TEST AND VERIFICATION REPORT

EVLA Correlator Chip Functional Test Results

TVR document: **A25082N0003**

Revision: 1.0

Rick Smegal

January 30, 2006

Aardvark Resources

Document No. EVLAC-R06D01 Rev. 0.2

Victoria, B.C., Canada

Table of Contents

1 Preamble....8
1.1 Revision history....8
1.2 Document organization....9
1.3 Document objectives and scope....9
1.4 Limitations....9
2 Post simulation analysis....10
2.1 Introduction....10
2.2 Nomenclature and symbol definition....11
2.3 CMAM and correlation....12
2.3.1 organization....12
2.3.2 test case 00....13
2.3.3 test case 32, 33....15
2.4 accumulator overflow (test case 01)....26
2.5 normal dump and readout....28
2.5.1 organization....28
2.5.2 test case 02....31
2.5.3 test case 03....37
2.5.4 test case 04....53
2.5.5 test case 05....60
2.5.6 test case 06....63
2.5.7 test cases 08, 09....81
2.5.8 test case 10....112
2.5.9 test case 11....118
2.5.10 test case 12....125
2.5.11 test case 14....130
2.5.12 test case 25....146
2.5.13 test case 28....163
2.5.14 test case 30....175
2.5.15 test case 64....193
2.6 shift enable clock (X|Y-SE_CLK)....202
2.6.1 organization....202
2.6.2 test case 21....202
2.6.3 test case 22....205
2.7 input configuration switches....208
2.7.1 organization....208
2.7.2 test case 15....209
2.7.3 test case 16....213
2.7.4 test case 17....216
2.7.5 test case 18....221
2.7.6 test case 19....226
2.7.7 test case 20....241
2.7.8 supplemental experiments....258
2.7.9 test case 68....265
2.8 extreme dump rate operation....270
2.8.1 organization....270
2.8.2 test case 07....270
2.9 overrun detection and recovery....276
2.9.1 organization....276
2.9.2 test case 24....277
2.9.3 test case 29....280

2.9.4 test case 31....283
2.9.5 test case 69....286
2.9.6 test case 70....297
2.10 on-line detection and reporting of errors....303
2.10.1 organization....303
2.10.2 test case 03....303
2.10.3 test case 04....303
2.10.4 test case 05....303
2.10.5 test case 06....304
2.10.6 test case 11....304
2.10.7 test case 12....304
2.10.8 test case 13....305
2.10.9 test case 17....307
2.10.10 test case 18....307
2.10.11 test case 26....307
2.10.12 test case 27....309
2.10.13 test case 35....313
2.10.14 test case 36....316
2.10.15 test case 37....318
2.10.16 test case 46....319
2.10.17 test case 47....319
2.10.18 test case 50....320
2.10.19 test case 51....325
2.10.20 test case 52....330
2.10.21 test case 53....331
2.10.22 test case 65....332
2.10.23 test case 66....333
2.11 X-Y dump enable offset detection....334
2.11.1 organization....334
2.11.2 test case 03....334
2.11.3 test case 05....334
2.11.4 test case 11....334
2.11.5 test case 37....334
2.11.6 test case 48....335
2.11.7 test case 49....339
2.12 off-line detection and reporting of synchronization errors....343
2.12.1 organization....343
2.12.2 test case 38....343
2.12.3 test case 39....345
2.12.4 test case 40....346
2.12.5 MCSR status flags....347
2.13 correlation hold off....347
2.13.1 test case 42....347
2.14 frame abort....350
2.14.1 test case 43....350
2.15 dump command capture....352
2.15.1 organization....352
2.15.2 test case 41....353
2.15.3 test case 42....355
2.15.4 test case 46....356
2.15.5 test case 47....358
2.16 timestamp capture....360

2.16.1 organization....360
2.16.2 test case 17....360
2.16.3 test case 18....360
2.16.4 test case 19....361
2.16.5 test case 20....361
2.17 data stream ID capture....361
2.17.1 organization....361
2.17.2 test case 17....362
2.17.3 test case 18....362
2.17.4 test case 19....362
2.17.5 test case 20....363
2.18 enable phase rotation....363
2.18.1 test case 44....363
2.19 enable correlation....366
2.19.1 test case 45....366
2.20 autocorrelation....368
2.20.1 organization....368
2.20.2 test case 56....369
2.20.3 test case 57....371
2.20.4 test case 58....373
2.20.5 test case 59....375
2.20.6 test case 60....377
2.20.7 test case 61....379
2.20.8 test case 62....381
2.20.9 test case 63....383
2.21 missing test cases....385
3 Post simulation scripts....386
3.1 Introduction....386
3.1.1 organization....386
3.1.2 function summary....386
3.2 register consistency checking....389
3.3 CMAM and correlation....391
3.3.1 tc00_postsim....391
3.3.2 tc32_33_postsim....392
3.3.3 make_lag_tables_tc32_33....396
3.3.4 make_fd_tables_tc32_33....397
3.3.5 validate_bit_exact....398
3.4 accumulator overflow....398
3.4.1 tc01_postsim....398
3.5 normal dump and readout....400
3.5.1 tc02_postsim....400
3.5.2 tc02_chain_similarity_check....402
3.5.3 tc03_postsim....402
3.5.4 tc04_postsim....404
3.5.5 tc04_inspect_status_captured....406
3.5.6 tc05_postsim....406
3.5.7 tc05_chain_similarity_check....408
3.5.8 tc06_postsim....408
3.5.9 tc06_inspect_status_captured....410
3.5.10 tc08_postsim....411
3.5.11 tc09_postsim....414
3.5.12 tc08_09_similarity_check....416

3.5.13 tc10_postsim....416
3.5.14 tc10_postsim_validation_1....418
3.5.15 tc11_postsim....418
3.5.16 tc11_inspect_status_captured....420
3.5.17 tc12_postsim....421
3.5.18 tc14_postsim....423
3.5.19 tc25_postsim....425
3.5.20 tc25_chain_similarity_check....427
3.5.21 tc30_postsim....427
3.5.22 tc30_bit_exact....429
3.5.23 tc64_postsim....430
3.6 shift enable clock (X|Y-SE_CLK)....432
3.6.1 tc21_postsim....432
3.6.2 tc22_postsim....434
3.7 input configuration switches....436
3.7.1 tc15_postsim....436
3.7.2 tc15_similarity_check....438
3.7.3 tc16_postsim....438
3.7.4 tc16_similarity_check....440
3.7.5 tc17_postsim....440
3.7.6 tc17_make_fb_stat_tables....442
3.7.7 tc18_postsim....442
3.7.8 tc18_make_fb_stat_tables....444
3.7.9 tc19_postsim....444
3.7.10 tc19_make_fb_stat_tables....446
3.7.11 tc20_postsim....446
3.7.12 tc20_make_fb_stat_tables....448
3.7.13 tc68_postsim....449
3.7.14 tc68_similarity_check....450
3.8 extreme dump rate operation....451
3.8.1 tc07_postsim....451
3.8.2 tc07_dump_overrun_tables....453
3.8.3 tc07_cell_bias_table....453
3.9 overrun detection and recovery....454
3.9.1 tc24_postsim....454
3.9.2 tc24_chain_diff_tables....456
3.9.3 tc24_dump_overrun_tables....456
3.9.4 tc29_postsim....457
3.9.5 tc29_dump_overrun_tables....459
3.9.6 tc29 chain_similarity_check....459
3.9.7 tc31_postsim....461
3.9.8 tc31_dump_overrun_tables....463
3.10 on-line detection and reporting of errors....464
3.10.1 tc13_postsim 464
3.10.2 tc13_list_dumped_chains....465
3.10.3 tc13_chain_similarity_check....465
3.10.4 tc27_postsim....466
3.10.5 tc27_list_dumped_chains....467
3.10.6 tc27_find_NaN....467
3.10.7 tc35_postsim....468
3.10.8 tc35_list_dumped_chains....469
3.10.9 tc35_find_NaN....469

3.10.10 tc36_postsim....470
3.10.11 tc50_postsim....471
3.10.12 tc50_check_captured_data....473
3.10.13 tc50_check_status_bits....473
3.10.14 tc51_postsim....473
3.10.15 tc51_check_captured_data....475
3.10.16 tc51_check_status_bits....475
3.10.17 tc69_postsim....475
3.10.18 tc69_make_dump_ovr_list....478
3.10.19 tc69_dump_overrun_tables....478
3.10.20 tc69_chain_similarity_check....479
3.10.21 tc69_ACC_OV_scan....479
3.10.22 tc70_postsim....479
3.10.23 tc70_make_dump_ovr_list....482
3.10.24 tc70_dump_overrun_tables....482
3.10.25 tc70_chain_similarity_check....483
3.10.26 tc70_ACC_OV_scan....483
3.10.27 tc69_70_chain_compare....484
3.11 X-Y dump enable offset detection....484
3.11.1 tc48_postsim....484
3.11.2 tc48_captured_table....485
3.11.3 tc48_data_ID_capture....486
3.11.4 tc49_postsim....486
3.11.5 tc49_captured_table....488
3.11.6 tc49_data_ID_capture....488
3.12 off-line detection and reporting of errors....489
3.13 MCSR status flags....489
3.14 correlation hold off....489
3.14.1 tc42_postsim....489
3.15 frame abort....491
3.15.1 tc43_postsim....491
3.15.2 tc43_print_timestamps....492
3.16 dump command capture....493
3.16.1 tc41_postsim....493
3.16.2 tc41_dump_capture....494
3.16.3 tc42_dump_capture....494
3.16.4 tc46_postsim....495
3.16.5 tc46_dump_capture....496
3.16.6 tc47_postsim....497
3.16.7 tc47_dump_capture....498
3.17 timestamp capture....499
3.17.1 tc17_timestamp_capture....499
3.17.2 tc18_timestamp_capture....499
3.17.3 tc19_timestamp_capture....499
3.17.4 tc20_timestamp_capture....499
3.18 data stream ID capture....500
3.18.1 tc17_data_ID_capture....500
3.18.2 tc18_data_ID_capture....500
3.18.3 tc19_data_ID_capture....500
3.18.4 tc20_data_ID_capture....500
3.18.5 tc41_data_ID_capture....500
3.18.6 tc42_data_ID_capture....501

3.18.7 tc46_data_ID_capture....501
3.18.8 tc47_data_ID_capture....501
3.19 enable phase rotation....502
3.19.1 tc44_postsim....502
3.19.2 tc44_similarity_check....503
3.20 enable correlation....504
3.21 autocorrelation....504
3.21.1 tc56_postsim....504
3.21.2 tc57_postsim....504
3.21.3 tc58_postsim....505
3.21.4 tc59_postsim....506
3.21.5 tc60_postsim....507
3.21.6 tc61_postsim....508
3.21.7 tc62_postsim....508
3.21.8 tc63_postsim....509
3.22 ancillary functions....511
3.22.1 function make_stat_tables....511
3.22.2 model_diff_chains....518
3.22.3 check_timestamp....519
3.22.4 write_lag_plots....520
3.22.5 write_fd_plots....521
3.22.6 function write_fmod_fd_plots....523
3.22.7 function list_dump_ovr....524
3.22.8 DS14_diff_phase....527
3.22.9 write_diff_phase_plots_tc32....527
3.23 function check_register_capture....528
3.24 postsim_lib function listing....541
3.24.1 get_lag_frames....541
3.24.2 make_dump_index....543
3.24.3 add_CCchain_dump....544
3.24.4 get_mcb_data....549
3.24.5 make_mcb_chains....556
3.24.6 get_bmodel_chains....559
3.24.7 get_bmodel_lags....560
3.24.8 lag2fd....560
3.24.9 chain2single....561
3.24.10 single2chain....561
3.24.11 scale_lag_data....562
3.24.12 which_DVCOUNT....563
3.24.13 list_versions....563
3.25 test vector editing scripts....564
3.25.1 DS10....564
3.25.2 DS11....564
3.25.3 DS14....564
3.25.4 DS15....565
3.25.5 DS16 565
3.25.6 get_CCtest_data....566
3.25.7 write_CCtest_data....566
3.26 behavioural model....567
3.26.1 corrsimCCtest.c....567
3.26.2 sigparms_CCtest.h....572

1 Preamble

1.1 Revision history

Revision	*Date*	*Notes*	*Author*
0.0	November 22, 2005	Created as draft **EVLA document type:** Test and Verification Plan; **EVLA document title:** "EVLA Correlator Chip Functional Test Results"; **EVLA document no.:** A25082N0003; Aardvark Resources doc. no. EVLAC-R06D01	R. Smegal
0.1	December 16, 2005	Change document type on title page to TVR; statement regarding RTL model lineage added to section 2.1; add differential phase plot to bit exact tests of section 2.3.3; update test case 07; add bit exact test of case 30; add script to generate DS16; test cases 65, 66, 68, 69, 70 added.	R. Smegal
0.2	January 30, 2006	EVLA project document revision designation changed from DRAFT to **Rev. 1.0**; fix outline levels; minor format fixes.	R. Smegal

1.2 Document organization

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

Chapter 2 contains some introductory material followed by output of the test bench, post simulation script and descriptive summary organized according to the operation categories developed in Chapter 3 of the test plan document [TVP-02].

Chapter 3 contains some introductory material followed by source listings for each top level (test case specific) post simulation script organized according to the chip operation categories of Chapter 2 above.

1.3 Document objectives and scope

This document presents the result of executing the validation programme for the EVLA Correlator ASIC (Application Specific Integrated Circuit) as described by the test case document, [TVP-02]. Unless otherwise stated, the individual test cases are described in detail within the test case document and only a description of the post simulation analysis procedure and subsequent result of its application to each test case is presented in this report. Hence, for convenience, the structure of the report will closely follow that of [TVP-02]. All output generated by the test bench is archived electronically with only the register output file reproduced within this report. Test case simulation results are represented by the output of post simulation analysis scripts as described in later chapters. The source codes of the post simulation scripts are included. Functions called by these scripts along with the source code for the behavioural model used in generating these results are included in chapter 3 for traceability. The objective of a post simulation script is to demonstrate that the corresponding test case has been successfully executed and to capture resultant information pertinent to the objective of that test case. The objective of this report is to present the output of the test bench and scripts (post simulation analysis) as clearly and succinctly as practical.

The overall chip test programme is described by the draft *EVLA Correlator Chip, Test and Verification Plan* document [TVP-01]. The *EVLA Correlator Chip*, RFS document, Revision 2.3 [RFS-00] and the behavioural model of the correlator provided by Brent Carlson [Mem001] comprise the correlator chip specification on which the validation is based.

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

1.4 Limitations

(1) This report is limited to documenting post simulation analysis of the validation tests described in [TVP-02] and does not in itself make a direct claim of chip RTL level compliance with the specification as described in [RFS-00] and [TVP-02].

(2) 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 of [TVP-02]) described in [RFS-00].

2 Post simulation analysis

2.1 Introduction

This chapter contains post simulation analysis of the test bench output obtained by executing the test cases described in the validation and verification plan document [TVP-02]. Post simulation analysis refers to the process of interpreting the raw output generated by the test bench and presenting the result in a manner in which realization of the test objectives can be readily identified. Material of this chapter is produced from execution of the test bench itself, execution of post simulation analysis scripts and a summarization. All test bench output is archived in raw form (ASCII files), although the format and quantity is too unwieldy to present in its entirety as text within this document; the only output presented from the test bench is that of the MCB register content file. The bulk of the remaining material was created from executing post simulation analysis scripts which apply a number of data sorting, reformatting, logical and arithmetic operations to the test bench output. For the most part, script output from each test case is imported into tables and plots for clarity and a synopsis provided to highlight expected features of the test result. The term *expected* , as used within the context of the *synopsis* refers to a specific result which is predicted within the test plan document [TVP-02]. An unexpected result that may be highlighted in the synopsis is not specifically addressed within the test plan but does not necessarily indicate failure of validation. However, an unexpected result may be noteworthy. A result that counters predicted behaviour or *contrary to expected* implies a failure of validation.

The structure of this chapter is similar to that of chapter 3 in the test plan document [TVP-02] insofar as the operations under test designation and their order of appearance. Each section (at the level of this introduction) represents a corresponding operation within the test plan document. However, further division (subsection) of some sections is required to group one or more test cases together in logical units that each satisfy a particular test objective; details are found under the *organization* heading of each section. Repetition may occur between chapters as some test cases represent more than one chip operation.

With exception of simple operations which can be adequately validated by inspection of the MCB register contents file, the analysis procedure for each operation is embodied in a *top level* post simulation script. Each script is specific to one or a small set of test cases. These top level scripts in turn call functions that are common to all or many scripts. Overall structure of the post simulation scripts is described in more detail within chapter 3 along with a source listing for each top level script. For convenience of locating the script for a specific test case, the structure of chapter 3 will follow that of this chapter.

Although timestamps are generated by top level post simulation scripts, there is no mechanism (such as a digital fingerprint) provided to associate output of a particular simulation run of a test case to its corresponding post simulation report or, for that matter, to associate a particular RTL model, test bench version and execution environment to a test bench output file. Hence, test traceability is limited. The only mechanism to currently audit results is a re-execution of all scripts and subsequent bit exact file difference inspection, that is to say, an independent duplication of the process. Otherwise, it is taken on faith that due diligence was exercised in maintaining the integrity of test data through out.

The RTL model code used by all test cases except bit exact tests involving test cases 32 and 33 was extracted from the archive: EVLA_Corr_chip_25082_Release_HDLFiles_Jul28-05.zip provided by NRC. Bit exact tests involving the test vector set DS14 and test cases 32 and 33 employed the RTL model extracted from archive: EVLA_Corr_chip_25082_Release_HDLFiles_Apr01-05.zip with updated file tvcompare80.v provided on May 2, 2005. The behavioural model code was unchanged throughout the programme.

