------------------------------------
//                        B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0______0______0______1______0______1______0______0;
cccscr_2_3_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_4_5_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_6_7_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_8_9_mem[tc] = 8'b___0______0______0______0______0______0______0______0;
cccscr_10_11_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_12_13_mem[tc] = 8'b_0______0______0______0______0______0______0______0;
cccscr_14_15_mem[tc] = 8'b_0______0______0______0______0______0______0______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                           B7        B6         B5         B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______1________1__________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1__________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______1_______0_______0_______0_______0_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
// On (1) or off (0).
// Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
// If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
// However, this can be overridden by set_modulo_mode_mem below.
// E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
// This could result in the detection of input synchronization errors since a given se_clk may not
// be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                       Input line number
//                       ------------------------------------------------------------------
//                          7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                       ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______1_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                       Input line number
//                       ------------------------------------------------------------------
//                          7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                       ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______1_______0_______0_______0_______0_______0;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
```

```
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;
```

```
//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ******************************************************************************************************************
```

4.63 Test case 62

```
// ******************************************************************************************************************
// File name : tc62_apr20-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ******************************************************************************************************************
// ************************    REVISION HISTORY    ******************************************************************
// ******************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc62_dec09-04_v1.v     Dec  9, 2004    1                creation
// tc62_apr20-05_v2.v     Apr 20, 2005    2                add verification and split acc. modes
//
//
//
// ******************************************************************************************************************
//
//
// ******************************************************************************************************************
// ******************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ******************************************************************************************************************
// ******************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data'; 'x' in place of inpY indicates
// the chip is in autocorrelation mode.
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 62;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 62:
# ver: v2,05apr20
# Synopsis:  AUTOCORRELATION INP 6, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      autocorrelation mode
#              Concatenation:        6(0:15)x
#              Num of Dumps-input:   1-6
#              Dump Control:         X
#              Num samples/dump-inp: 20000-6
#              Distinct samp/dump:   10000-6
#              Active SE clocks:     6
#              Modulo se_clk gen:    2-6
#              Compare result with:  expected general behaviour
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                   MCSR register:
//                   ------------------------------------------------------
//                   X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                   ------------------------------------------------------
//                     B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b__0_____0_____0_____1_____1_____0_____0_____0;
cccscr_2_3_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3         B2         B1         B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______1_______0_______0_______0_______0_______0_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
// On (1) or off (0).
// Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
// If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
// However, this can be overridden by set_modulo_mode_mem below.
// E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
// This could result in the detection of input synchronization errors since a given se_clk may not
// be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______1_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____0_______1_______0_______0_______0_______0_______0_______0;
y_dvalid_mem[tc] = 8'b____0_______0_______0_______0_______0_______0_______0_______0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
```

```
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
```

```
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.64 Test case 63

```
// ********************************************************************************************************************
// File name : tc63_apr20-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc63_dec09-04_v1.v     Dec  9, 2004    1                creation
// tc63_apr20-05_v2.v     Apr 20, 2005    2                add verification and split acc. modes
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data'; 'x' in place of inpY indicates
// the chip is in autocorrelation mode.
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 63;
test_description[tc] =
"# ------------------------------------------------------------------------------
# TEST 63:
# ver: v2,05apr20
# Synopsis:  AUTOCORRELATION INP 7, MOD 2 CLK, 1 DUMP ON X
# Description:
#              Purpose of Test:      autocorrelation mode
#              Concatenation:        7(0:15)x
#              Num of Dumps-input:   1-7
#              Dump Control:         X
#              Num samples/dump-inp: 20000-7
#              Distinct samp/dump:   10000-7
#              Active SE clocks:     7
#              Modulo se_clk gen:    2-7
#              Compare result with:  expected general behaviour
#              Additional details:
#
# ------------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                   MCSR register:
//                   ------------------------------------------------------
//                   X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                   ------------------------------------------------------
//                     B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b__0_____0_____0_____1_____1_____1_____0_____0;
cccscr_2_3_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b__0_____0_____0_____0_____0_____0_____0_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------------
//                           B7       B6         B5       B4         B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______1________1_________1________1_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________1_________1________1_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______0_______0_______0_______0_______0_______0_______0;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
// On (1) or off (0).
// Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
// If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
// However, this can be overridden by set_modulo_mode_mem below.
// E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
// This could result in the detection of input synchronization errors since a given se_clk may not
// be 1 when schid_frame is asserted.
//
//             modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__1_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ---------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ---------------------------------------------------------------
x_dvalid_mem[tc] = 8'b____1_______0_______0_______0_______0_______0_______0_______0;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

// these are dummies
x_dumpen0_mem[tc] = {70'bx,2'b0};
x_dumpen1_mem[tc] = {70'bx,2'b0};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
```

```
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem6[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
```

```
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 85;
y_station_id_mem[tc] = 170;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

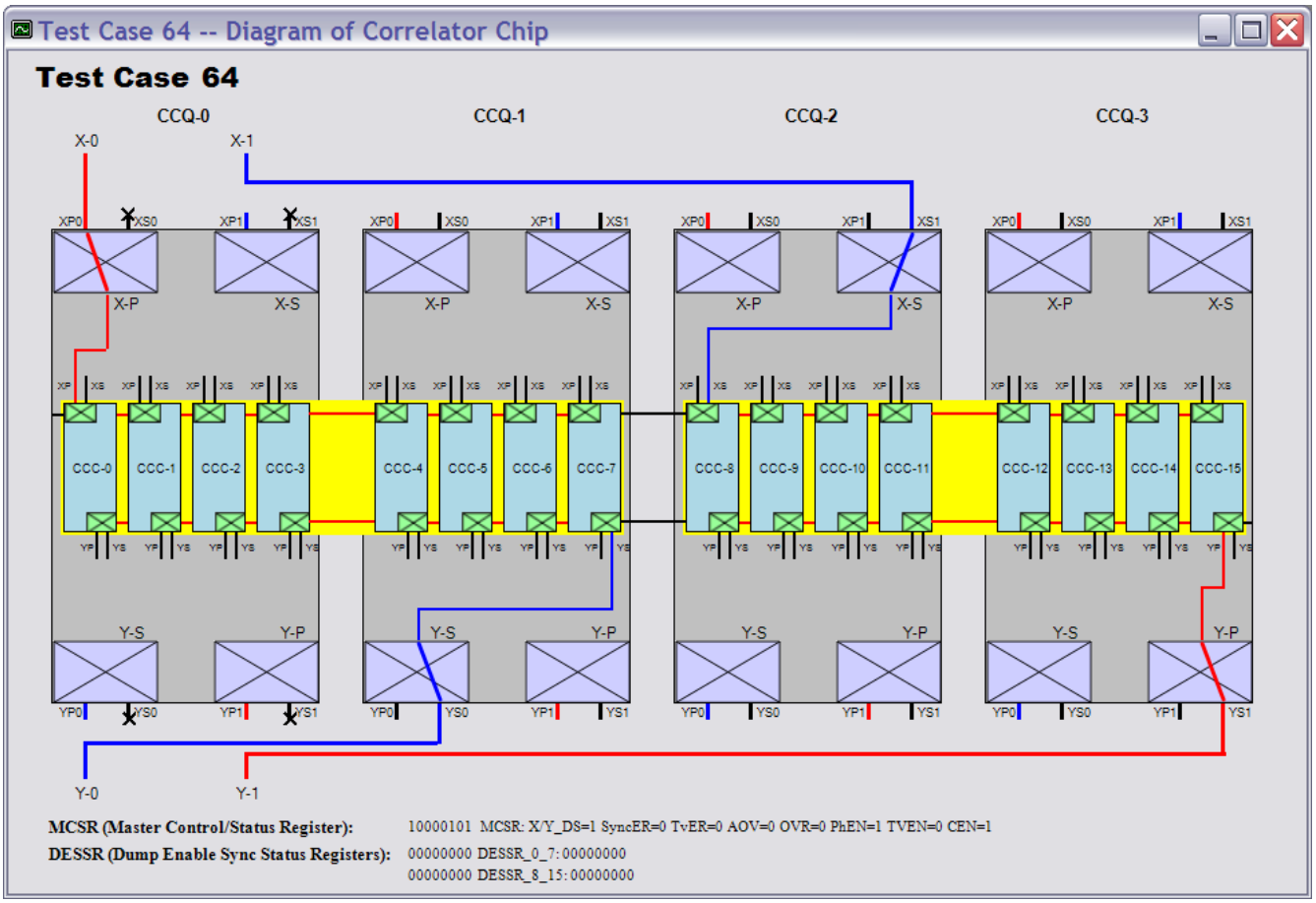
test_duration_mem[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.65 Test case 64




```
// ********************************************************************************************************************
// File name : tc64_apr24-05_v2.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V2
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc64_jan11-05_v1.v     Jan 11, 2005    1                creation
// tc64_apr24-05_v2.v     Apr 24, 2005    2                add verification mode and split acc. options;
//                                                         upgrade to new test bench
//
//
//
// ********************************************************************************************************************
//
```

```
//
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                                BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 64;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 64:
# ver: v2,05apr24
# Synopsis:   CONCAT 2 QUADS, 2-POL, MULT DUMP ON X, MOD 2 CLK
# Description:
#              Purpose of Test:      quad concatentation
#              Concatenation:        0(0:7)0, 1(8:15)1
#              Num of Dumps-input:   2-0, 2-1
#              Dump Control:         X
#              Num samples/dump-inp: 20000-0, 20000-1
#              Distinct samp/dump:   10000-0, 10000-1
#              Modulo se_clk gen:    MOD 2 CLK
#              Compare result with:  behavioural simulation test vectors for data set DS4
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_4_5_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____0_____0;
cccscr_6_7_mem[tc] = 8'b___1_____0_____0_____0_____0_____0_____0_____0;
cccscr_8_9_mem[tc] = 8'b___0_____0_____0_____0_____0_____0_____1_____0;
cccscr_10_11_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_12_13_mem[tc] = 8'b_0_____0_____0_____0_____0_____0_____0_____0;
cccscr_14_15_mem[tc] = 8'b_0_____1_____0_____0_____0_____0_____0_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________1__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______1________0__________0________0_________0_________0_________1_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------
//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

//            Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//            On (1) or off (0).
//            Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//            If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//            However, this can be overridden by set_modulo_mode_mem below.
//            E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//            This could result in the detection of input synchronization errors since a given se_clk may not
//            be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______1;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b_____0_______0_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};


// these are dummies
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};
x_dumpen6_mem[tc] = {70'bx,2'b0};
x_dumpen7_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 10000;
dump_period_mem1[tc] = 10000;
dump_period_mem2[tc] = 10000;
dump_period_mem3[tc] = 10000;
dump_period_mem4[tc] = 10000;
```

```
dump_period_mem5[tc] = 10000;
dump_period_mem5[tc] = 10000;
dump_period_mem7[tc] = 10000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

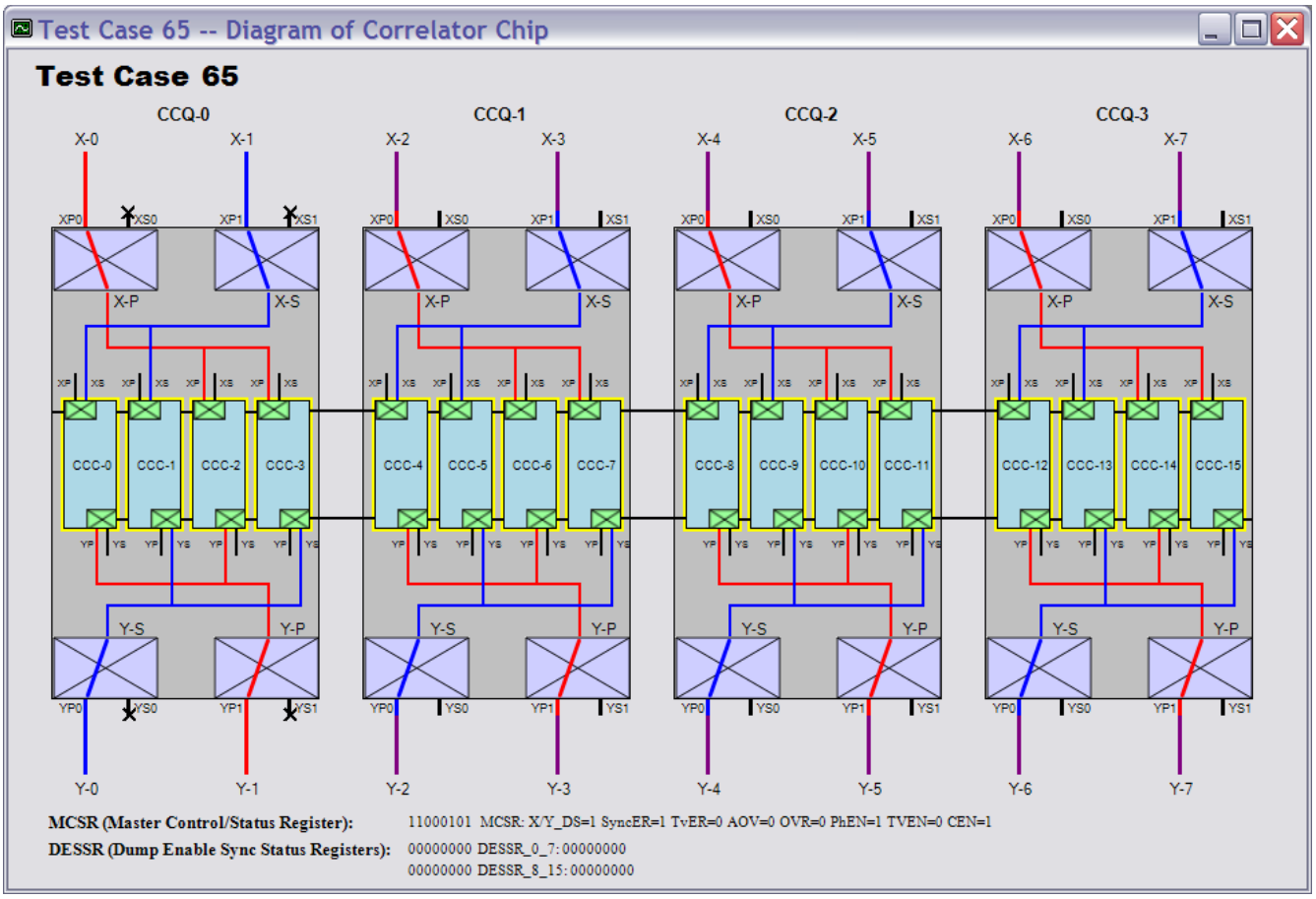
x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.
// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 20000;
```

4.66 Test case 65