2.2 Nomenclature and symbol definition

This report follows conventions and symbol definitions described in the like named section of the test case document [TVP-02]. They are reproduced below for convenience. Additional conventions, symbols and definitions that pertain to this document are added.

(1) **BOLD** font: names of quantities referred to in the [RFS-00] such as register names, control lines and data stream names (such as **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-00] The symbol, - , in place of *inpX* or *inpY* indicates a *no-data*[1] 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], **XY-SDATA[0:7][0:3]**; **XY-PHASE[0:7][0:3]**; single bit data valid line, **XY-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”.

(16) *cell number*: cells are labelled by an integer that increases sequentially beginning with 0, as defined in the RFS document [RFS-00].

(17) *chain number* (index): chains formed by concatenated cells are numbered sequentially with cell number, beginning with 1. In chip configurations containing both individual cells and chains, the single cell chains are numbered before chains of concatenated cells (both with increasing cell number).

(18) *chip dump number* (index): dumps are numbered sequentially beginning with 1. The timestamp is

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

likewise incremented by 1 in the case of a normal dump.

(19) *chain dump number* (index): dumps for a particular chain are numbered sequentially beginning with 1. In general, not all chains dump at a given timestamp, hence, a one to one correspondence between chip and chain dumps cannot be assumed.

(20) WARNINGS: issued by post simulation scripts and functions; indicate a condition contrary to expected, or result that should be investigated and reported upon.

(21) ADVISORIES: issued by post simulation scripts and functions; indicate an unusual or unexpected condition that could either be noted or simply ignored for a particular test.

2.3 CMAM and correlation

Test cases: 00, 32, 33

2.3.1 organization

This section is divided into subsections according to test case (or group of test cases). Each subsection contains (in approximately the following order)

(1) a post simulation synopsis;

(2) optional notes, if any, that detail outstanding features or conditions of the post simulation result;

(3) a *post simulation log* that documents script execution;

(4) tables of statistical parameters describing the arithmetic difference between the RTL simulation and behavioural model lag coefficients for test cases 32 and 33;

(5) selected plots of the RTL simulation result and the above model differences for test cases 32 and 33;

(6) content of the MCB register file that is provided by the test bench.

Additional notes on the above entities now follow.

Post simulation log: is the console output that results from running one or more post simulation scripts. The script *tc**nn**_postsim* (where ***nn*** is the test case number) is executed to read and perform processing common to all test cases of this section. Details of its operation are found in the relevant section of chapter 3. Output of any additional scripts required to validate specific operations is appended to the console log and described in the *synopsis* of each test case.

Cells, chains and dumps are indexed as described in 2.2 .

Register and captured data consistency checking is performed by a call to function *check_register_capture* (described with more detail in 2.5.1).

2.3.2 test case 00

Synopsis: No advisories or warnings regarding captured data or register content were logged. The expected value for data bias was found in the accumulator of all cells.

Notes:

(1) A warning would have been reported in the log for *check_register_capture* had the data bias differed between cells.

(2) The division by zero warning in the console log of Text 2.3.1 is a consequence of maintaining the data valid line low by a parameter setting within the test bench which in turn results in a **DVCOUNT** of zero. This condition is of no consequence in this test case.

```
***--------------------------------------------------------------------------***
tc00_postsim.m : V2.00 run time: Mon Sep 19 18:13:30 2005
Using postsim_lib: 1.40: 05-08-18

Using check_register_capture: 3.10: 05-08-26

Process functional model simulator output for test case 00.
16 frames read for test case 00.
test case version: v3,05apr05
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Input configuration from MCSR regsister settings:
0(0:15)0

warning: in /rigel1/smegal/EVLAC/corrchip/postsim_lib/scale_lag_data.m near line 62, column 7:

>>> clags = (CCchain.clag - data_bias * (1 + 1i)) / dvcount;


warning: division by zero

Initial (single dump) timestamp (word 0): 222333444

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file for test case 00.

Check OK: DESSR bits are set as expected.

Data bias for test case 00 (split accumulator mode): 5416192
Check OK: Accumultor value in test case 00 is as expected.

***--------------------------------------------------------------------------***
```

Text 2.3.1 console log for script tc00_postsim.

```
# ------------------------------------------------------------------------------
# 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 results with: split accumulator bias 5416192
#              Additional details:   valid data reset; 660 sample bias is 42240.
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111110  XSTATUS: X-SE_CLK: 11111110
11111110  YSTATUS: Y-SE_CLK: 11111110
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.3.2 content of file mcb_reg00.txt.

2.3.3 test case 32, 33

Synopsis: No extraordinary plot features or table entries are noted using general test vector set DS4. Bit exact comparison with the behavioural model using special test vector set DS14 was successful (see Text 2.3.4); warnings reported in the log of captured data and register content (Text 2.3.5) while using test vector set DS14 are of no consequence (see notes below).

Notes:

(1) Lag data based on test vector set DS14 and used in bit exact comparisons were generated with a chip RTL model version that predates that used for other test cases of this document. The RTL code for the **CMAM** was unchanged between these versions, hence, the bit exact result remains valid.

(2) The RTL model of point (1) above was stimulated by a version of the test bench which provided **BBID** data embedded within the input streams different from that of more recent versions. The current version of function *check_register_capture* is designed for the more recent test bench and, hence, reported warnings in the log.

(3) Statistical parameters presented in Table 2.3.1 and Table 2.3.2 are used as a normalization base in subsequent sections.

(4) The plot of Illustration 2.3.1 depicts the differential phase between input data streams applied to the behavioural model during bit exact tests involving test case 32 in order to demonstrate that complete phase state coverage is obtained. **XDVALID** is overlaid on the plot to identify the window of generally valid data. The scripts that generated this plot are found in sections 3.22.8 and 3.22.9.

```
***--------------------------------------------------------------------------***
tc32_33_postsim.m : V2.00 run time: Mon Sep 19 18:30:07 2005
Using postsim_lib: 1.40: 05-08-18

Using check_register_capture: 3.10: 05-08-26

Process functional model simulator output for test case 32.
16 frames read for test case 32.
test case version: v3,05apr19
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Input configuration from MCSR regsister settings:
0(0:15)0


Initial (single dump) timestamp (word 0): 222333444

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file for test case 32.

Check OK: DESSR bits are set as expected.
```

Text 2.3.3 console log for scripts tc32_33_postsim, make_lag_tables_tc32_33, make_fd_tables_tc32_33, validate_bit_exact; part (1).

```
Process functional model simulator output for test case 33.
16 frames read for test case 33.
test case version: v3,05apr19
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Input configuration from MCSR regsister settings:
0(0:15)0


Initial (single dump) timestamp (word 0): 4294967295

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file for test case 33.

Check OK: DESSR bits are set as expected.
Read lag data for tc32 and tc33 created with the test vector data set
DS14 (used for bit exact tests. Use the same mcb_reg files as for DS4.
16 frames read for test case 32_DS14.
test case version: v3,05apr19
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


Initial (single dump) timestamp (word 0): 222333444

WARNINGS encountered during register status and data capture checks;
                        details in log file for test case 32_DS14.

16 frames read for test case 33_DS14.
test case version: v3,05apr19
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


Initial (single dump) timestamp (word 0): 4294967295

WARNINGS encountered during register status and data capture checks;
                        details in log file for test case 33_DS14.
***--------------------------------------------------------------------------***

octave:76> make_lag_tables_tc32_33

Writing tables.
octave:77> make_fd_tables_tc32_33

Writing freq. domain difference tables.
octave:78> validate_bit_exact

Finding the difference between functional and behavioural model lags
formed using bit exact test vectors.

No differences exceeding the floating point precision limit of this
machine were found beween models; behavioural and functional models for test
case 32 agree on a bit exact basis.

No differences exceeding the floating point precision limit of this
machine were found beween models; behavioural and functional models for test
case 33 agree on a bit exact basis.
```

Text 2.3.4 console log for scripts tc32_33_postsim, make_lag_tables_tc32_33, make_fd_tables_tc32_33, validate_bit_exact; part (2).

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:0 and Y:0.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                             template of expected states.
```

Text 2.3.5 content of register_capture log file for test case 32 using test vector set DS14 (for bit exact tests). This log is identical to its counterpart of test case 33.

test case	*test vector set*	*dvcount*	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$
32	DS4	8694	0.2105	0.2078	4.65e-04	1.23e-04	0.0271	0.0279
33	DS4	17360	0.1531	0.1585	9.82e-04	-6.34e-04	0.0228	0.0203
32	DS14	5274	0.1141	0.1145	1.27e-04	-9.94e-05	0.0154	0.0144
33	DS14	10558	0.0939	0.0857	2.74e-04	-3.38e-04	0.0141	0.0132

Table 2.3.1 test cases 32 and 33; statistical parameters of the difference between RTL and behavioural model lag coefficients.

test case	*test vector set*	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$
32	DS4	9.83e-03	1.38e-02	4.42e-05	1.08e-04	7.47e-04	7.54e-04
33	DS4	1.10e-02	7.49e-03	-6.07e-05	4.78e-05	6.50e-04	5.28e-04
32	DS14	1.01e-02	3.15e-03	-9.96e-06	-4.22e-06	4.32e-04	3.28e-04
33	DS14	9.42e-03	3.22e-03	-1.10e-05	3.25e-06	4.30e-04	3.24e-04

Table 2.3.2 test cases 32 and 33; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

test case	*test vector set*	*stream number*	*dump interval*	*bias*	*dump-bias*	*DVCOUNT: behavioural model*	*DVCOUNT : RTL model*
32	DS4	0	10000	9724	276	8741	8694
32	DS14	0	10000	9724	276	5342	5274
33	DS4	0	20000	19724	276	8741	17360
33	DS14	0	20000	19724	276	5342	10558
32	DS14-S	n-a	n-a	n-a	n-a	5274	n-a
33	DS14-S	n-a	n-a	n-a	n-a	5279	n-a

*Table 2.3.3 test cases 32 and 33; bias and data valid count (**DVCOUNT**) for both RTL and behavioural models.*

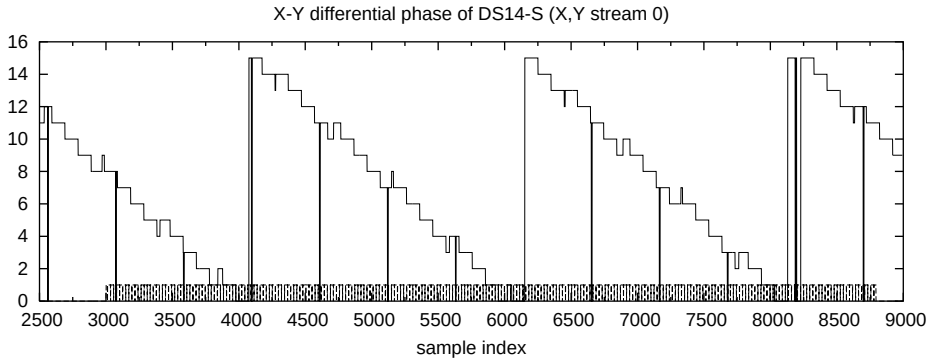

*Illustration 2.3.1 Differential phase of the input data streams applied to the behavioural model during bit exact tests involving test case 32. The window of generally valid data is identified by the overlay of **XDVALID**.*

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111110  XSTATUS: X-SE_CLK: 11111110
11111110  YSTATUS: Y-SE_CLK: 11111110
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.3.6 content of file mcb_reg32.txt.

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
00000101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111110  XSTATUS: X-SE_CLK: 11111110
11111110  YSTATUS: Y-SE_CLK: 11111110
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.3.7 content of file mcb_reg33.txt.