```
// ********************************************************************************************************************
// File name : tc65_aug18-05_v1.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V1
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc65_aug18-05_v1.v     Aug 18, 2005    1                creation
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 65;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 65:
# ver: v1,05aug18
# Synopsis:   NO CONCAT., ALL POLN, 8 INPUTS, SELECTED STREAMS RESET
# Description:
#              Purpose of Test:      detection of missing phase or data streams
#              Concatenation:        1(0)1, 1(1)0, 0(2)1, 0(3)0, 3(4)3, 3(5)2, 2(6)3,
#                                 2(7)2, 5(8)5, 5(9)4, 4(10)5, 4(11)4, 7(12)7, 7(13)6,
#                                 6(14)7, 6(15)6
#              Num of Dumps-input:   1-0, 1-1, 1-2, 1-3, 1-4, 1-5, 1-6, 1-7
#              Dump Control:         X
#              Num samples/dump-inp: 5000-0, 5000-1, 5000-2, 5000-3, 5000-4, 5000-5,
#                                 5000-6, 5000-7
#              Distinct samp/dump:  5000-0, 5000-1, 5000-2, 5000-3, 5000-4, 5000-5,
#                                 5000-6, 5000-7
#              Active SE clocks:     0, 1, 2, 3, 4, 5, 6 , 7
#              Modulo se_clk gen:    none
#              Compare result with:  expected error reporting
#              Additional details:
#
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______1______0______0______1______1______0;
cccscr_2_3_mem[tc] = 8'b___1______0______0______1______0______1______0______1;
cccscr_4_5_mem[tc] = 8'b___1______0______1______0______0______1______1______0;
cccscr_6_7_mem[tc] = 8'b___1______0______0______1______0______1______0______1;
cccscr_8_9_mem[tc] = 8'b___1______0______1______0______0______1______1______0;
cccscr_10_11_mem[tc] = 8'b_1______0______0______1______0______1______0______1;
cccscr_12_13_mem[tc] = 8'b_1______0______1______0______0______1______1______0;
cccscr_14_15_mem[tc] = 8'b_1______0______0______1______0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ---------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ---------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b______0________0_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//             x_dead_mem[tc] -- Bits that are reset (0) in this 89-bit register for each testcase will cause
//             the corresponding X-inputs to the chip to be forced low.  If a bit is set (1),
//             the corresponding X-input to the chip is unaffected.  Default: all bits set (1).
//             Bit assignments are as follows:
//             =============                ================
//             Bit locations                Signals affected
//             =============                ================
//             [7:0]                        DUMP_EN[7:0]
//             [15:8]                       DVALID[7:0]
//             [23:16]                      SE_CLK[7:0]
//             [55:24]                      PHASE-[0...7][3:0] (e.g. [27:24] = PHASE-0[3:0]
//             [87:56]                      SDATA-[0...7][3:0] (e.g. [59:56] = SDATA-1[3:0]
//             [88]                         TIMESTAMP.
//             y_dead_mem[tc] -- Same as x_dead_mem, only for Y inputs to the chip.

x_dead_mem[tc]= 'h1f0f0f0f0f0f0f0f0aaffff;
y_dead_mem[tc]= 'h10f0f0f0f0f0f0f0f55ffff;

//             x_force_dvalid_on_mem[] and y_force_dvalid_on_mem[] memories. Â If they are not set, then
//             they have no effect, if they are set, then dvalid is force on during and after schid_frame when the
//             ccorrelator chip expects dvalid to be low. Â Each dvalid bit for x and y is independently settable.
//             E.g. x_force_dvalid_on_mem[tc]=8'b1010_1010; forces dvalids 7, 5, 3, and 1 to be '1' when they are
//             supposed to be '0', but leaves dvalids 6, 4, 2, and 0 in the normal condition.
x_force_dvalid_on_mem[tc]=8'b0101_0101;
y_force_dvalid_on_mem[tc]=8'b1010_1010;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
```

```
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 2500;
dump_period_mem1[tc] = 2500;
dump_period_mem2[tc] = 2500;
dump_period_mem3[tc] = 2500;
dump_period_mem4[tc] = 2500;
dump_period_mem5[tc] = 2500;
dump_period_mem6[tc] = 2500;
dump_period_mem7[tc] = 2500;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------
```

```
// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

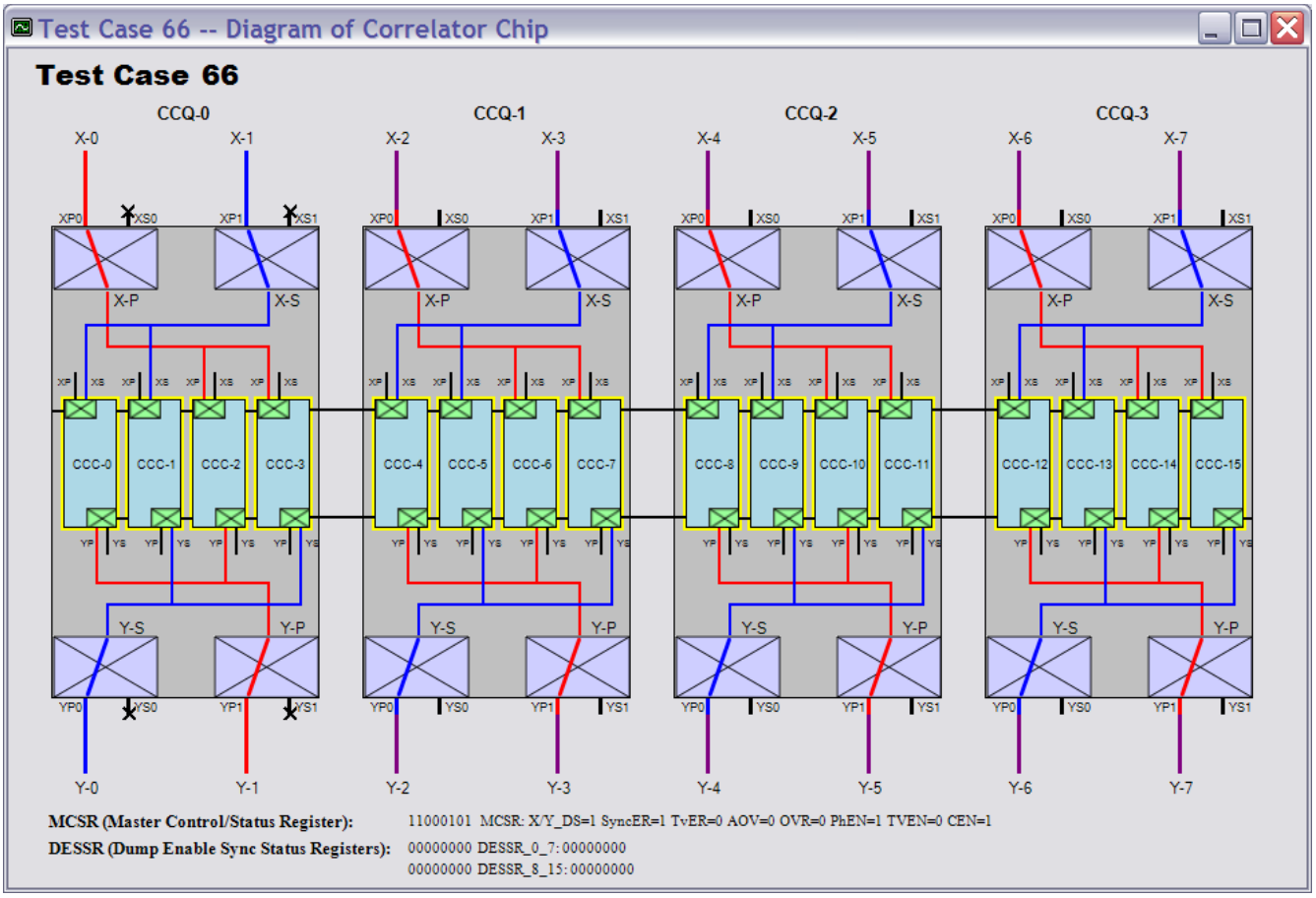
test_duration_mem[tc] = 2500;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// ***********************************************************************************************************
```

4.67 Test case 66




```
// ********************************************************************************************************************
// File name : tc66_aug18-05_v1.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V1
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc66_aug18-05_v1.v     Aug 18, 2005    1                creation
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                                 BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 66;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 66:
# ver: v1,05aug18
# Synopsis:   NO CONCAT., ALL POLN, 8 INPUTS, SELECTED STREAMS RESET
# Description:
#              Purpose of Test:      detection of missing phase or data streams
#              Concatenation:        1(0)1, 1(1)0, 0(2)1, 0(3)0, 3(4)3, 3(5)2, 2(6)3,
#                                 2(7)2, 5(8)5, 5(9)4, 4(10)5, 4(11)4, 7(12)7, 7(13)6,
#                                 6(14)7, 6(15)6
#              Num of Dumps-input:   1-0, 1-1, 1-2, 1-3, 1-4, 1-5, 1-6, 1-7
#              Dump Control:         X
#              Num samples/dump-inp: 5000-0, 5000-1, 5000-2, 5000-3, 5000-4, 5000-5,
#                                 5000-6, 5000-7
#              Distinct samp/dump:  5000-0, 5000-1, 5000-2, 5000-3, 5000-4, 5000-5,
#                                 5000-6, 5000-7
#              Active SE clocks:     0, 1, 2, 3, 4, 5, 6 , 7
#              Modulo se_clk gen:    none
#              Compare result with:  expected error reporting
#              Additional details:
#
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ---------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ---------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______1______0______0______1______1______0;
cccscr_2_3_mem[tc] = 8'b___1______0______0______1______0______1______0______1;
cccscr_4_5_mem[tc] = 8'b___1______0______1______0______0______1______1______0;
cccscr_6_7_mem[tc] = 8'b___1______0______0______1______0______1______0______1;
cccscr_8_9_mem[tc] = 8'b___1______0______1______0______0______1______1______0;
cccscr_10_11_mem[tc] = 8'b_1______0______0______1______0______1______0______1;
cccscr_12_13_mem[tc] = 8'b_1______0______1______0______0______1______1______0;
cccscr_14_15_mem[tc] = 8'b_1______0______0______1______0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ---------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ---------------------------------------------------------------------------
//                           B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b______0________0_________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b______0________0_________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//         modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//             x_dead_mem[tc] -- Bits that are reset (0) in this 89-bit register for each testcase will cause
//             the corresponding X-inputs to the chip to be forced low.  If a bit is set (1),
//             the corresponding X-input to the chip is unaffected.  Default: all bits set (1).
//             Bit assignments are as follows:
//             =============                ================
//             Bit locations                Signals affected
//             =============                ================
//             [7:0]                        DUMP_EN[7:0]
//             [15:8]                       VALID[7:0]
//             [23:16]                      SE_CLK[7:0]
//             [55:24]                      PHASE-[0...7][3:0] (e.g. [27:24] = PHASE-0[3:0]
//             [87:56]                      SDATA-[0...7][3:0] (e.g. [59:56] = SDATA-1[3:0]
//             [88]                         TIMESTAMP.
//             y_dead_mem[tc] -- Same as x_dead_mem, only for Y inputs to the chip.

y_dead_mem[tc]= 'h1f0f0f0f0f0f0f0f0aaffff;
x_dead_mem[tc]= 'h10f0f0f0f0f0f0f0f55ffff;

// x_force_dvalid_on_mem[] and y_force_dvalid_on_mem[] memories. If they are not set, then
// they have no effect, if they are set, then dvalid is force on during and after schid_frame when the
// ccorrelator chip expects dvalid to be low. Each dvalid bit for x and y is independently settable.
// E.g. x_force_dvalid_on_mem[tc]=8'b1010_1010; forces dvalids 7, 5, 3, and 1 to be '1' when they are
// supposed to be '0', but leaves dvalids 6, 4, 2, and 0 in the normal condition.

y_force_dvalid_on_mem[tc]=8'b0101_0101;
x_force_dvalid_on_mem[tc]=8'b1010_1010;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
```

```
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 2500;
dump_period_mem1[tc] = 2500;
dump_period_mem2[tc] = 2500;
dump_period_mem3[tc] = 2500;
dump_period_mem4[tc] = 2500;
dump_period_mem5[tc] = 2500;
dump_period_mem6[tc] = 2500;
dump_period_mem7[tc] = 2500;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 3;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 6;

//-----------------------------------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 2500;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

$display("---Test Case %d initialized",tc);

// *************************************************************************************************************
```

4.68 Test case 68

This region is intentionally left blank.