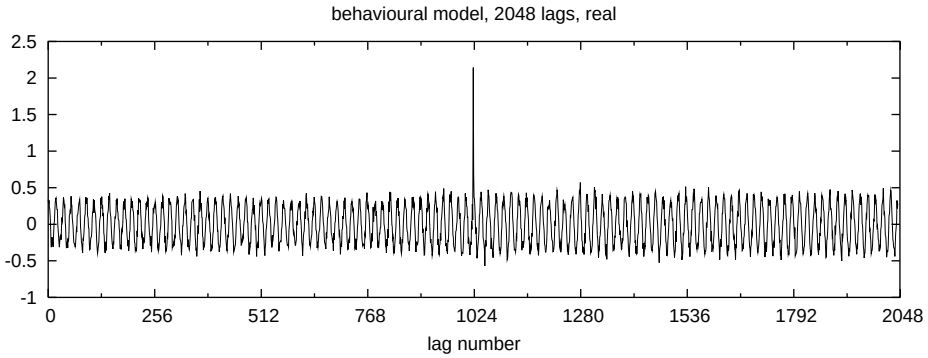

Illustration 2.3.2

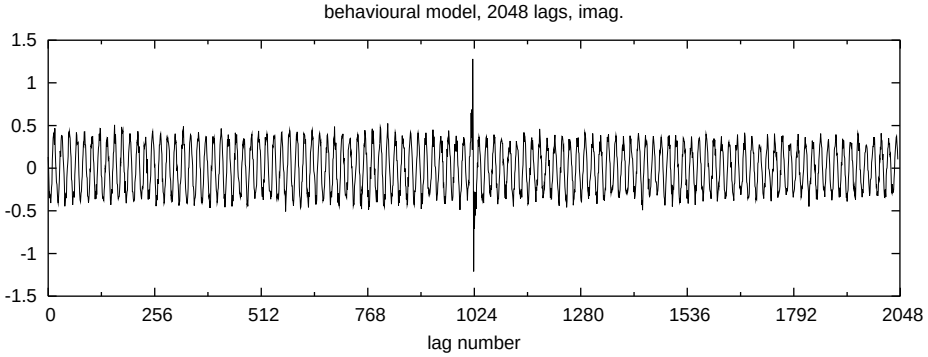

Illustration 2.3.3

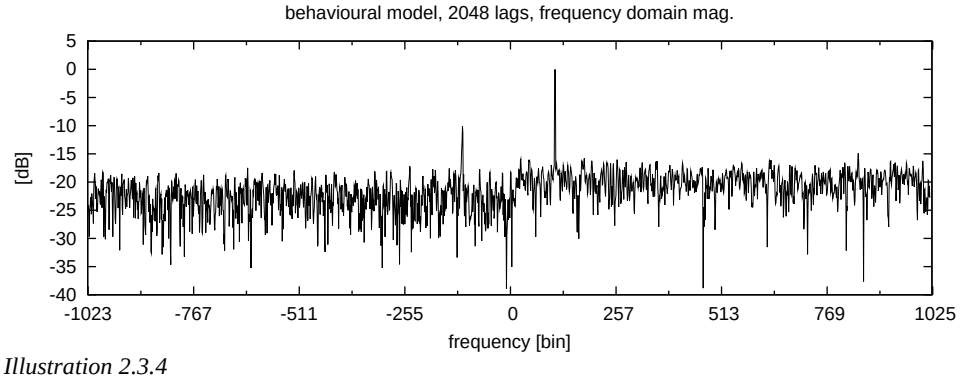

Illustration 2.3.4

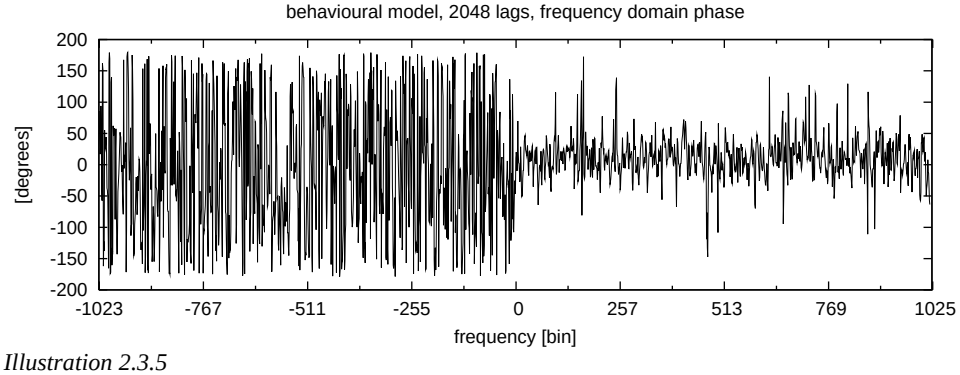

Illustration 2.3.5

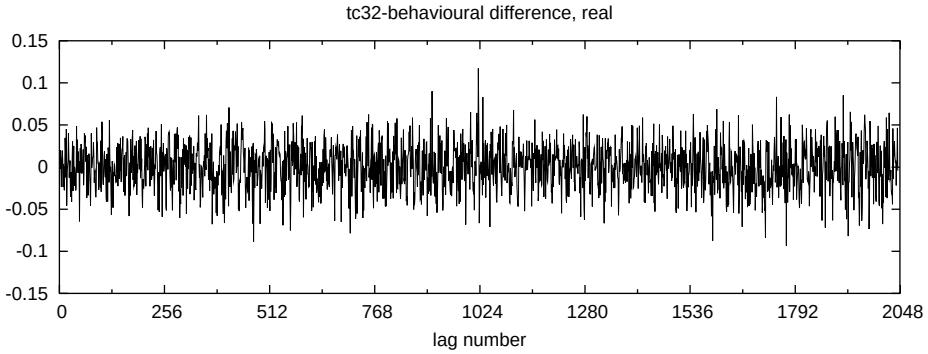

Illustration 2.3.6

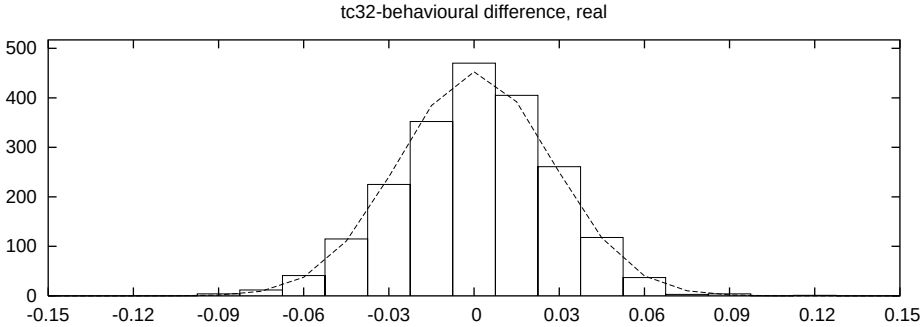

Illustration 2.3.7

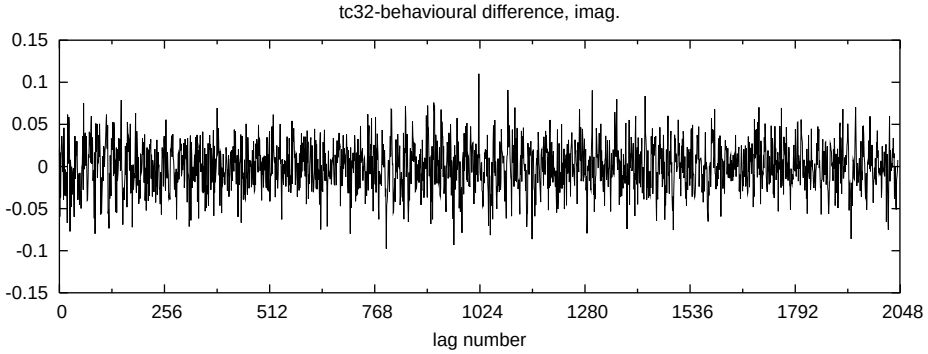

Illustration 2.3.8

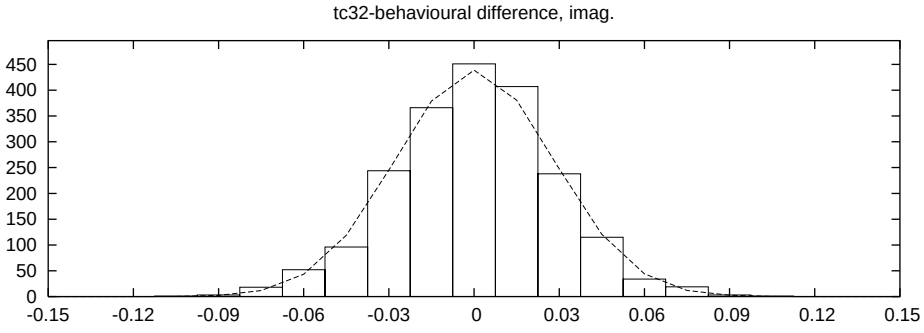

Illustration 2.3.9

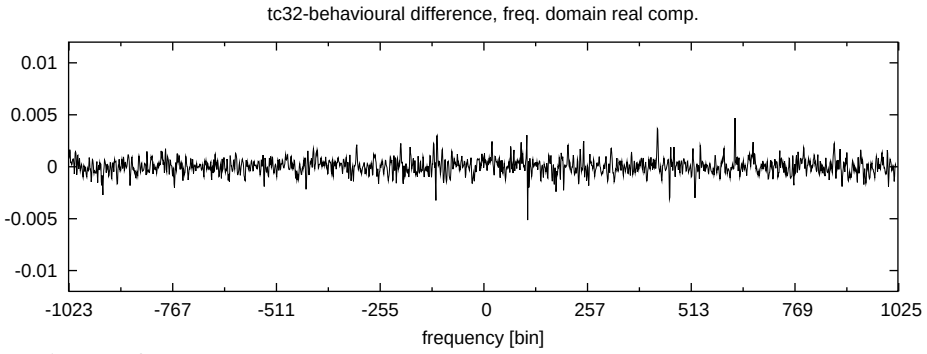

Illustration 2.3.10

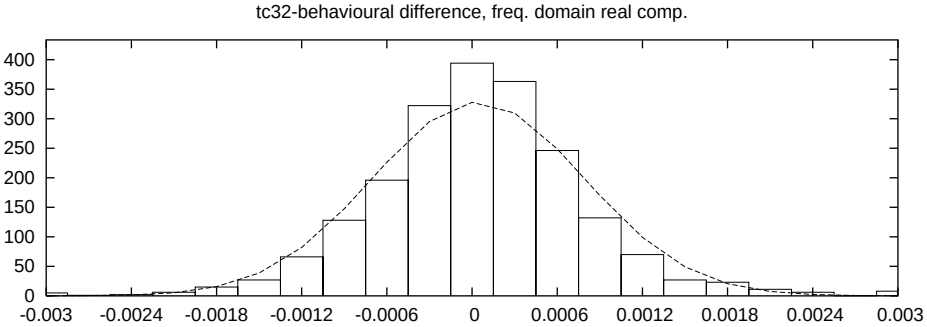

Illustration 2.3.11

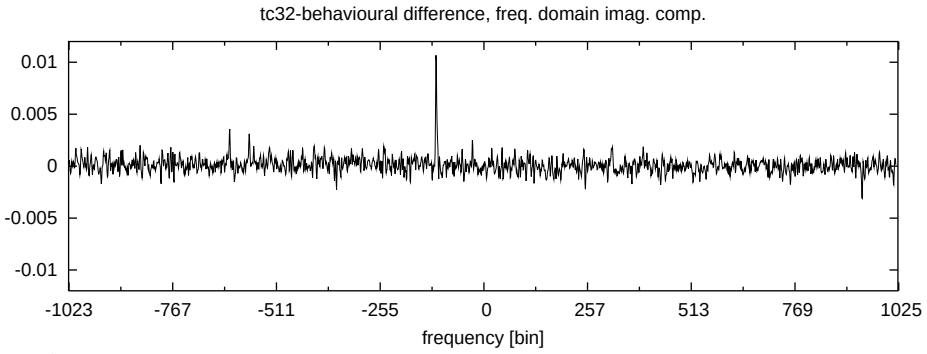

Illustration 2.3.12

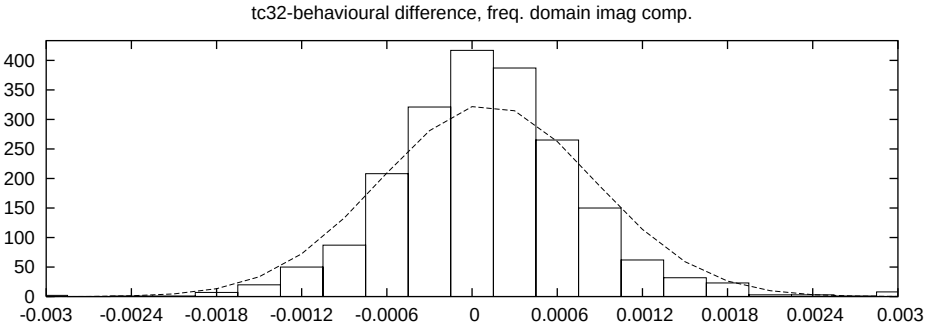

Illustration 2.3.13

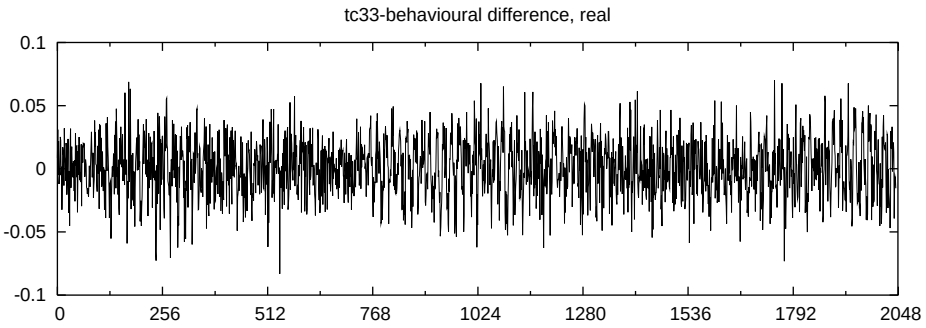

Illustration 2.3.14

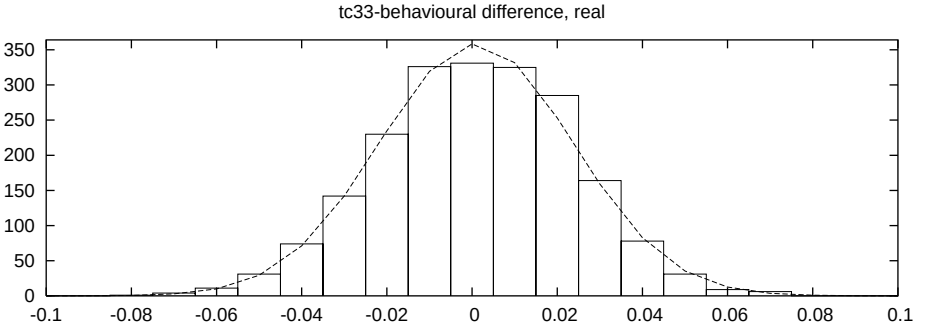

Illustration 2.3.15

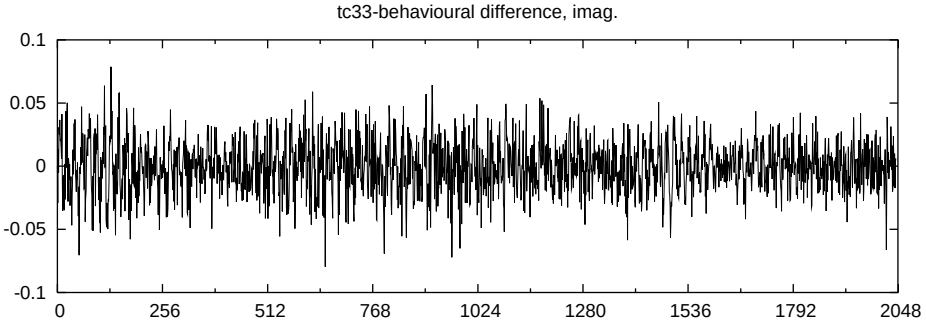

Illustration 2.3.16

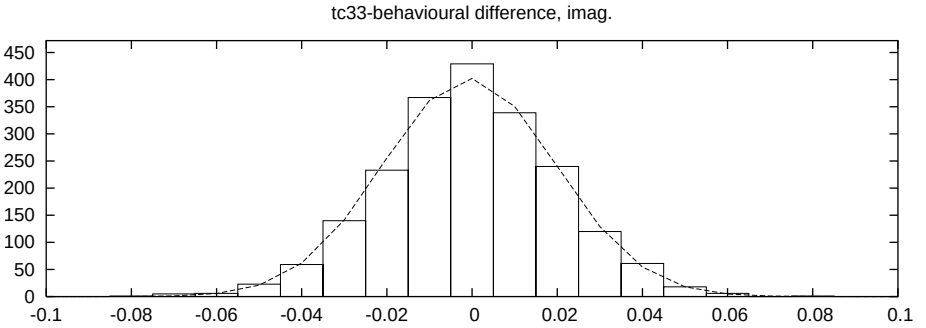

Illustration 2.3.17

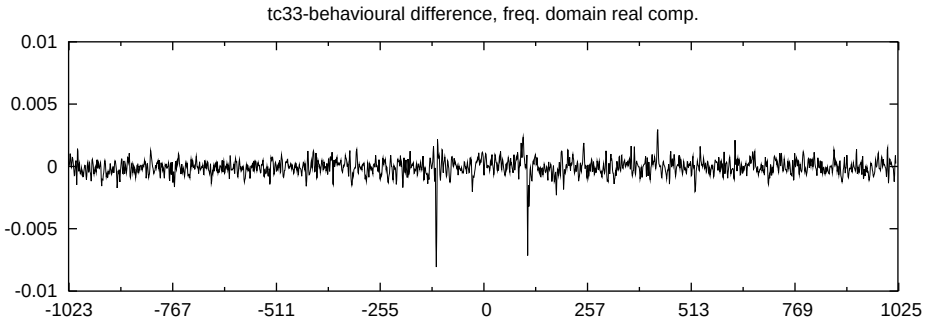

Illustration 2.3.18

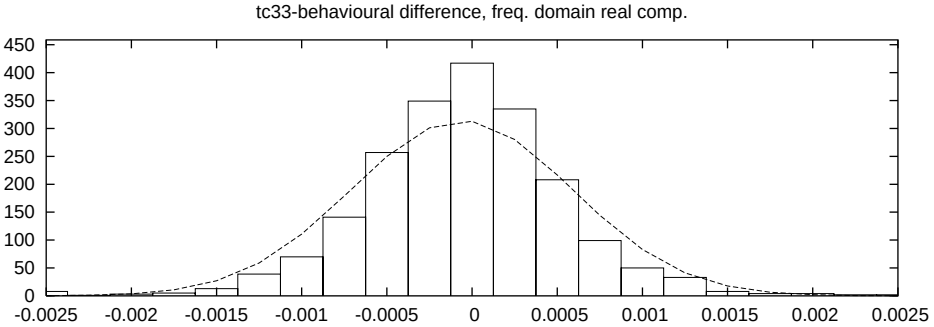

Illustration 2.3.19

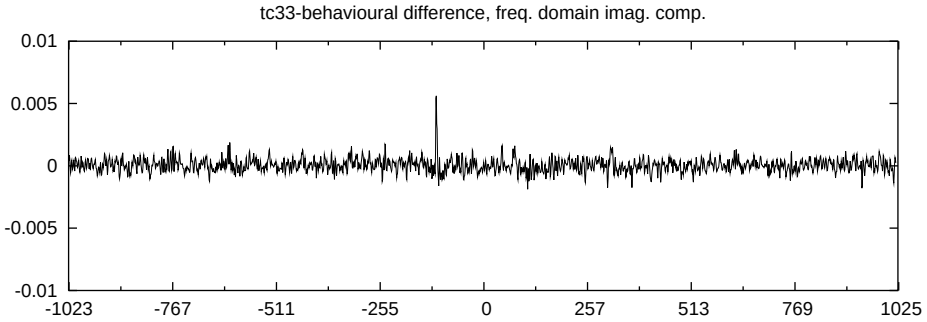

Illustration 2.3.20

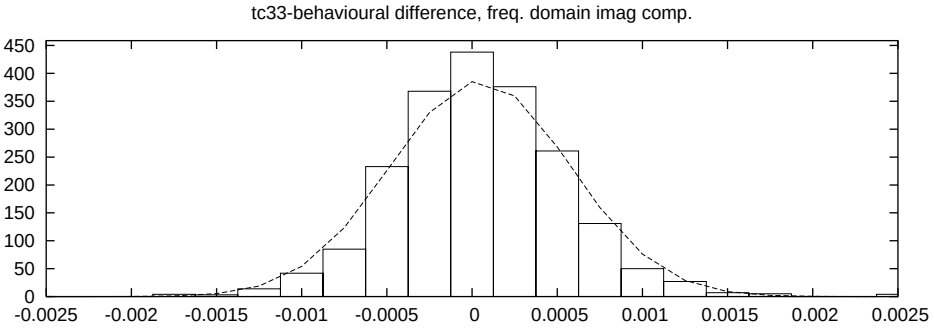

Illustration 2.3.21

2.4 accumulator overflow (test case 01)

Test cases: 01

Synopsis: Accumulator overflow is validated with a single test case. No advisories or warnings regarding captured data or register content were logged. The expected value for data bias was found in the accumulator of all cells.

Notes:

(1) A warning would have been reported in the log for *check_register_capture* if data bias differed between cells.