```
// ********************************************************************************************************
// File name : tc68_nov04-05_v1.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V1
// Company   : National Research Council of Canada
//                  P.O. Box 248
//                  Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                  phone: 250-490-4346
//                  email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************
// ********************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc68_nov04-05_v1.v     Nov 04, 2005    1                creation
//
//
// ********************************************************************************************************
//
//
```

```
// ********************************************************************************************************************
// ********************************************************************************************************************
//                                                    BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 68;
test_description[tc] =
"# ---------------------------------------------------------------------------
# TEST 68:
# ver: v1,05nov04
# Synopsis:   MIX OF DEFINED AND UNDEFINED CELL INPUTS, DUMP ON X
# Description:
#              Purpose of Test:      configure with stream 6, 7 on quad 0
#              Concatenation:        6(0)0, 6(1)1, 7(2)0, 7(3)1
#              Num of Dumps-input:   1-0, 1-1 , 1-2, 1-3
#              Dump Control:         X
#              Num samples/dump-inp: 1000-0, 1000-1, 1000-2, 1000-3
#              Distinct samp/dump:   500-0, 500-3, 500-4, 500-5
#              Active SE clocks:     0, 1, 6, 7
#              Modulo se_clk gen:    mod 2
#              Compare result with:  expected spectral signature
#              Additional details:
#
# ---------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b_____1_______0_______0____0____0_____1_____0_____1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCCSCR Input Selection:
//                       00 = adjacent CCC
//                       01 = primary CCC input
//                       10 = secondary CCC input
//                       11 = undefined: no lag frames produced
//
//                          CCC0,1 switch config register: (same for CCC2...CCC15)
//                         |--------- CCC1 ----------|---------- CCC0 -----------|
//                         ------------------------------------------------------
//                         Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                         ------------------------------------------------------
//                          B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___0_____1_____0_____1_____1_____0_____0_____1;
cccscr_2_3_mem[tc] = 8'b___0_____1_____1_____0_____1_____0_____1_____0;
cccscr_4_5_mem[tc] = 8'b___1_____1_____1_____1_____1_____1_____1_____1;
cccscr_6_7_mem[tc] = 8'b___1_____1_____1_____1_____1_____1_____1_____1;
cccscr_8_9_mem[tc] = 8'b___0_____1_____0_____1_____1_____0_____0_____1;
cccscr_10_11_mem[tc] = 8'b_0_____1_____1_____0_____1_____0_____1_____0;
cccscr_12_13_mem[tc] = 8'b_0_____1_____0_____1_____1_____0_____0_____1;
cccscr_14_15_mem[tc] = 8'b_0_____1_____1_____0_____1_____0_____1_____0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                        CCQR Input Selection:
//                             0 = primary
//                             1 = secondary
//
//                        CCQ0,1 switch config register
//                        ------------------------------------------------------------------------------
//                        Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                        ------------------------------------------------------------------------------
//                            B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________1_________1;
ccqr_2_3_mem[tc] = 8'b_______1________1__________0________0_________1_________1_________1_________1;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______0_______0_______0_______0_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//          modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__1_______1_______0_______0_______0_______0_______1_______1;
y_se_clkmod_mem[tc] = 8'b__1_______1_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______0_______0_______0_______0_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______0_______0_______0_______0_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, X controls dumping.  Only one input line is used.
// dump_en is transmitted LSB first.
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {70'bx,2'b0};
x_dumpen3_mem[tc] = {70'bx,2'b0};
x_dumpen4_mem[tc] = {70'bx,2'b0};
x_dumpen5_mem[tc] = {70'bx,2'b0};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 500;
dump_period_mem1[tc] = 500;
dump_period_mem2[tc] = 500;
dump_period_mem3[tc] = 500;
dump_period_mem4[tc] = 500;
dump_period_mem5[tc] = 500;
dump_period_mem6[tc] = 500;
dump_period_mem7[tc] = 500;
```

```
//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = SCHID_PERIOD;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------

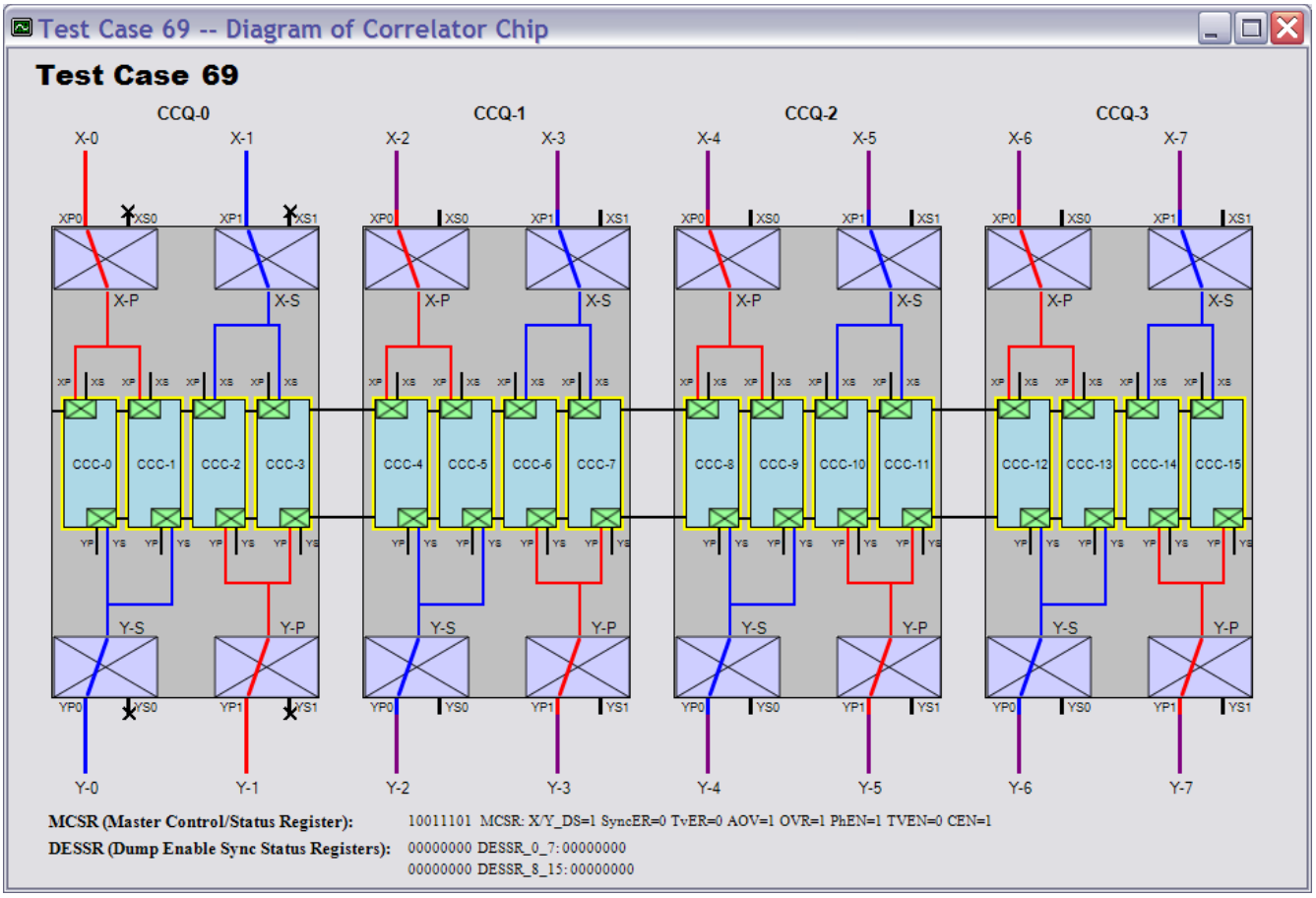
// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 500;

//-----------------------------------------------------------------------------------------------------------
```

4.69 Test case 69




```
// ********************************************************************************************************************
// File name : tc69_dec09-05_v1.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V1
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                phone: 250-490-4346
//                email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc69_dec09-05_v1.v     Dec  9, 2005    1                creation
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ********************************************************************************************************************
//                                                          BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ********************************************************************************************************************
// ********************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 69;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 69:
# ver: v1,05dec09
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, MULTIPLE DUMP RATES ON X
# Description:
#              Purpose of Test:      check sync. error reporting
#              Concatenation:        0(0)0,0(1)0,1(2)1,1(3)1,2(4)2,2(5)2,3(6)3,3(7)3,
#                                 4(8)4,4(9)4,5(10)5,5(11)5,6(12)6,6(13)6, 7(14)7,7(15)7
#              Num of Dumps-input:   128-0, 64-1, 32-2, 16-3, 8-4, 4-5, 2-6, 1-7
#              Dump Control:         X
#              Num samples/dump-inp: 1000-0, 2000-1, 4000-2, 8000-3, 16000-4, 32000-5,
#                                 64000-6, 128000-7
#              Distinct samp/dump:   1000-0, 2000-1, 4000-2, 8000-3, 16000-4, 32000-5,
#                                 64000-6, 128000-7
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with: behavioural model with test vector set DS4
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______1________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______1______0______0______1______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______1______0______0______1______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______1______0______0______1______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______1______1______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______1______0______0______1______1______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.

x_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
x_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

// The Y timestamp is ignored because X controls dumping.
ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 500;
dump_period_mem1[tc] = 1000;
dump_period_mem2[tc] = 2000;
dump_period_mem3[tc] = 4000;
dump_period_mem4[tc] = 8000;
dump_period_mem5[tc] = 16000;
dump_period_mem6[tc] = 32000;
```

```
dump_period_mem7[tc] = 64000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

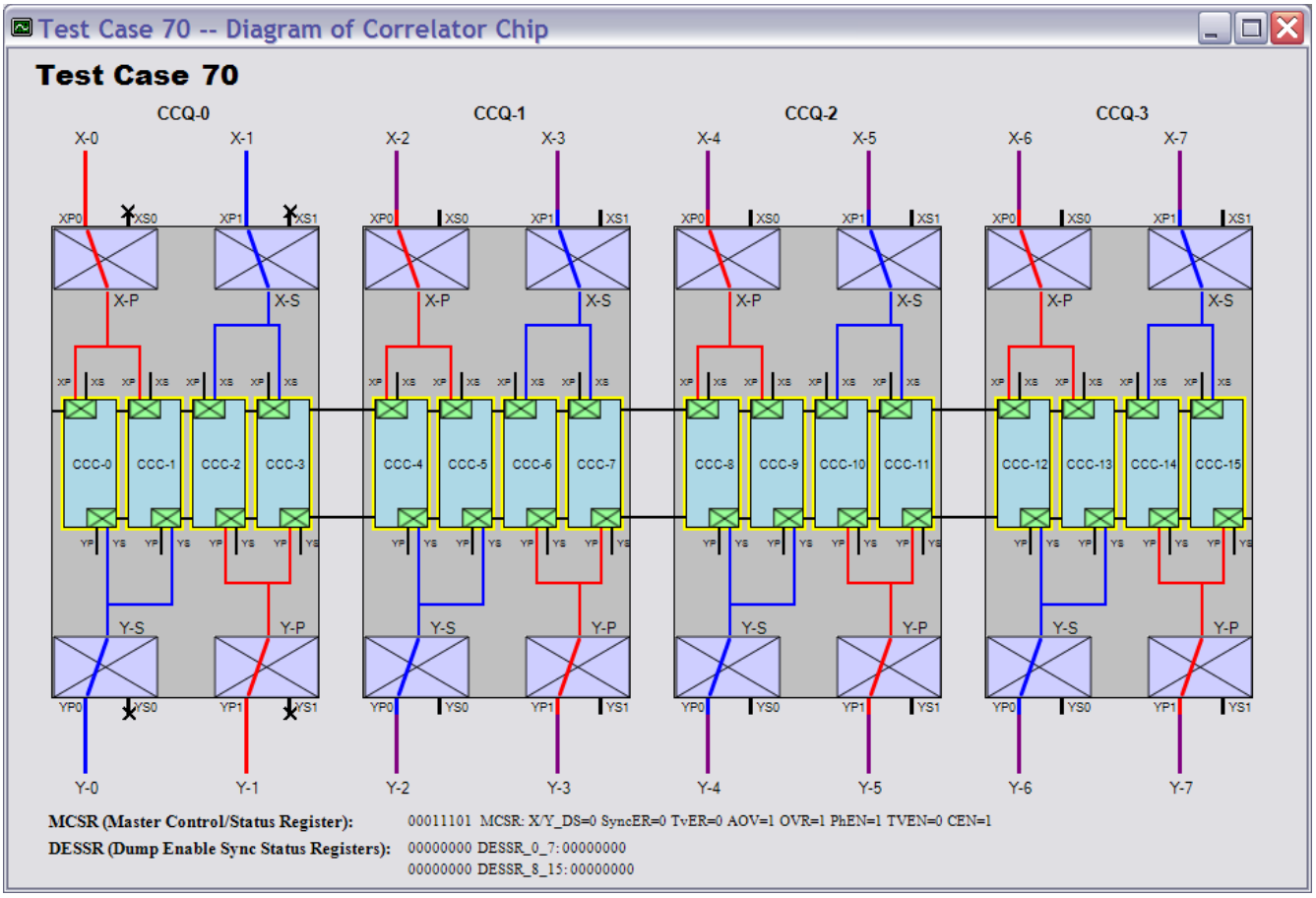
//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 64000;
```

4.70 Test case 70




```
// ********************************************************************************************************************
// File name : tc70_dec09-05_v1.v
// Project   : Correlator chip (25082) CORRCHIP_25082_TB_LIB
// Version   : V1
// Company   : National Research Council of Canada
//                 P.O. Box 248
//                 Penticton, BC, V2A 6K3
// Author    :  Brent Carlson
//                phone: 250-490-4346
//                email: Brent.Carlson@nrc.ca
//
// Contributors :  Rick Smegal
//
//
// Design description:
// This file contains a single testcase for the top-level correlator chip testbench.
// This test is based on the original work by
// Brent Carlson and Frances Lau.
//
//
// ********************************************************************************************************************
// ************************    REVISION HISTORY    ********************************************************************
// ********************************************************************************************************************
// FILE NAME              DATE            VERSION          COMMENTS
// tc70_dec09-05_v1.v     Dec  9, 2005    1                creation
//
//
//
// ********************************************************************************************************************
//
//
// ********************************************************************************************************************
```