(2) The *division by zero* warning in the console log of Text 2.3.1 is a consequence of maintaining the data valid line (**DVCOUNT**) low by a parameter setting within the test bench which in turn results in **DVCOUNT** of zero. This condition is of no consequence in this test case.

```
***--------------------------------------------------------------------------***
tc01_postsim.m : V2.00 run time: Mon Sep 19 19:23:53 2005
Using postsim_lib: 1.40: 05-08-18

Using check_register_capture: 3.10: 05-08-26

Process functional model simulator output for test case 01.
16 frames read for test case 01.
test case version: v2,05apr06
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Input configuration from MCSR regsister settings:
0(0:15)0

warning: in /rigel1/smegal/EVLAC/corrchip/postsim_lib/scale_lag_data.m near line 62, column 7:

>>> clags = (CCchain.clag - data_bias * (1 + 1i)) / dvcount;


warning: division by zero

Initial (single dump) timestamp (word 0): 222333444

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file for test case 01.

Check OK: DESSR bits are set as expected.

Data bias for test case 01 (split accumulator mode): 132352
Check OK: Accumultor value in test case 01 is as expected.

***--------------------------------------------------------------------------***
```

Text 2.4.1 test case 01; console log for scripts tc01_postsim.

```
# ------------------------------------------------------------------------------
# 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.
#
# ------------------------------------------------------------------------------
10010101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=1 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111110  XSTATUS: X-SE_CLK: 11111110
11111110  YSTATUS: Y-SE_CLK: 11111110
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.4.2 test case 01; content of file mcb_reg01.txt.

2.5 normal dump and readout

Test cases: 02, 03, 04, 05, 06, 08, 09, 10, 11, 12, 14, 25, 28, 30, 64

2.5.1 organization

This section is divided into subsections according to test case (or group of test cases). Each subsection contains (in approximately the following order)

(1) a post simulation synopsis;

(2) optional notes, if any, that detail outstanding features or conditions of the post simulation result;

(3) a *post simulation log* that documents script execution;

(4) a table of statistical parameters describing the arithmetic difference between the RTL simulation and behavioural model lag coefficients;

(5) a table of statistical parameters describing the arithmetic difference, between adjacent dumps, of the RTL simulation and behavioural model lag coefficients;

(6) a table of statistical parameters describing the arithmetic difference between the RTL simulation and behavioural model frequency domain coefficients;

(7) a table of bias (expressed in number of samples) accumulated by cells associated with each stream;

(8) any miscellaneous test case specific tables;

(9) selected plots of the RTL simulation result and the above model differences;

(10) content of the MCB register file that is provided by the test bench.

Additional notes on the above entities now follow.

Post simulation log: is the console output that results from running one or more post simulation scripts. The script *tc**nn**_postsim* (where ***nn*** is the test case number) is executed to read and perform processing common to all test cases of this section. Details of its operation are found in relevant sections of chapter 3. Output of any additional scripts required to validate specific operations is appended to the log and described in the *synopsis* of each test case.

Validation of a normal dump and readout operation occurs on a comparative basis between test cases, with special reference to 32 and 33. 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 statistical parameters (points 5, 6 above) from those of the reference indicate possible validation failure. A threshold for failure is not based on a statistical model but empirically set through experiments such as those of 2.5.8 in which test cases were modified to simulate a validation failure.

Cells, chains and dumps are indexed as described in 2.2. The timestamp is incremented by 1 in the case of the normal dumps of this section. A initial timestamp of 2^{32} will roll over to zero for subsequent dumps; this situation is noted in the *post simulation log*.

Register and captured data consistency checking is performed by a call to function *check_register_capture*. Script *tc**nn**_postsim* reports *Check OK: no inconsistencies reported in captured data* and the output log of *check_register_capture* is not printed (except once for reference in 3.2) if no inconsistencies are found; notice of advisories or warnings appear in the *post simulation log* and the output log of *check_register_capture* is included as part of this report if some inconsistency or unexpected register content is detected. The template to which register contents are compared is generated on the basis of each test case within the test plan document and supplied as an argument to *check_register_capture* (see each case of chapter 3).

Tables of statistics are labelled based on the nomenclature presented in 2.2. Thus,

$\overline{\Re[f-b]}$: is the mean of the real part of *[f - b]*,

$\overline{\Im[f-b]}$: is the mean of the imaginary part of *[f - b]*,

$range\,\Re[f-b]$: is the range in real part of *[f- b]*,

$range\,\Im[f-b]$: is the range in imaginary part of *[f - b]*,

$\sigma_{\Re[f-b]}$: is the standard deviation of the real part of *[f - b]*,

$\sigma_{\Im[f-b]}$: is the standard deviation of the real part of *[f - b]* and

$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$: is the standard deviation of the real part of *[f - b]* normalized to that of a 12000 sample test case based on a fully concatenated (2048 lag) chain. Analogous quantities are tabulated for the frequency domain coefficients, $[f^F - b^F]$ and the dump to dump lag coefficients, *[f - f]*. The normalized standard deviation is replaced within the tables of dump to dump lag coefficients by the quantities,

$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$: the standard deviation ratio of the real parts of *[f - f]* to *[b-b]* and

$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$: the standard deviation ratio of the imaginary parts of *[f - f]* to *[b-b]*.

Normalization within the lag domain tables is performed according to

$$\sigma_N = \frac{0.02}{\sqrt{\frac{n}{m}}} \quad ,$$

where *n* is the sample count in multiples of 12000 and *m* is the shift enable (**SE_CLK**) clock modulus (always set for *m = 1*). Likewise, for the frequency domain case,

$$\sigma_N = \frac{6\times10^{-4}}{\sqrt{\frac{n}{m}}} \quad .$$

The quantity *peak count* is a count of lag coefficients exceeding a threshold, which in all cases here is the sum of occurrences of real and imaginary coefficients exceeding $3\sigma_{\Re[f-b]}$ and $3\sigma_{\Im[f-b]}$ respectively for the lag coefficients, $10\sigma_{\Re[f^F-b^F]}$ and $10\sigma_{\Im[f^F-b^F]}$ in the frequency domain case. The thresholds used are logged in the *post simulation log.*

The quantity, *co-dump peaks* within the dump to dump tables is a list of lag numbers associated with coefficient peaks (as described above) that are coincident with other dumps .

The *plots* column within the above tables contain ' * ' in cells for which plots are presented. These plots were manually selected during initial table inspection, usually on the basis of worst case or outstanding statistical coefficients. Samples beyond the displayed domain in the histogram plots are dumped into the extreme most bins. Overlaid pdf curves of the histogram plots are generated on the basis of the mean and standard deviation supplied in the accompanying tables.

The bias table contains the *applied dump* (dump interval in number of samples as specified in the test case); the *bias* obtained for all chains dumping on the specified *stream*; the *difference* between the bias and the applied dump for chains for which the dump overrun bit is reset (!**OVR**) for all cells of the chain; an analogous difference for chains in which the dump overrun bit is set (**OVR**). The latter quantity was included for completeness as, by definition, all normal dump and readout cases do not involve dump overruns. The difference between applied dump and bias should be constant (when **OVR** is reset) and represent approximately 1 μs of integration dead time (276 samples at 256 *Msamples/s*).

Execution date policy: key functions used in processing data such as those in the function library *postsim_lib* and functions involved in generating tables were prompted for version by *tc**nn**_postsim* and recorded in the postsim log; the script *tc**nn**_postsim* records the time and date of execution; scripts that follow *tc**nn**_postsim* in the post simulation log (after '***--- ---***') such as plot generating scripts are *not* subject to version or time logging and may have been subsequently re-executed. *However, in such cases, care was taken (including bit exact comparison with documented tables and plots) to insure that these extended scripts were operating with the data presented in this document; source codes for these scripts are listed in the subsections of their respective test cases (chapter 3).*[2]

2 This was done to save time re-importing unchanged data into the tables of this document in the event minor changes were made to extended scripts.

2.5.2 test case 02

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged. All 16 chains (single cell) share a common input data stream and passed a bit exact comparison between chains (logged in Text 2.5.1). Plots are presented for a single chain as all chains generate identical lag data.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc02_postsim.m : V2.00 run time: Fri Aug 26 10:35:47 2005
Using postsim_lib: 1.40: 05-08-18

160 frames read for test case 02.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
0(0:0)0  0(1:1)0  0(2:2)0  0(3:3)0
0(4:4)0  0(5:5)0  0(6:6)0  0(7:7)0
0(8:8)0  0(9:9)0  0(10:10)0  0(11:11)0
0(12:12)0  0(13:13)0  0(14:14)0  0(15:15)0

Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc02
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.


***--------------------------------------------------------------------------***
octave:3> tc02_chain_similarity_check

No differences exceeding the floating point precision limit of this
machine were found between chains for any of the 10 dumps; all chains generated
identical lag data.

octave:4> tc02_make_plots
```

Text 2.5.1: console log for scripts tc02_postsim, tc02_chain_similarity_check, tc02_make_plots

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	4.39e-03	-1.38e-03	0.1669	0.0985	0.0272	0.0223	1.36	1.11	1	*
1	2	2.82e-05	1.57e-03	0.1216	0.1286	0.0253	0.0231	1.26	1.16	0	
1	3	-5.80e-05	2.16e-05	0.0933	0.1054	0.0216	0.0248	1.08	1.24	0	
1	4	-6.35e-04	-1.02e-03	0.1445	0.1235	0.0280	0.0258	1.40	1.29	0	
1	5	-4.17e-04	3.76e-04	0.1493	0.1662	0.0269	0.0290	1.34	1.45	2	*

chain	dump	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	range $\Re[f-b]$	range $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	peak count	plot
1	6	-4.62e-04	-2.18e-03	0.0994	0.1257	0.0201	0.0235	1.01	1.17	1	
1	7	2.21e-03	-4.38e-04	0.1047	0.1366	0.0201	0.0254	1.00	1.27	2	
1	8	-1.75e-03	-2.56e-05	0.1377	0.1342	0.0242	0.0258	1.21	1.29	0	
1	9	-1.23e-04	1.71e-03	0.1613	0.1272	0.0259	0.0255	1.29	1.28	1	
1	10	-8.92e-04	-8.34e-04	0.1584	0.1354	0.0272	0.0240	1.36	1.20	2	

Table 2.5.1: test case 02; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	dump	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	range $\Re[f-f]$	range $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	co-dump peaks
1	1	5.19e-04	3.57e-03	0.4313	0.4946	0.0931	0.0994	1.17	1.06	null
1	2	1.58e-03	9.53e-03	0.3789	0.4052	0.0835	0.0801	0.99	1.03	null
1	3	-1.59e-02	-1.17e-02	0.4930	0.4227	0.0911	0.0847	1.07	0.99	null
1	4	2.07e-02	6.63e-03	0.4573	0.4704	0.0914	0.0794	1.05	0.91	null
1	5	-1.02e-02	-4.67e-04	0.5579	0.4584	0.1090	0.0829	1.04	1.03	null
1	6	7.89e-03	-3.05e-03	0.4708	0.4614	0.0783	0.0833	1.06	1.05	null
1	7	-1.14e-02	-2.07e-03	0.4183	0.4523	0.0770	0.0941	1.03	1.11	null
1	8	-1.76e-03	1.97e-03	0.5274	0.4591	0.0913	0.0831	1.12	1.04	null
1	9	-5.92e-04	1.59e-03	0.5007	0.3197	0.1033	0.0779	1.08	1.03	null

Table 2.5.2: test case 02; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	dump	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	range $\Re[f^F-b^F]$	range $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	peak count	plot
1	1	1.40e-03	-1.54e-04	0.0143	0.0099	2.79e-03	2.11e-03	4.6	3.5	0	*
1	2	3.39e-05	9.74e-04	0.0187	0.0135	2.68e-03	2.25e-03	4.5	3.8	0	
1	3	-6.45e-04	-4.90e-04	0.0207	0.0142	2.82e-03	2.26e-03	4.7	3.8	0	
1	4	2.64e-04	-9.53e-04	0.0164	0.0270	2.92e-03	3.30e-03	4.9	5.5	0	
1	5	5.13e-04	1.20e-03	0.0166	0.0323	2.88e-03	3.86e-03	4.8	6.4	0	*
1	6	-4.95e-05	2.40e-04	0.0129	0.0121	2.27e-03	2.15e-03	3.8	3.6	0	
1	7	-1.50e-04	-9.95e-04	0.0147	0.0157	2.34e-03	2.33e-03	3.9	3.9	0	
1	8	1.76e-04	-5.46e-04	0.0206	0.0126	2.93e-03	2.23e-03	4.9	3.7	0	
1	9	7.29e-04	1.00e-03	0.0220	0.0157	3.05e-03	2.47e-03	5.1	4.1	0	
1	10	-2.11e-04	3.50e-04	0.0141	0.0232	2.61e-03	3.06e-03	4.4	5.1	0	

Table 2.5.3: test case 02; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	applied dump	bias:!OVR	applied dump - bias: !OVR	bias: OVR	applied dump - bias: OVR
0	12000	11724	276	0	n-a

Table 2.5.4 test case 02; applied dump intervals and bias.

```
--------------------------------------------------------------------------------
# TEST 2:
# ver: v3,05jun14
# 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: 12000-1
#              Distinct samp/dump:   12000-1
#              Active SE clocks:     input 0 SE_CLK always on
#              Modulo se_clk gen:    none
#              Compare result with:  identical operation between cells
#              Additional details:
#
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10011001  CCCSCR_0_1:   CCC1: Y=sec X=pri   CCC0: Y=sec X=pri
10011001  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=sec X=pri
10011001  CCCSCR_4_5:   CCC5: Y=sec X=pri   CCC4: Y=sec X=pri
10011001  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=sec X=pri
10011001  CCCSCR_8_9:   CCC9: Y=sec X=pri   CCC8: Y=sec X=pri
10011001  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=sec X=pri
10011001  CCCSCR_12_13: CCC13: Y=sec X=pri CCC12: Y=sec X=pri
10011001  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=sec X=pri

01010000  CCQR_0_1: Q1: YSw1=pri YSw0=sec XSw1=pri XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01010101  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=sec  Q2: YSw1=pri YSw0=sec XSw1=pri XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111110  XSTATUS: X-SE_CLK: 11111110
11111110  YSTATUS: Y-SE_CLK: 11111110
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.2: content of file MCB_reg02.txt

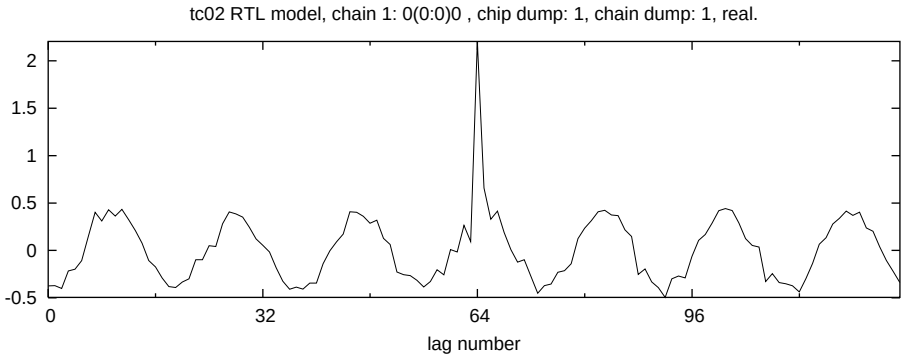

Illustration 2.5.1

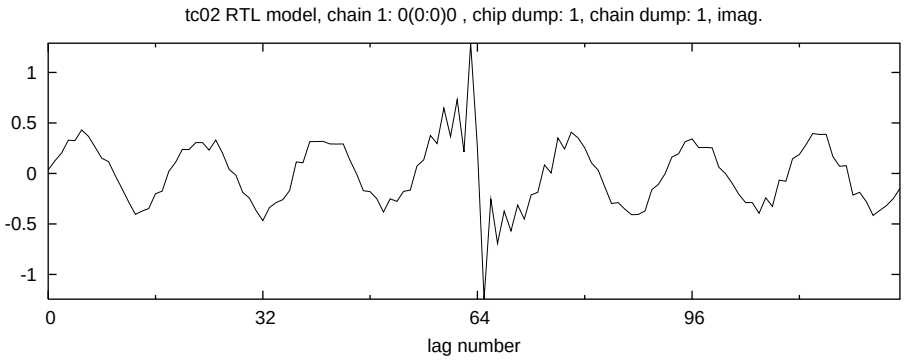

Illustration 2.5.2

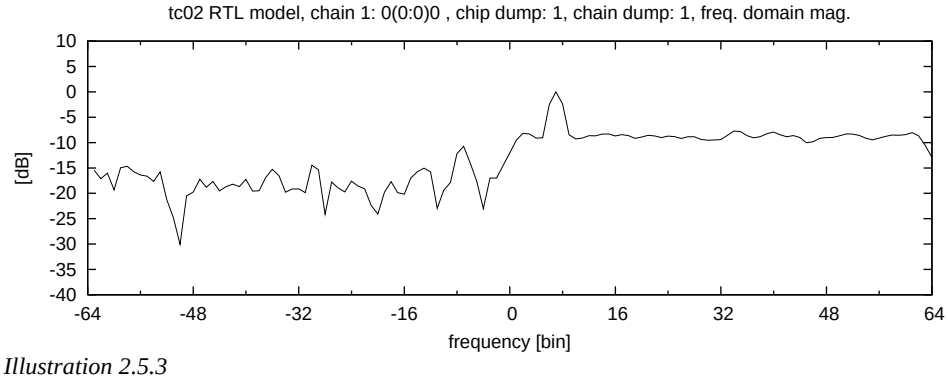

Illustration 2.5.3

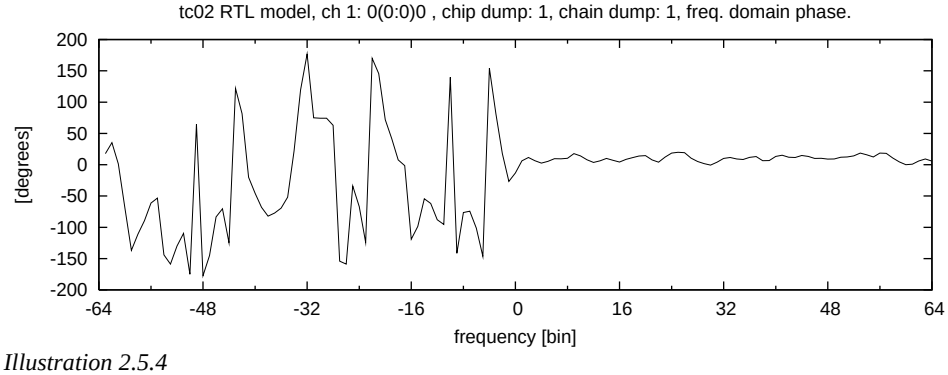

Illustration 2.5.4

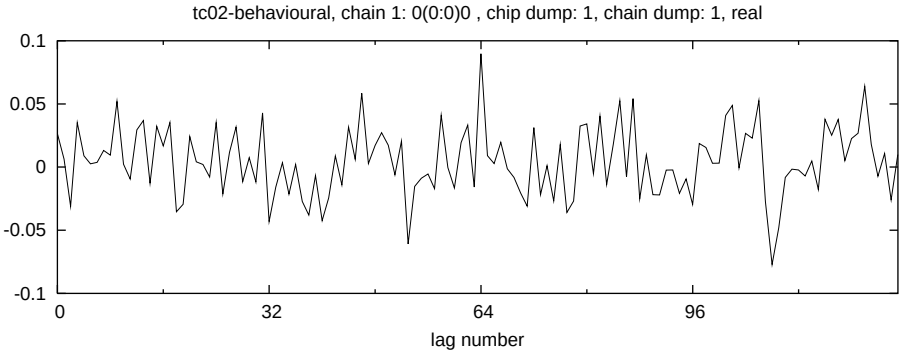

Illustration 2.5.5

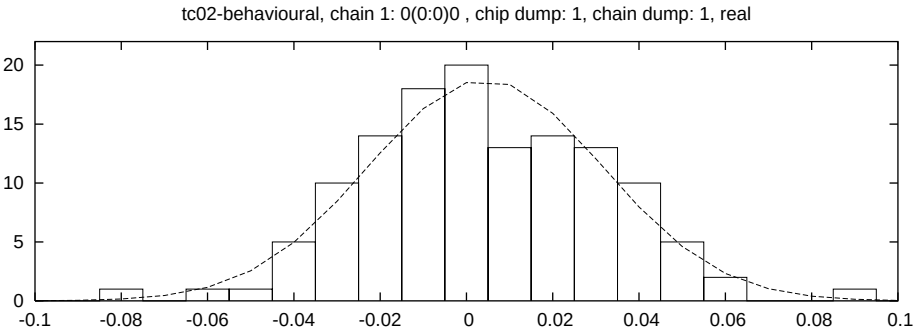

Illustration 2.5.6

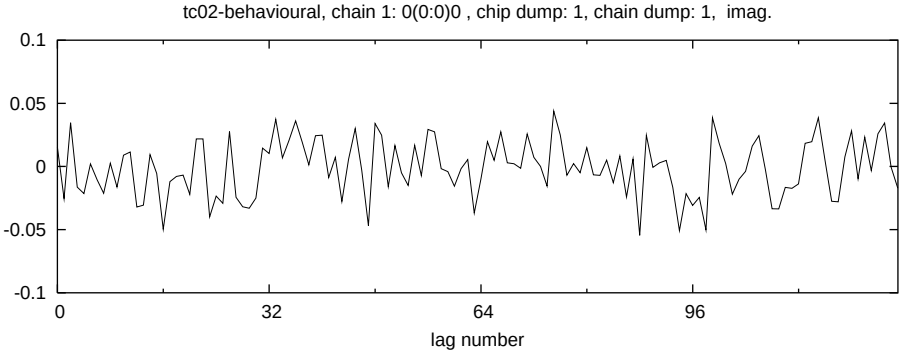

Illustration 2.5.7

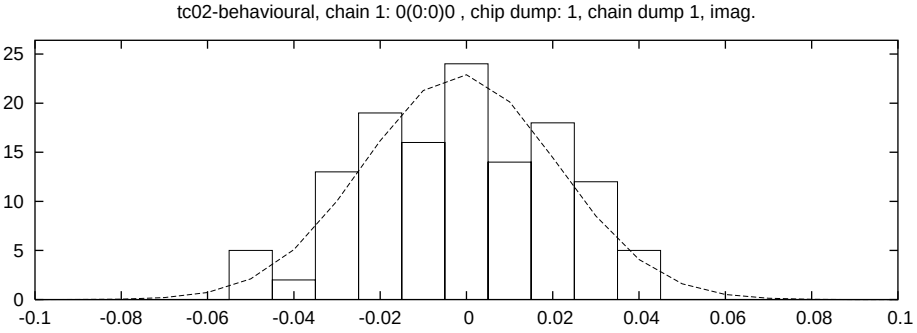

Illustration 2.5.8

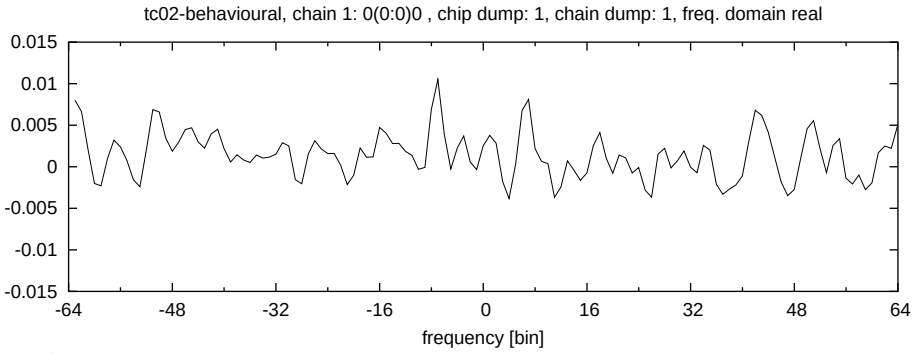

Illustration 2.5.9

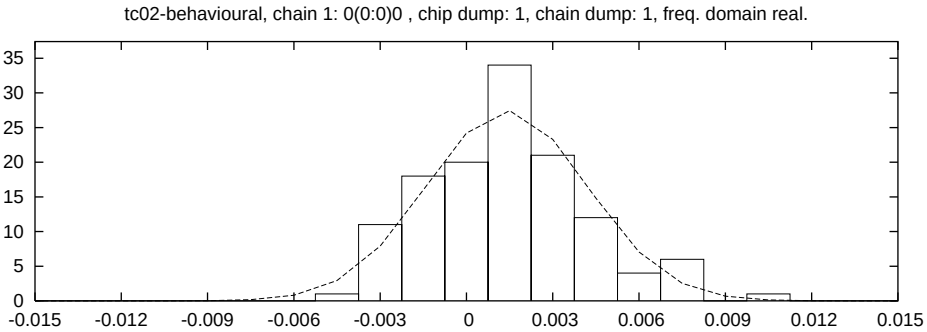

Illustration 2.5.10

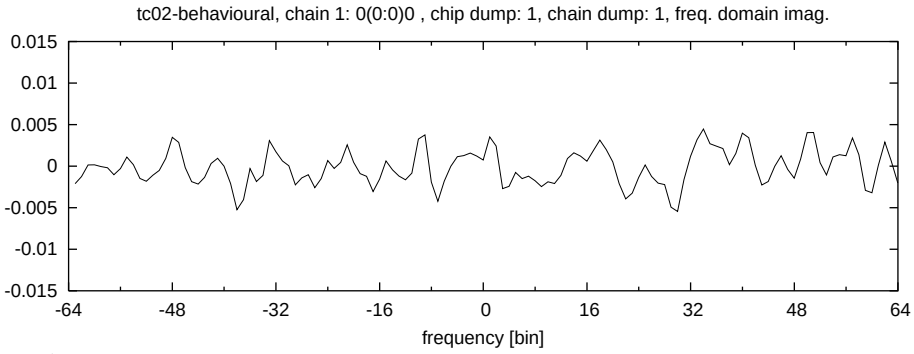

Illustration 2.5.11

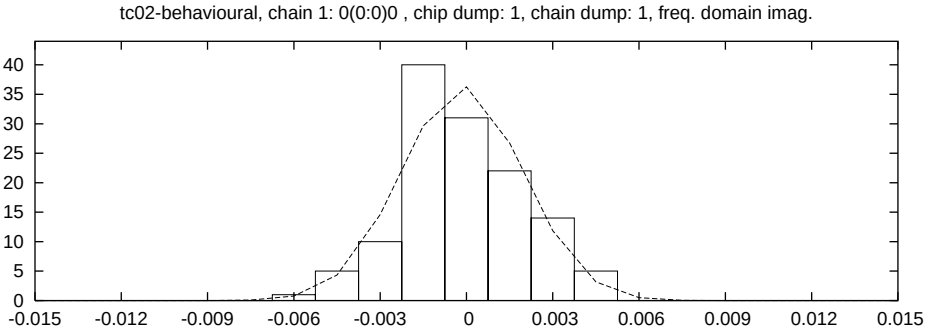

Illustration 2.5.12

2.5.3 test case 03

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged.

Notes: This test case is referenced in *on line detection of errors*, section 2.10 (missing **X|Y-SE_CLK**). This test case also sets all bits of register **DESSR**.

```
***------------------------------------------------------------------------***
tc03_postsim.m : V2.00 run time: Fri Aug 26 16:18:15 2005
Using postsim_lib: 1.40: 05-08-18

120 frames read for test case 03.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data was not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
0(0:3)0  2(4:7)2  4(8:11)4  6(12:15)6

Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc03
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***------------------------------------------------------------------------***
```

Text 2.5.3: console log for scripts tc03_postsim

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	1.74e-03	-4.40e-04	0.1795	0.1298	0.0266	0.0224	1.33	1.12	2	
1	2	-7.87e-04	1.61e-04	0.1593	0.1512	0.0269	0.0241	1.35	1.21	3	
1	3	-4.11e-04	2.13e-03	0.1892	0.1491	0.0233	0.0259	1.17	1.29	3	
1	4	1.21e-03	-2.89e-03	0.1718	0.1430	0.0267	0.0239	1.33	1.19	2	
1	5	1.17e-03	1.09e-03	0.1557	0.1816	0.0261	0.0287	1.31	1.44	3	
1	6	-5.55e-04	1.65e-06	0.1342	0.1382	0.0226	0.0243	1.13	1.22	2	
1	7	9.68e-05	-5.24e-04	0.1609	0.1709	0.0236	0.0279	1.18	1.40	4	
1	8	2.67e-04	-2.68e-04	0.1971	0.1523	0.0284	0.0293	1.42	1.46	2	
1	9	-2.74e-04	8.54e-04	0.1547	0.1548	0.0229	0.0244	1.14	1.22	6	*
1	10	-1.48e-04	-3.15e-04	0.1748	0.1346	0.0238	0.0242	1.19	1.21	2	
1	11	8.37e-05	-4.34e-04	0.1968	0.1633	0.0313	0.0281	1.57	1.40	3	
1	12	2.60e-04	7.43e-04	0.1826	0.1512	0.0277	0.0258	1.38	1.29	3	
1	13	-3.61e-04	-1.30e-03	0.1432	0.1385	0.0238	0.0238	1.19	1.19	2	

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	14	8.05e-04	-9.92e-04	0.1502	0.1508	0.0262	0.0251	1.31	1.25	3	
1	15	9.85e-04	6.68e-04	0.1990	0.1400	0.0248	0.0234	1.24	1.17	3	
1	16	-1.19e-03	3.36e-04	0.1511	0.1532	0.0231	0.0283	1.15	1.41	4	
2	2	1.89e-04	5.29e-04	0.0737	0.0740	0.0124	0.0119	0.88	0.84	5	
2	4	-1.59e-04	-2.15e-04	0.0658	0.0794	0.0120	0.0131	0.85	0.92	1	
2	6	3.25e-04	3.37e-04	0.0831	0.0745	0.0140	0.0134	0.99	0.95	1	
2	8	-1.39e-05	1.31e-04	0.0768	0.0836	0.0112	0.0120	0.79	0.85	4	
2	10	-4.00e-04	-1.08e-04	0.0764	0.0748	0.0135	0.0129	0.95	0.91	1	*
2	12	-9.49e-05	1.15e-03	0.0747	0.0823	0.0124	0.0132	0.88	0.94	2	
2	14	-4.96e-04	-1.12e-04	0.0745	0.0781	0.0136	0.0146	0.96	1.03	0	
2	16	-4.25e-04	-1.26e-04	0.0731	0.0715	0.0114	0.0114	0.80	0.81	6	
3	4	2.20e-04	3.52e-04	0.0471	0.0417	0.0086	0.0077	0.86	0.77	2	
3	8	-1.84e-04	3.90e-04	0.0450	0.0491	0.0072	0.0082	0.72	0.82	5	*
3	12	2.34e-04	-1.38e-04	0.0516	0.0503	0.0074	0.0073	0.74	0.73	5	
3	16	-1.19e-04	5.18e-04	0.0424	0.0463	0.0070	0.0075	0.70	0.75	4	
4	8	-1.82e-04	-3.40e-04	0.0288	0.0318	0.0055	0.0054	0.78	0.76	1	*
4	16	1.56e-04	2.62e-04	0.0360	0.0332	0.0060	0.0059	0.85	0.83	1	

Table 2.5.5: test case 03; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-3.57e-03	4.05e-03	0.4976	0.5204	0.0912	0.0941	1.07	1.04	256
1	2	3.45e-03	5.26e-03	0.4755	0.4328	0.0834	0.0773	0.99	0.99	256
1	3	-6.11e-04	-7.70e-03	0.5647	0.4995	0.0894	0.0808	1.05	0.95	256
1	4	6.81e-03	2.38e-03	0.5450	0.4700	0.0903	0.0773	1.01	1.00	null
1	5	-6.84e-03	-1.22e-03	0.6165	0.5394	0.1025	0.0888	1.05	1.10	256
1	6	-4.30e-04	1.21e-03	0.4738	0.4778	0.0843	0.0823	1.06	1.04	null
1	7	-6.13e-04	-2.19e-03	0.4734	0.5109	0.0785	0.0906	0.99	1.12	null
1	8	-1.30e-03	9.89e-04	0.5146	0.4954	0.0819	0.0802	1.02	1.06	null
1	9	-1.96e-03	-1.04e-03	0.5095	0.4329	0.0896	0.0829	1.05	1.05	210
1	10	2.24e-03	1.54e-03	0.5711	0.5628	0.0950	0.0892	1.13	1.06	213
1	11	1.43e-04	-1.64e-03	0.4977	0.4703	0.0821	0.0785	0.91	0.93	213 503
1	12	3.84e-04	1.05e-03	0.5772	0.4819	0.0852	0.0868	0.96	0.97	null
1	13	2.17e-03	-2.38e-03	0.5664	0.5784	0.0880	0.0872	1.03	1.05	284
1	14	1.24e-03	3.70e-03	0.5249	0.4895	0.1013	0.0883	1.15	1.00	null
1	15	-5.44e-03	-2.24e-03	0.4346	0.5718	0.0799	0.0857	0.99	1.06	null
2	2	-5.22e-04	1.36e-03	0.3040	0.3384	0.0554	0.0583	1.03	1.00	null
2	4	5.25e-04	2.50e-04	0.3687	0.3958	0.0583	0.0605	1.03	1.01	null
2	6	-3.37e-05	-3.54e-03	0.3944	0.3243	0.0599	0.0616	1.03	1.00	null
2	8	-6.96e-04	1.29e-03	0.3840	0.3429	0.0622	0.0619	1.06	0.99	null
2	10	2.79e-03	7.78e-04	0.3150	0.3365	0.0576	0.0581	1.01	1.04	null
2	12	-2.62e-03	1.19e-03	0.3087	0.3272	0.0580	0.0588	0.96	0.99	null
2	14	1.34e-03	9.65e-04	0.3012	0.3762	0.0558	0.0566	0.95	1.03	null

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
3	4	-1.88e-04	1.84e-03	0.2692	0.2251	0.0382	0.0365	1.02	0.99	null
3	8	2.88e-04	-2.45e-03	0.2530	0.2204	0.0365	0.0384	0.99	1.04	null
3	12	-1.52e-03	2.48e-03	0.2520	0.3064	0.0449	0.0448	1.02	1.02	null
4	8	1.12e-03	-4.98e-04	0.1517	0.1496	0.0249	0.0254	0.98	0.99	null

Table 2.5.6: test case 03; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	3.99e-04	6.36e-05	0.0140	0.0096	1.44e-03	1.13e-03	2.4	1.9	0	
1	2	-1.13e-04	3.10e-04	0.0212	0.0092	1.56e-03	1.16e-03	2.6	1.9	1	
1	3	-4.36e-05	-3.09e-04	0.0093	0.0157	1.28e-03	1.36e-03	2.1	2.3	0	
1	4	1.07e-04	-1.41e-04	0.0102	0.0182	1.32e-03	1.51e-03	2.2	2.5	0	
1	5	1.20e-04	4.00e-04	0.0135	0.0248	1.25e-03	1.74e-03	2.1	2.9	1	
1	6	-3.06e-05	-2.16e-05	0.0146	0.0066	1.32e-03	1.11e-03	2.2	1.9	0	
1	7	6.69e-05	-3.24e-04	0.0091	0.0184	1.22e-03	1.38e-03	2.0	2.3	1	
1	8	1.78e-05	-9.29e-06	0.0251	0.0156	1.65e-03	1.32e-03	2.7	2.2	1	
1	9	9.01e-05	2.15e-04	0.0096	0.0116	1.27e-03	1.29e-03	2.1	2.2	0	*
1	10	-3.21e-05	-8.41e-05	0.0120	0.0141	1.22e-03	1.37e-03	2.0	2.3	0	
1	11	7.08e-05	-2.24e-04	0.0129	0.0246	1.44e-03	1.84e-03	2.4	3.1	2	
1	12	7.31e-05	2.79e-04	0.0137	0.0192	1.41e-03	1.49e-03	2.4	2.5	1	
1	13	-2.63e-04	1.77e-04	0.0103	0.0137	1.24e-03	1.33e-03	2.1	2.2	0	
1	14	-7.19e-05	-2.81e-04	0.0158	0.0113	1.47e-03	1.29e-03	2.4	2.1	0	
1	15	4.42e-04	6.50e-05	0.0123	0.0077	1.40e-03	1.22e-03	2.3	2.0	0	
1	16	-2.39e-04	2.57e-04	0.0094	0.0182	1.24e-03	1.47e-03	2.1	2.5	1	
2	2	3.87e-05	9.33e-06	0.0080	0.0033	7.66e-04	5.45e-04	1.8	1.3	0	
2	4	-7.19e-06	-7.11e-06	0.0079	0.0056	7.41e-04	6.39e-04	1.7	1.5	0	
2	6	-1.15e-05	2.13e-05	0.0102	0.0062	8.63e-04	6.16e-04	2.0	1.5	0	
2	8	2.14e-05	1.16e-05	0.0060	0.0066	6.89e-04	6.14e-04	1.6	1.4	0	
2	10	2.98e-05	1.43e-04	0.0098	0.0069	7.50e-04	6.58e-04	1.8	1.5	1	*
2	12	-4.88e-05	-2.82e-05	0.0091	0.0041	7.40e-04	5.72e-04	1.7	1.3	0	
2	14	-1.14e-05	-1.57e-05	0.0132	0.0060	9.27e-04	6.55e-04	2.2	1.5	1	
2	16	1.15e-04	-7.27e-05	0.0069	0.0057	6.32e-04	6.61e-04	1.5	1.6	0	
3	4	-2.85e-05	3.11e-06	0.0080	0.0029	5.67e-04	3.26e-04	1.9	1.1	1	
3	8	-7.42e-07	-2.30e-06	0.0059	0.0033	4.91e-04	3.12e-04	1.6	1.0	0	*
3	12	7.49e-06	-1.48e-06	0.0074	0.0028	4.52e-04	3.21e-04	1.5	1.1	1	
3	16	2.25e-05	-2.95e-06	0.0053	0.0024	4.65e-04	3.05e-04	1.6	1.0	0	
4	8	-1.02e-05	2.07e-05	0.0045	0.0016	3.87e-04	1.78e-04	1.8	0.8	0	*
4	16	5.31e-06	-1.38e-05	0.0043	0.0022	4.06e-04	2.20e-04	1.9	1.0	0	

Table 2.5.7: test case 03; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: !OVR*	*bias: OVR*	*applied dump - bias: OVR*
0	12000	11724	276	0	n-a
2	24000	23724	276	0	n-a
4	48000	47724	276	0	n-a
6	96000	95724	276	0	n-a

Table 2.5.8: test case 03; applied dump intervals and bias.