```
// ***************************************************************************************************************
//                                                    BEGINNING OF TEST CASE GENERATION/INITIALIZATION
// ***************************************************************************************************************
// ***************************************************************************************************************
//

// Notation for correlator cell concatenation : inpX(begin_cell:end_cell)inpY
// Cells not listed are disabled by an undefined setting of CCC data switches;
// a hyphen indicates inputs with 'no-data';
// inputs not shown have data valids reset (0) unless otherwise indicated.

tc = 70;
test_description[tc] =
"# ------------------------------------------------------------------------
# TEST 70:
# ver: v1,05dec09
# Synopsis:   ALL ACTIVE INP, CONCAT IN CELL PAIRS, MULTIPLE DUMP RATES ON Y
# Description:
#              Purpose of Test:      check sync. error reporting
#              Concatenation:        0(0)0,0(1)0,1(2)1,1(3)1,2(4)2,2(5)2,3(6)3,3(7)3,
#                                 4(8)4,4(9)4,5(10)5,5(11)5,6(12)6,6(13)6, 7(14)7,7(15)7
#              Num of Dumps-input:   128-0, 64-1, 32-2, 16-3, 8-4, 4-5, 2-6, 1-7
#              Dump Control:         X
#              Num samples/dump-inp: 1000-0, 2000-1, 4000-2, 8000-3, 16000-4, 32000-5,
#                                 64000-6, 128000-7
#              Distinct samp/dump:   1000-0, 2000-1, 4000-2, 8000-3, 16000-4, 32000-5,
#                                 64000-6, 128000-7
#              Active SE clocks:     all
#              Modulo se_clk gen:    none
#              Compare result with: behavioural model with test vector set DS4
#              Additional details:
#
# ------------------------------------------------------------------------";

//-------------------------------------------------------------------------------------------------------------------

//                    MCSR register:
//                    ---------------------------------------------------
//                    X_Y_DS | SyncER | TvER | AOV | OVR | PhEN | TVEN | CEN
//                    ---------------------------------------------------
//                      B7       B6      B5    B4    B3     B2     B1     B0
mcsr_mem[tc] = 8'b______0________0_______0_____0_____0______1______0______1;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                      CCCSCR Input Selection:
//                      00 = adjacent CCC
//                      01 = primary CCC input
//                      10 = secondary CCC input
//                      11 = undefined: no lag frames produced
//
//                         CCC0,1 switch config register: (same for CCC2...CCC15)
//                        |--------- CCC1 ----------|---------- CCC0 -----------|
//                        ---------------------------------------------------------
//                        Y-S1 | Y-S0 | X-S1 | X-S0 | Y-S1 | Y-S0 | X-S1 | X-S0
//                        ---------------------------------------------------------
//                         B7     B6     B5     B4     B3     B2     B1     B0
cccscr_0_1_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_2_3_mem[tc] = 8'b___0______1______1______0______0______1______1______0;
cccscr_4_5_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_6_7_mem[tc] = 8'b___0______1______1______0______0______1______1______0;
cccscr_8_9_mem[tc] = 8'b___1______0______0______1______1______0______0______1;
cccscr_10_11_mem[tc] = 8'b_0______1______1______0______0______1______1______0;
cccscr_12_13_mem[tc] = 8'b_1______0______0______1______1______0______0______1;
cccscr_14_15_mem[tc] = 8'b_0______1______1______0______0______1______1______0;

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//                       CCQR Input Selection:
//                            0 = primary
//                            1 = secondary
//
//                       CCQ0,1 switch config register
//                       ------------------------------------------------------------------------------
//                       Q1-YSw1 | Q1-YSw0 | Q1-XSw1 | Q1-XSw0 | Q0-YSw1 | Q0-YSw0 | Q0-XSw1 | Q0-XSw0
//                       ------------------------------------------------------------------------------
//                          B7       B6         B5       B4        B3        B2        B1        B0
ccqr_0_1_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;
ccqr_2_3_mem[tc] = 8'b_______0________0__________0________0_________0_________0_________0_________0;

//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------------------

//                      shift enable clock(X,Y-SE_CLK) generation for each test, for each input.
//                      On (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_se_clk_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

//             Modulo shift enable clock (X,Y-SE_CLK) generation for each test, for each input.
//             On (1) or off (0).
//             Is only looked at if the corresponding bit in x/y_se_clk_mem for the same test is set (1).
//             If modulo generation is on, then the se_clk is on when: !(samplenum % [input_number+2]).
//             However, this can be overridden by set_modulo_mode_mem below.
//             E.g. se_clk_0 would be on every other clock cycle, se_clk_1 would be on every third clock cycle etc.
//             This could result in the detection of input synchronization errors since a given se_clk may not
//             be 1 when schid_frame is asserted.
//
//            modulo X,Y-SE_CLK generation is on (1) or off (0).
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;
y_se_clkmod_mem[tc] = 8'b__0_______0_______0_______0_______0_______0_______0_______0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// If modulo generation is on and this bit is reset (0), then modulo 2 clock
// generation is used no matter what the input number.  If this bit is set (1),
// then modulo (input_number+2) generation is used.
//
set_modulo_mode_mem[tc] = 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// For each test, the contents of this memory is ANDed with the data valid lines
// for inputs X and Y.
//
//                      Input line number
//                      ------------------------------------------------------------------
//                         7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
//                      ------------------------------------------------------------------
x_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;
y_dvalid_mem[tc] = 8'b_____1_______1_______1_______1_______1_______1_______1_______1;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// set the test mode:
// 00 -- normal chip operation.
// 01 -- scan-insertion for mfg test.  Not normally used for sim.
// 10 -- split accumulator for faster accum exercise. Accumulators, including the overflow bits can
//             be fully exercised in 660 128 MHz clock cycles (dump period of ~5.17 usec).
// 11 -- hi-speed internal test for mfg test.  Not normally used for sim.
//       ***Default if not specified is normal chip operation (00) ****
//
testmode_mem[tc]= 2'b00;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------

// VERIFICATION ONLY
// If set(1), then the test ends at the end of test_duration_mem
// and no mcb_regs files are produced.  lag_frames files may
// be produced, as the chip will only continue to dump frames as
// long as the test has not exceeded the test_duration.
// A "verifyNN.txt" file is produced, which is one or more ASCII dumps
// of all important internal nodes to be used for comparison with gate-level
// sim results.  In this mode, all lag shift regs are intialized with
// random data so that it is not necessary to wait for data to shift
// down the shift regs to exercise the accumulators well.  In this mode
// no useful lag data for validation is likely produced.
//
verification_only_mem[tc]= 0;