```
--------------------------------------------------------------------------------
# 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.
#
# --------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
10000000  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=adj
00000001  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=pri
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000001  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=pri
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
10101010  XSTATUS: X-SE_CLK: 10101010
10101010  YSTATUS: Y-SE_CLK: 10101010
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000100  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=1
00000100  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=1

11111111  DESSR_0_7: 11111111
11111111  DESSR_8_15: 11111111
```

Text 2.5.4: contents of mcb_reg03.txt

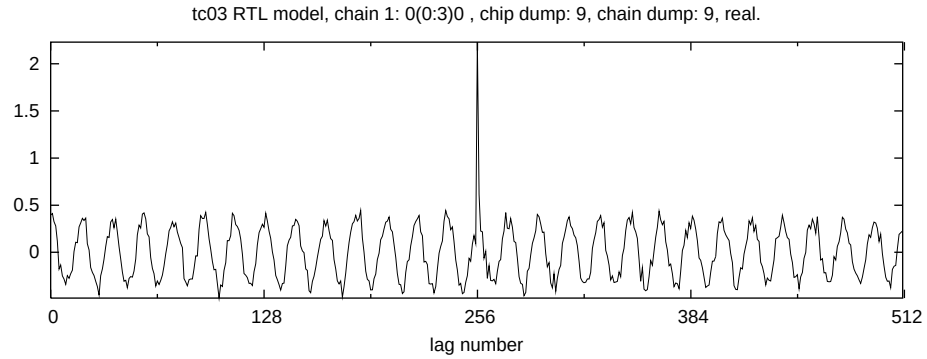

Illustration 2.5.13

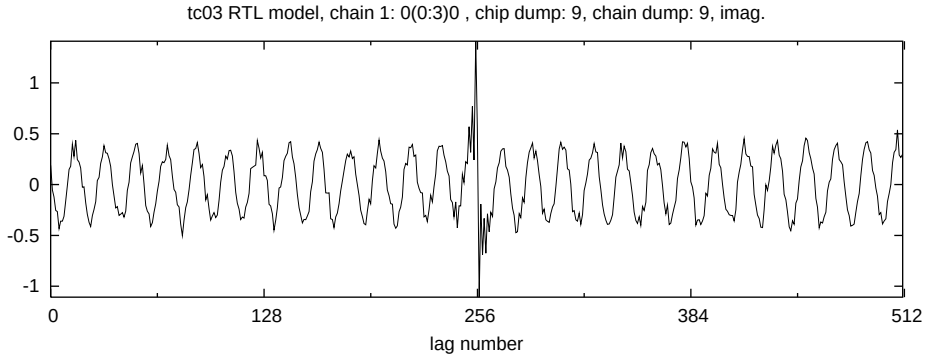

Illustration 2.5.14

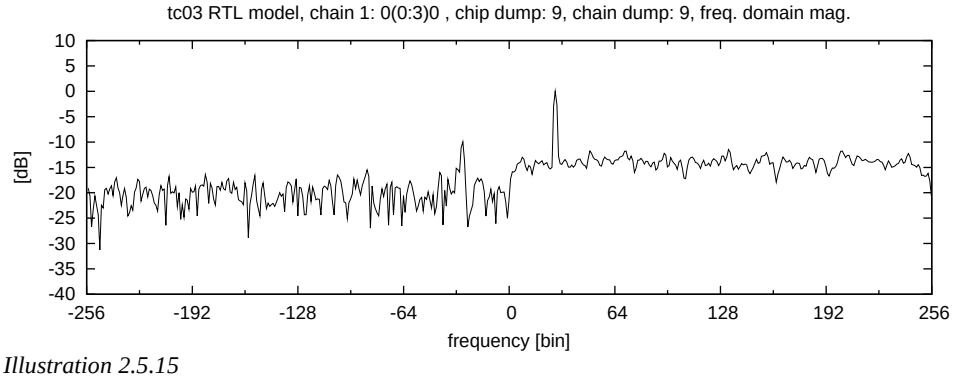

Illustration 2.5.15

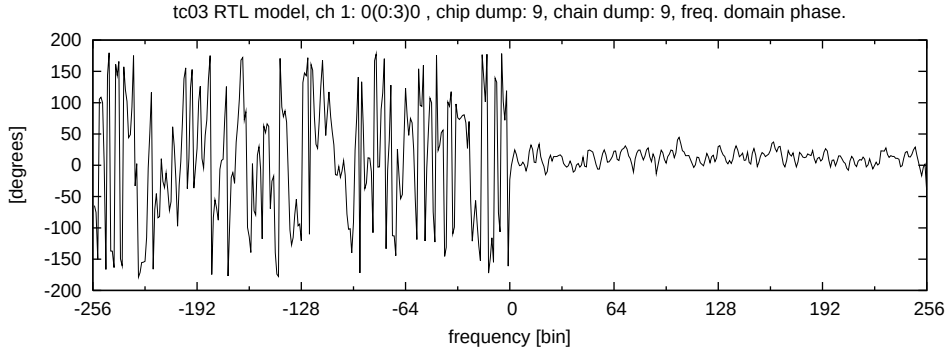

Illustration 2.5.16

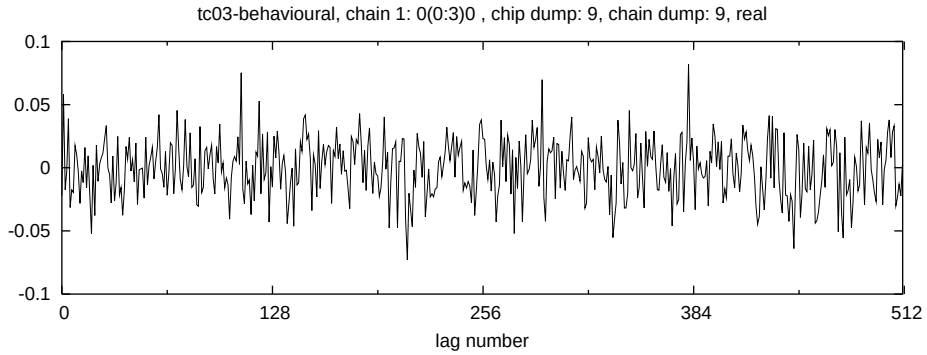

Illustration 2.5.17

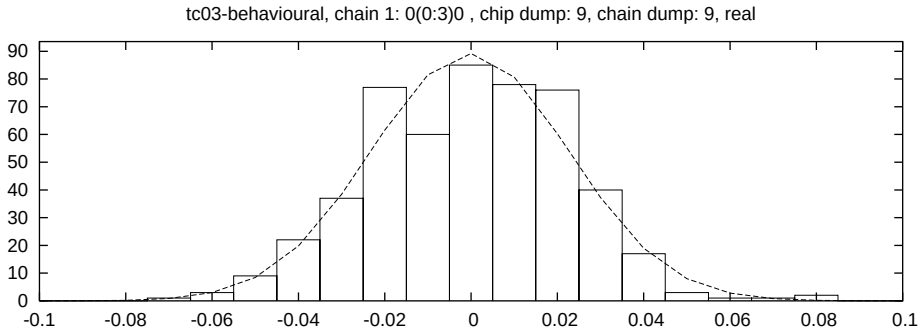

Illustration 2.5.18