//-------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------
```

```
// This block defines temporary registers to hold data that
// is to be transmitted serially on one or more X,Y-DUMP_EN lines and assigns this data
// to a register corresponding to an active X,Y-DUMP_EN line.
// Different data may be assigned to each line, if required.
// The length of the serial sequence for dumpen and timestamp is defined by the
// parameter dts_len, and currently dts_len = 72.
// So, there are 72 bits total...here

fill_sync = 2'b01;                          // 2 bits

// Dump control bits:
//             000 = First dump of data into LTA. Just save data in LTA bin.
//             001 = Add to existing LTA data and save in LTA.
//             010 = Last dump: add data to LTA, and flag data as ready for readout.
//             011 = Speed dump: bypass LTA directly to output. Data bias removed.
//             100 = Correlator chip dumps, discards data, clears registers.
//             1X1 = Reserved.
//             11X = Reserved.

dump_command = 3'b010;                      // 3 bits: last dump

// Phase bin number that the data gets dumped/accumulated into.
//             Note: PB15 is the bank number (0 or 1)
//                   With speed dumping, all 16 PB bits can be used for a total of 65,536 bins.

phase_bin = 55666;                          // 16 bits
recirc_blk = 233;                           // 8 bits
fill3 = 3'b111;                              // 3 bits

// Holdoff held high to disable correlation until lag shift registers
// fill up with data.
holdoff = 40'b0000000000000000000000000000000000001111;    // 40 bits

// Either X or Y controls dumping, as defined by X/Y_DS (X/Y dump select) in the MCSR.
// Here, Y controls dumping. All lines are used.
// dump_en is transmitted LSB first.

y_dumpen1_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen3_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen5_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen7_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen0_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen2_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen4_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};
y_dumpen6_mem[tc] = {holdoff,fill3,recirc_blk,phase_bin,dump_command,fill_sync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// This block defines temporary registers to hold the TIMESTAMP data and assigns
// this data to a register corresponding to the X,Y-TIMESTAMP data line.

// Timestamps
// The X timestamp is ignored because Y controls dumping.
ts_fillsync = 2'b01;
ts_0 = 222333444;
ts_1 = 555666777;
ts_endfill = 0;
x_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

ts_fillsync = 2'b01;
ts_0 = 'hffffffff;
ts_1 = 'hffffffff;
ts_endfill = 0;
y_timestamp_mem[tc] = {ts_endfill,ts_1,ts_0,ts_fillsync};

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

// These registers contains the dump period in 128 MHz clock cycles. All tests
// start with a dump and discard immediately after reset so that everything gets
// into a known state.
// The facility to set the dump period independently for each DUMP_EN line
// was added Feb. 5, 2004.
// NOTE: IT IS UP TO THE USER TO ENSURE THAT THERE
//    ARE NO "COLLISIONS"...THUS, ALL OF THE DUMP_PERIOD_MEM(0...7) MUST BE
//    HARMONICALLY RELATED.

// DEFAULT for tests in 128 MHz clock cycles is DUMP_PERIOD=5000

dump_period_mem0[tc] = 500;
dump_period_mem1[tc] = 1000;
dump_period_mem2[tc] = 2000;
dump_period_mem3[tc] = 4000;
dump_period_mem4[tc] = 8000;
dump_period_mem5[tc] = 16000;
dump_period_mem6[tc] = 32000;
```

```
dump_period_mem7[tc] = 64000;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often, in 256 MHz clock cycles, the X,Y-SCHID_FRAME pulses are generated.
// **MUST** BE 512 * 2^n
// DEFAULT for tests in 256 MHz clock cycles SCHID_PERIOD=512

schid_period_mem[tc] = 512;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define memory that allows us to generate frame abort tests.  If the mem is
// set (1), then every other dump for each CCC is aborted partway through
// transmission.

frame_abort_mem[tc] = 0;  // set (1) to abort CCC first dump frames

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Generate input sync errs on the X and Y data at 256 MHz.  If bit in this memory is set (1) then
// error generation occurs.  If reset (0) then no error generation.

sync_err_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how often errors are generated for each test.
// If TVEN=1, then this is how often in samples.  If TVEN=0, then
// this is how often in terms of the input synchronization pattern (i.e. every
// sync_err_mod_mem sync patterns, the error is generated).

sync_err_mod_mem[tc] = 0;


//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y dump_sync and dump_en is later than the X dump_sync and dump_en
// in terms of 128 MHz clock cycles.

y_dump_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Set how much the Y schid_frame is offset from the X schid_frame
// in terms of 256 MHz clock cycles.

y_schid_offset_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y station IDs for each test.
// define embedded identifiers...make them something easily recognized.

x_station_id_mem[tc] = 200;
y_station_id_mem[tc] = 100;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// Define X and Y sub-band IDs BASES.  The actual sub-band ID is this plus the
// input number (0...7)

x_sbid_mem[tc] = 0;
y_sbid_mem[tc] = 0;

//-----------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------

// The number of 128 MHz clock cycles over which the test runs; this time includes only dumps following
// an initial dump and clear. The first dump (dump-discard) occurs about 5 usec after the start of the
// test case and is included as part of the intialization process.  The intial dump-discard occurs
// about 2.9 usec after the beginning of the testvectors (or, 740 samples at full speed).
//
// NOTE: Enough time must be allowed to fill the longest concatenated chain with valid samples before
// generally valid data is accumulated.

// DEFAULT test duration for tests in 128 MHz clock cycles is TEST_DURATION.

test_duration_mem[tc] = 64000;
```

Bibliography

TVP: Carlson, Brent, EVLA Correlator Chip, Test and Verification Plan, A25082N0001, Rev. DRAFT, National Research Council of Canada, August 26, 2004

Mem001: Carlson, Brent, A proposed WIDAR Correlator for the Expansion Very Large Array Project: Discussion of Capabilities, Implementation, and Signal Processing , NRC-EVLA Memo, #001, Rev. 0, National Research Council of Canada, May 2000

KroFHV: Kropf, Thomas, Introduction to Formal Hardware Verification, Springer-Verlag, 1999

Pix01: Pixley, Carl, Formal Verification of Commercial Integrated Circuits, IEEE Design and Test of Computers, v18 n4, 2001

RFS: Carlson, Brent, EVLA Correlator Chip, Requirements and Functional Specification, A25082N0000, Rev. 2.3, National Research Council of Canada, October 29, 2004

LauTAS: Lau, Frances, EVLA Correlator Chip Test Analysis Software, Requirements and Functional Specification, A25082N0011, Rev. 1.2, National Research Council of Canada, July 2003

OctWWW: Octave Home Page, http://www.octave.org/

LauTS: Lau, Frances, EVLA Correlator Chip Testing Software, Requirements and Functional Specification, A25082N0010, Rev. 1.2, National Research Council of Canada, May 2003