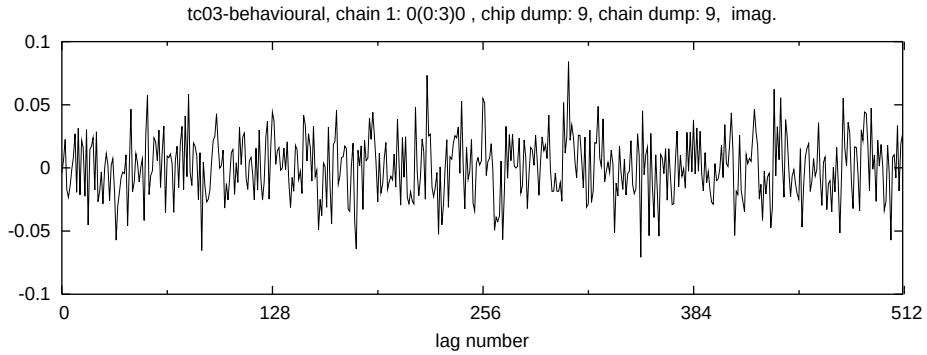

Illustration 2.5.19

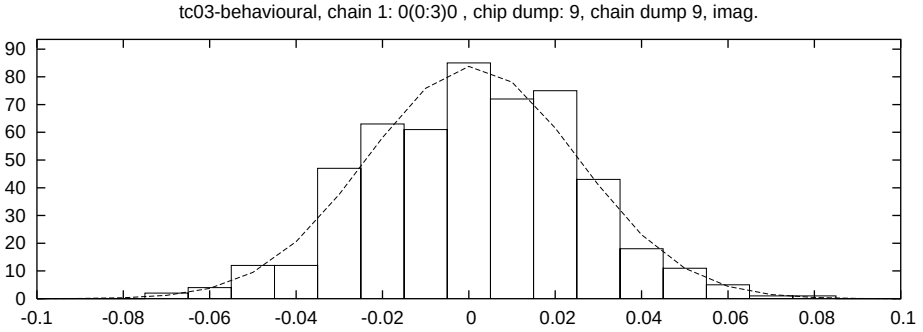

Illustration 2.5.20

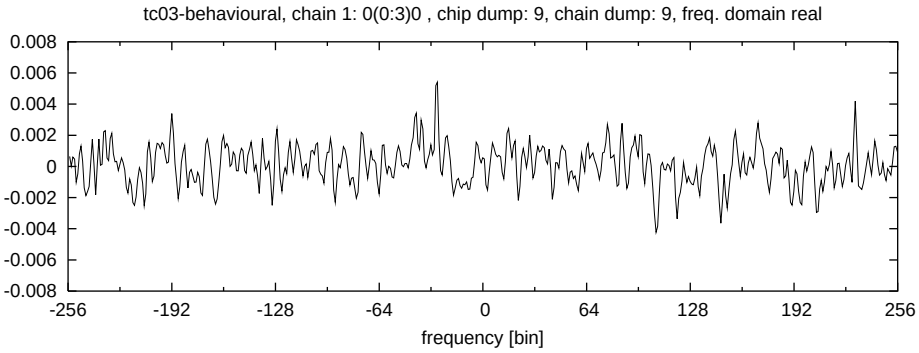

Illustration 2.5.21

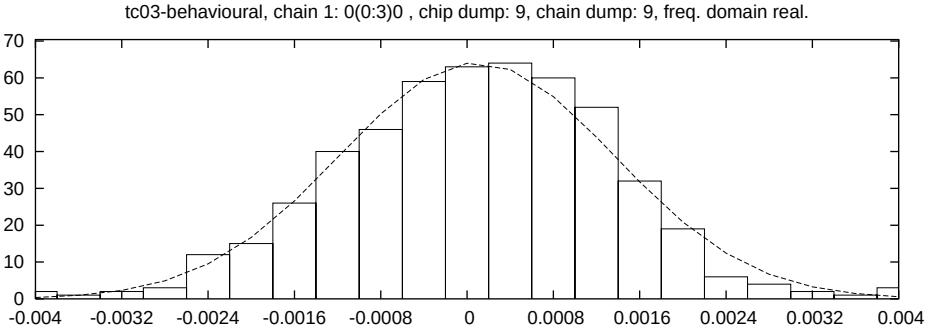

Illustration 2.5.22

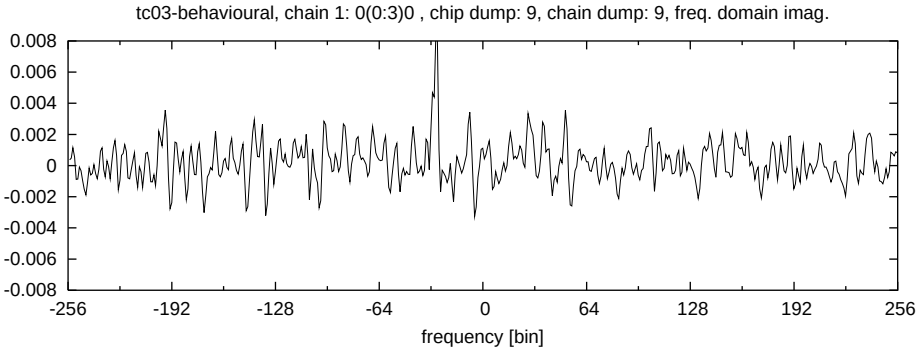

Illustration 2.5.23

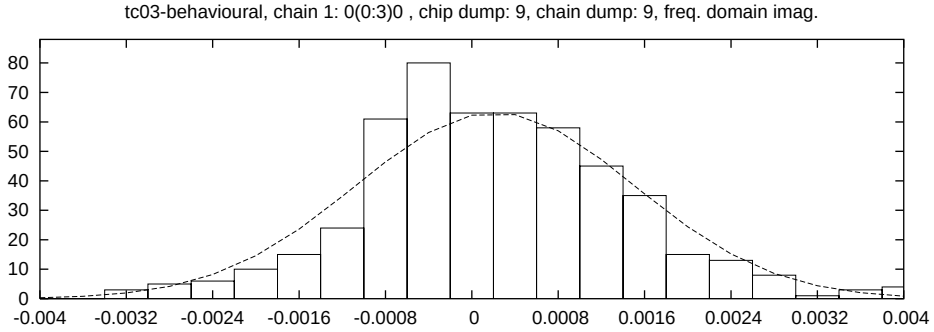

Illustration 2.5.24

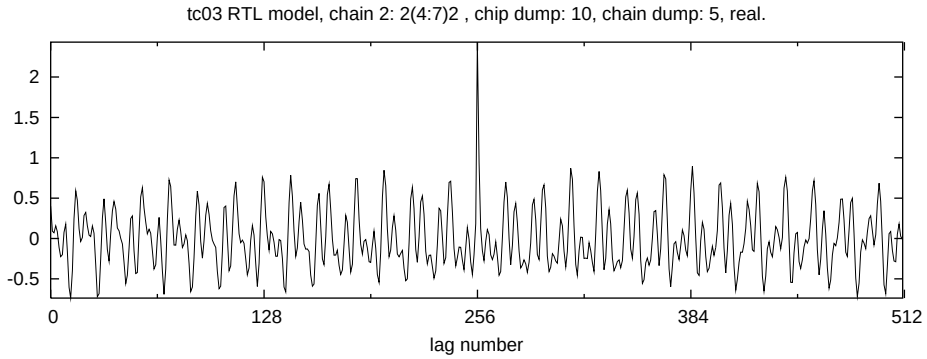

Illustration 2.5.25

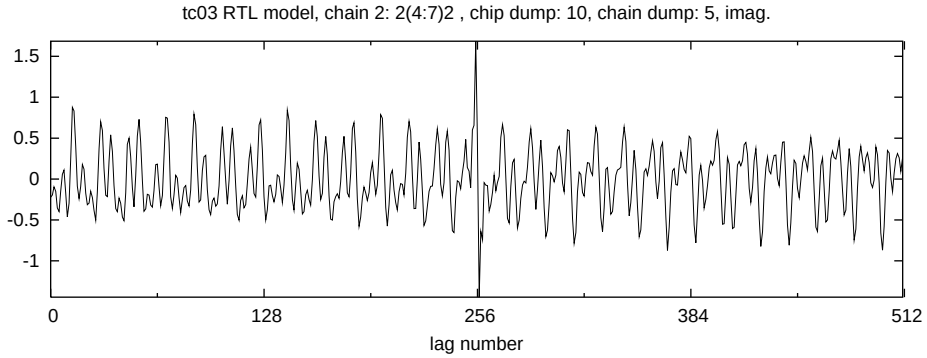

Illustration 2.5.26

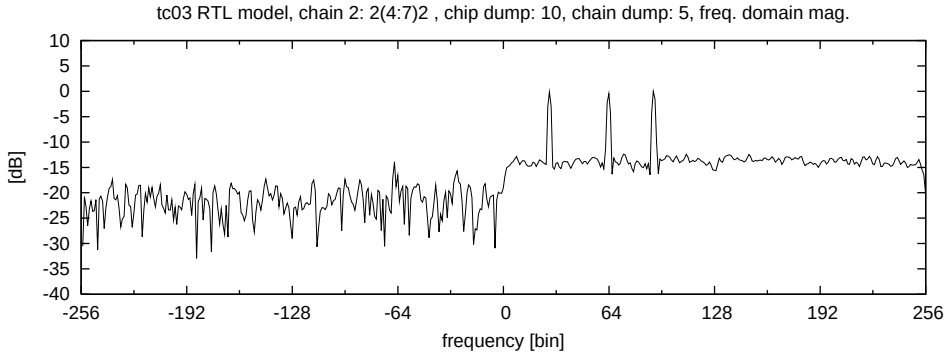

Illustration 2.5.27

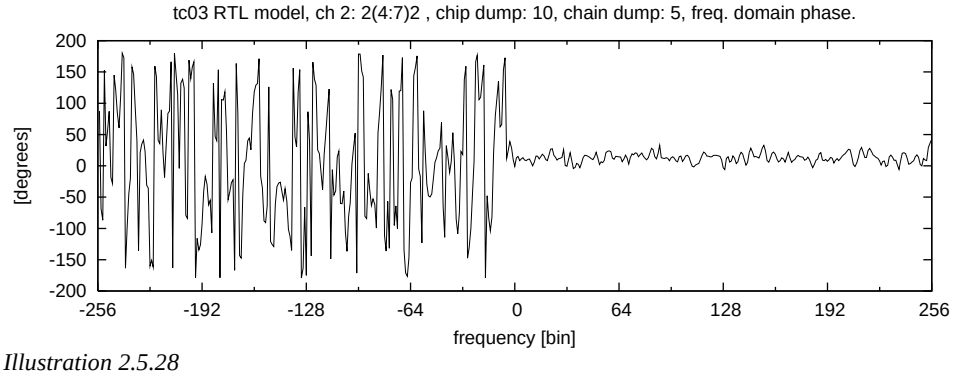

Illustration 2.5.28

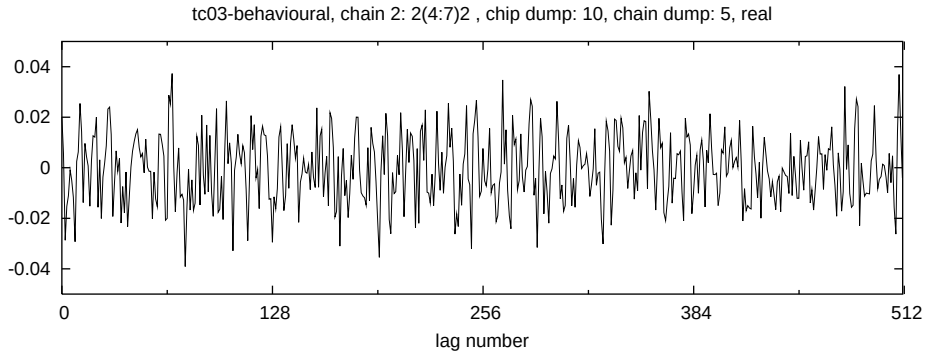

Illustration 2.5.29

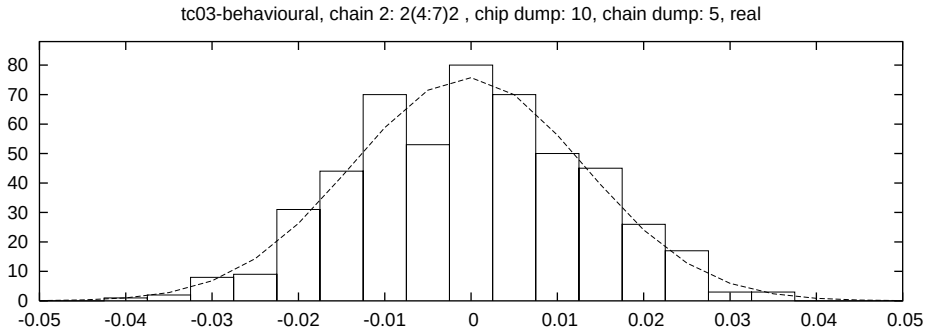

Illustration 2.5.30

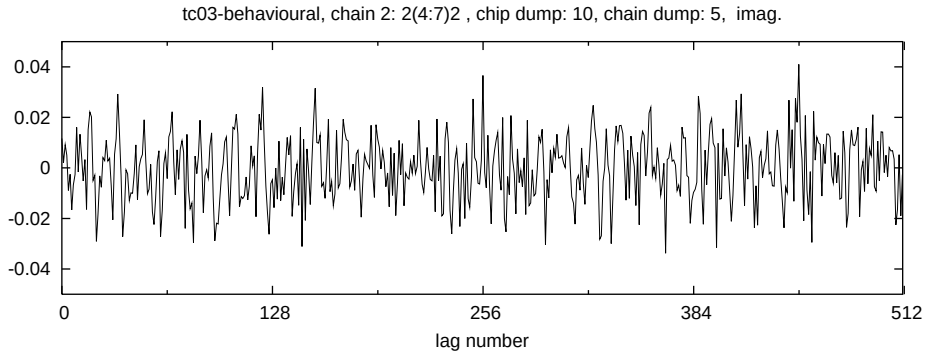

Illustration 2.5.31

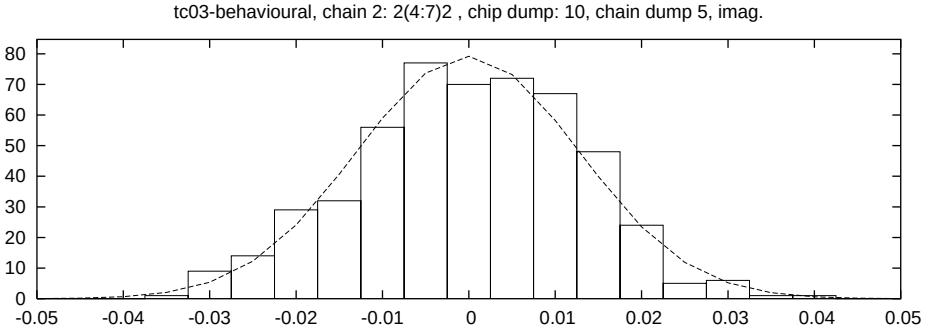

Illustration 2.5.32

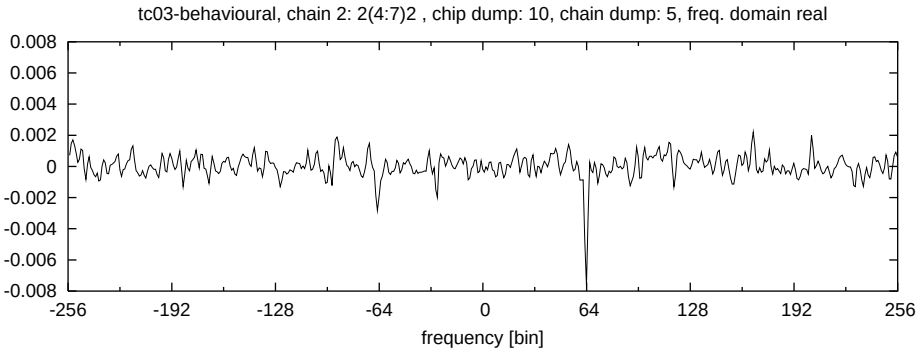

Illustration 2.5.33

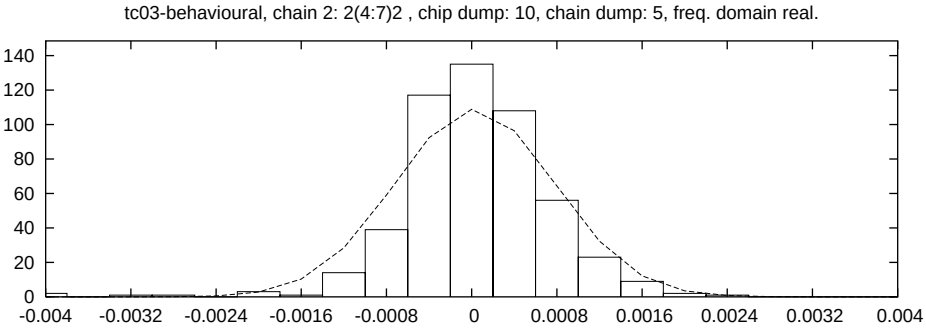

Illustration 2.5.34

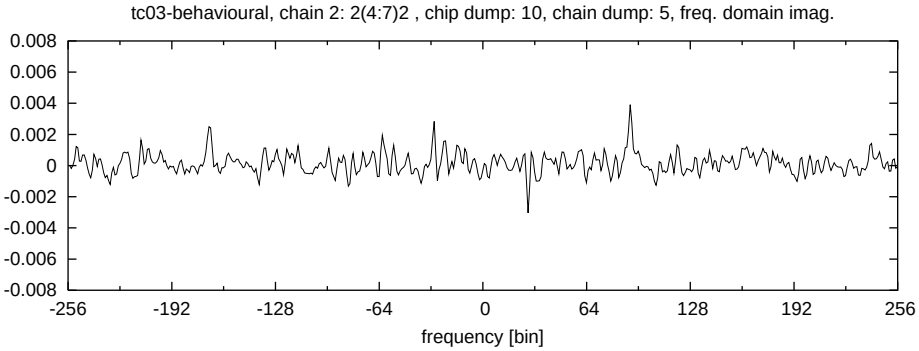

Illustration 2.5.35

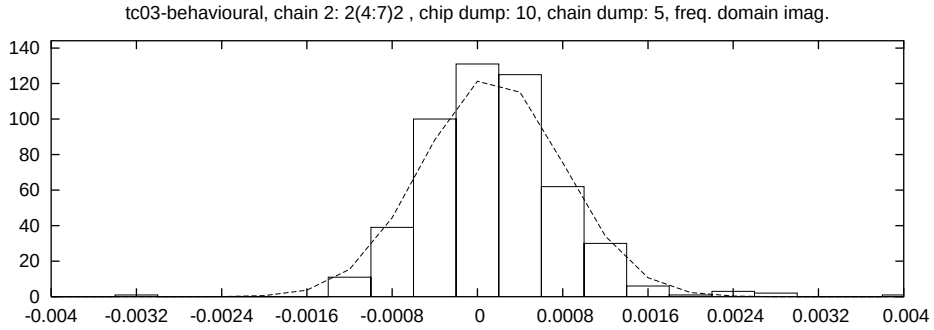

Illustration 2.5.36

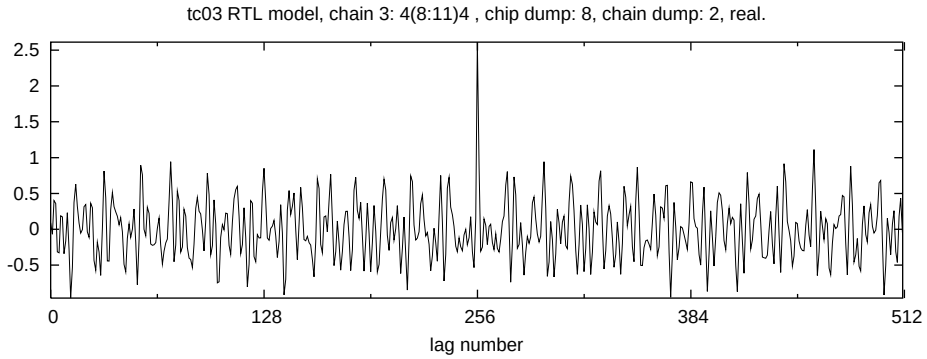

Illustration 2.5.37

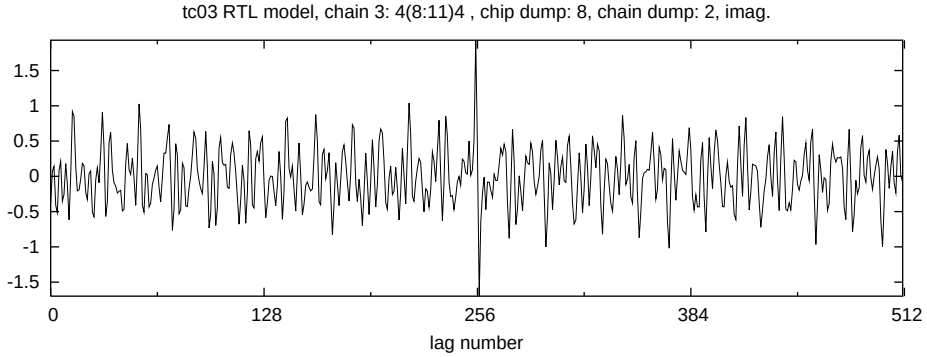

Illustration 2.5.38

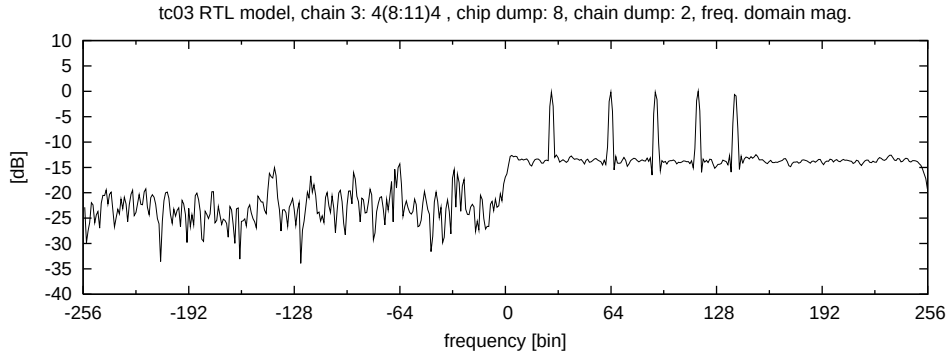

Illustration 2.5.39

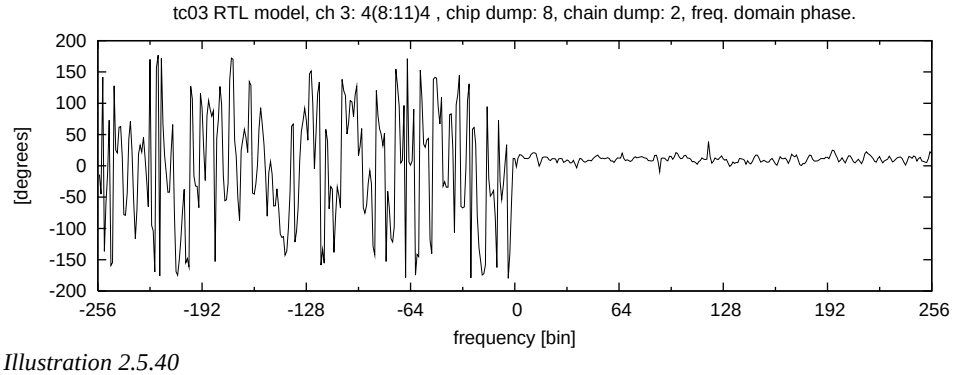

Illustration 2.5.40

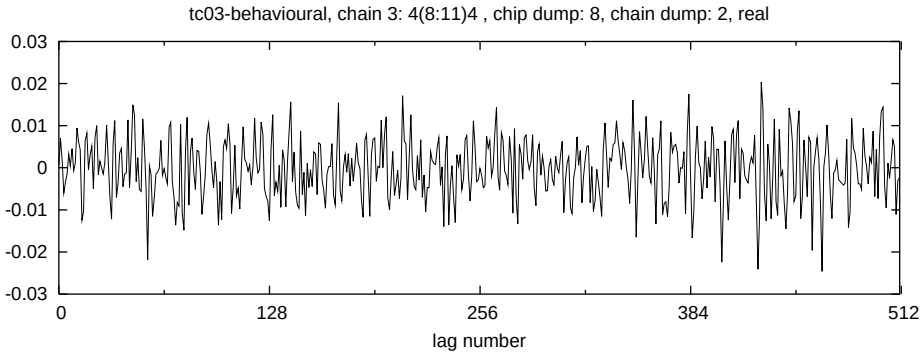

Illustration 2.5.41

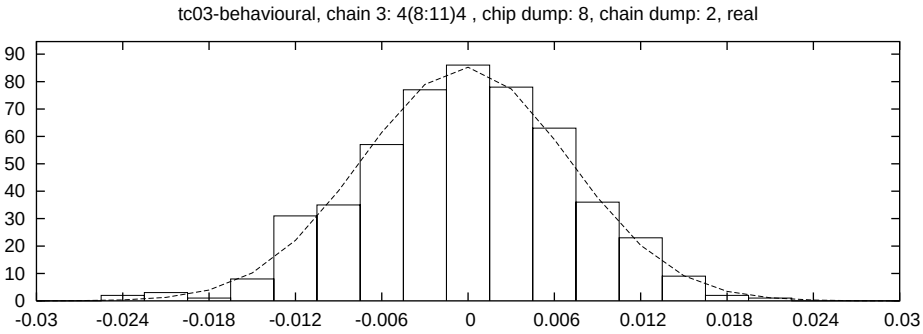

Illustration 2.5.42

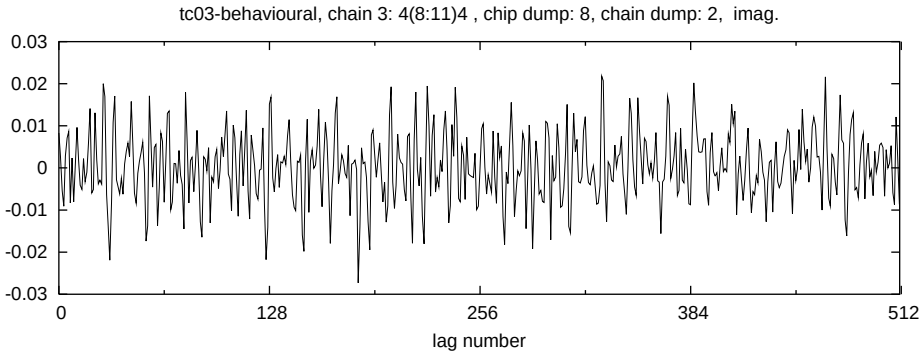

Illustration 2.5.43

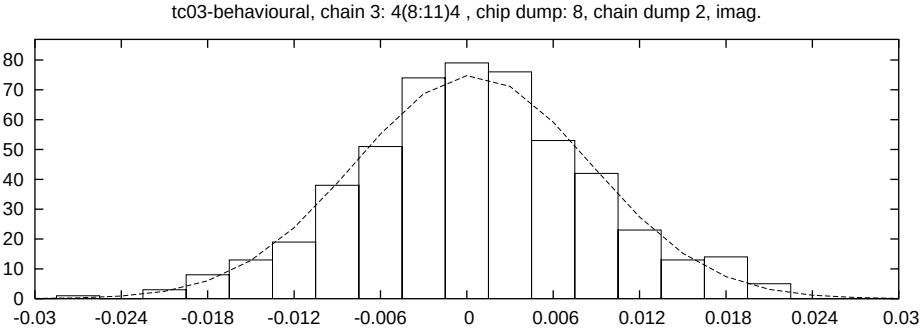

Illustration 2.5.44

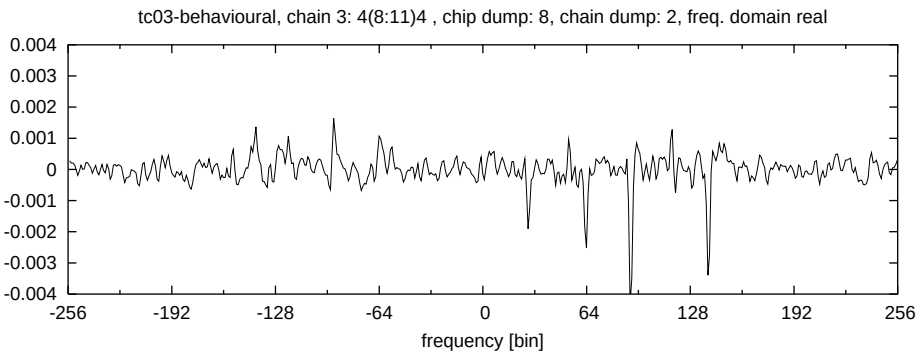

Illustration 2.5.45

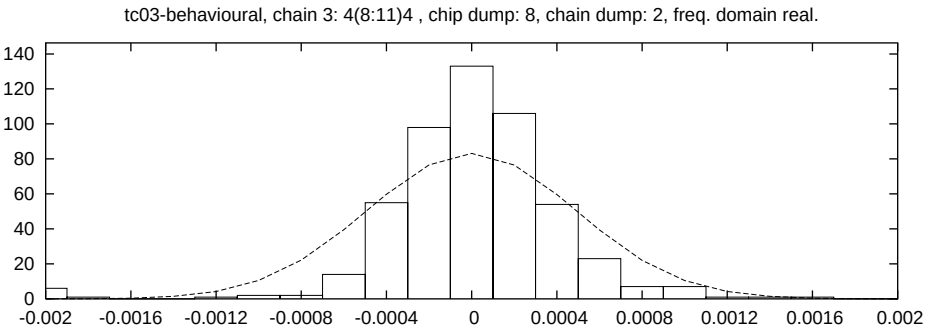

Illustration 2.5.46

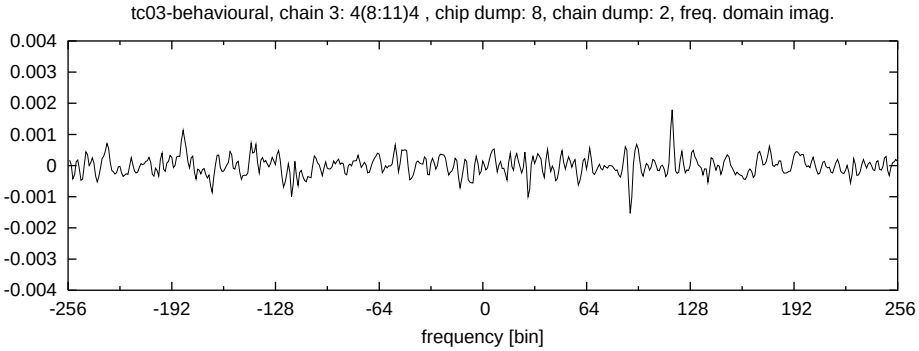

Illustration 2.5.47

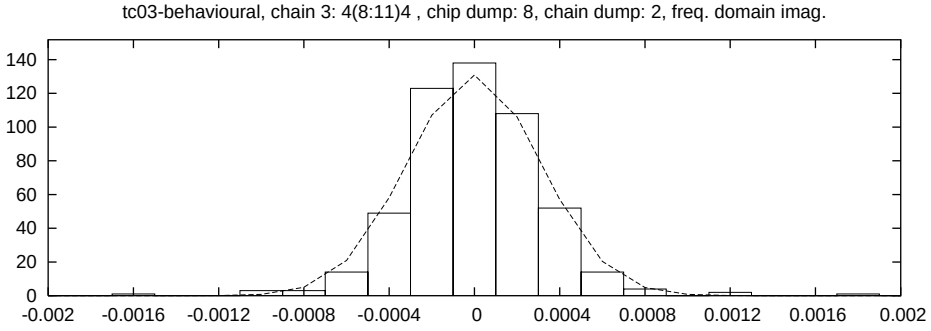

Illustration 2.5.48

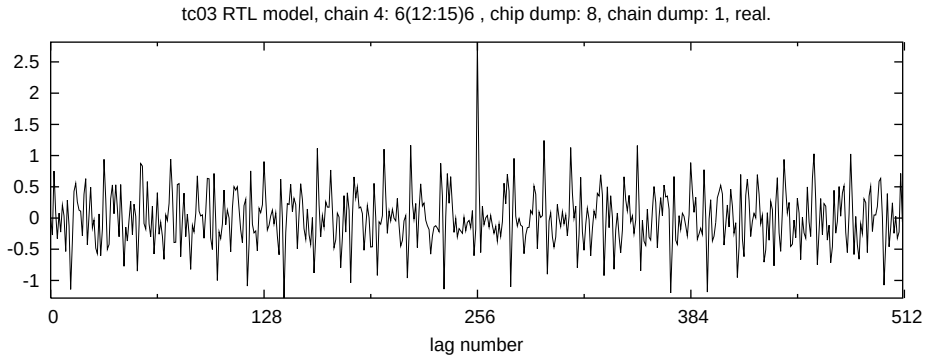

Illustration 2.5.49

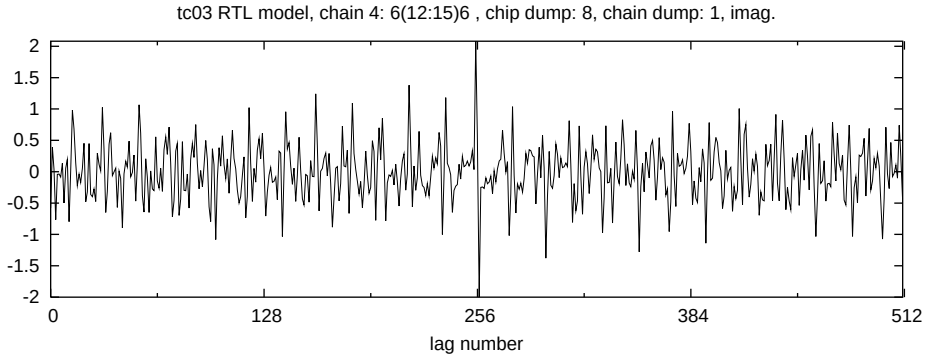

Illustration 2.5.50

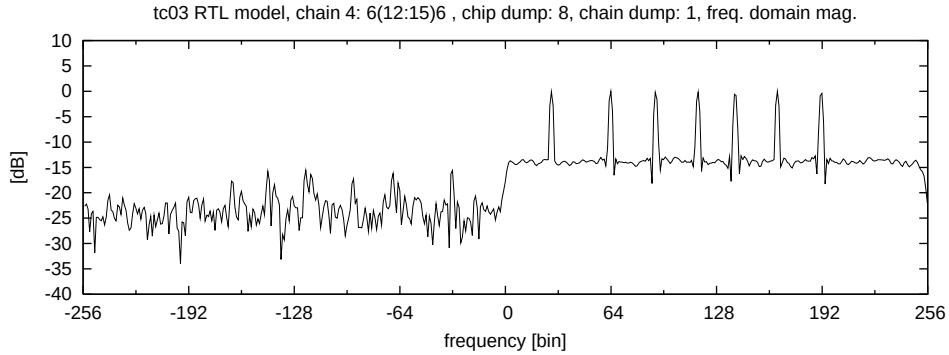

Illustration 2.5.51

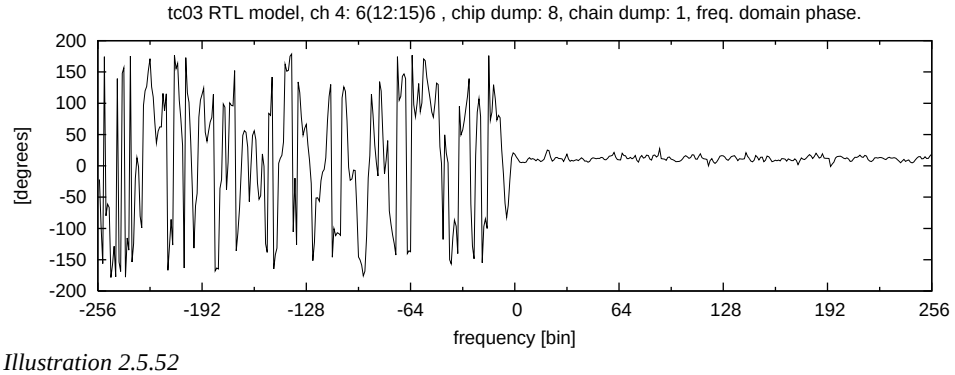

Illustration 2.5.52

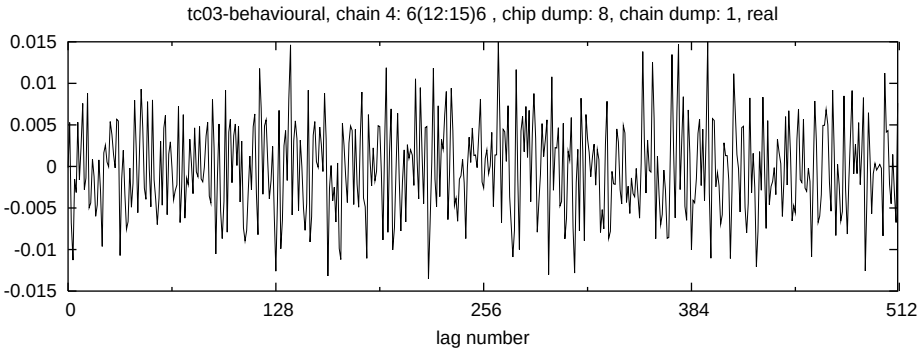

Illustration 2.5.53

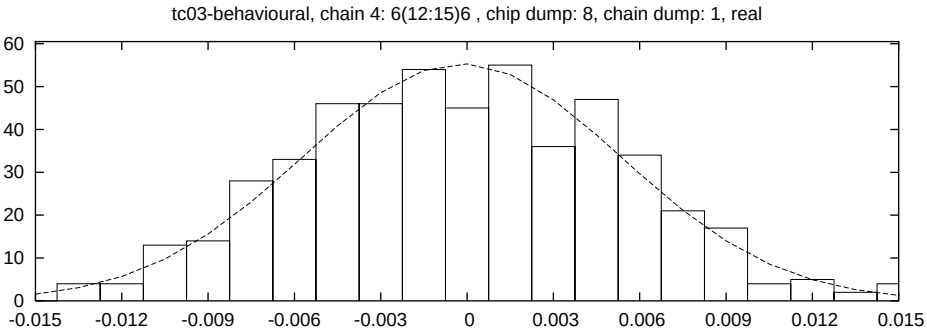

Illustration 2.5.54

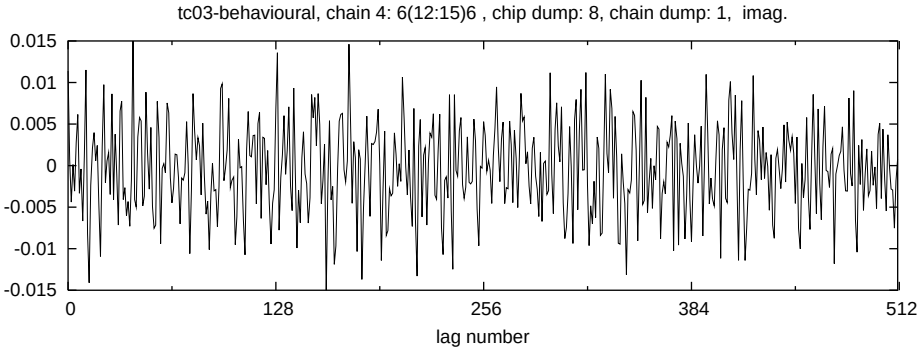

Illustration 2.5.55

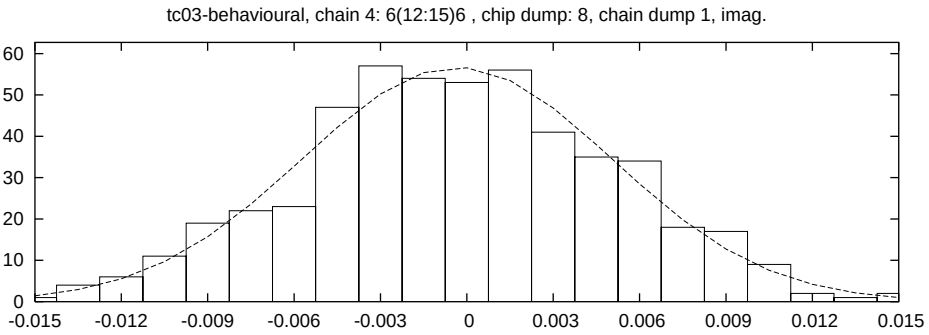

Illustration 2.5.56

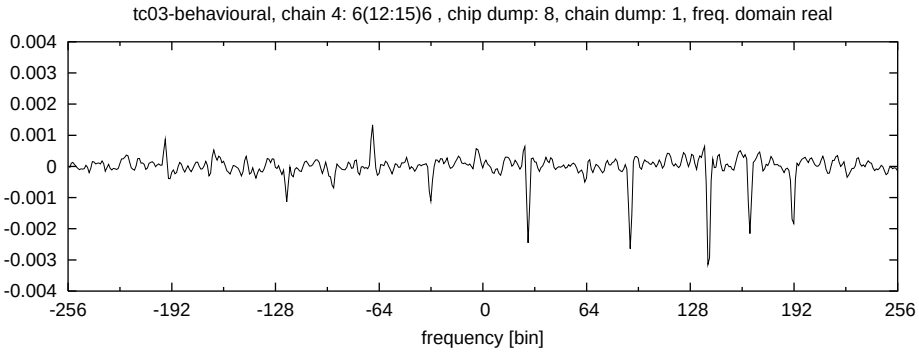

Illustration 2.5.57

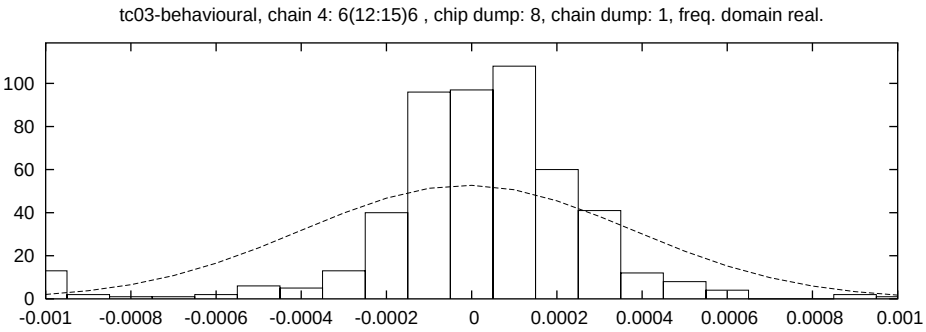

Illustration 2.5.58

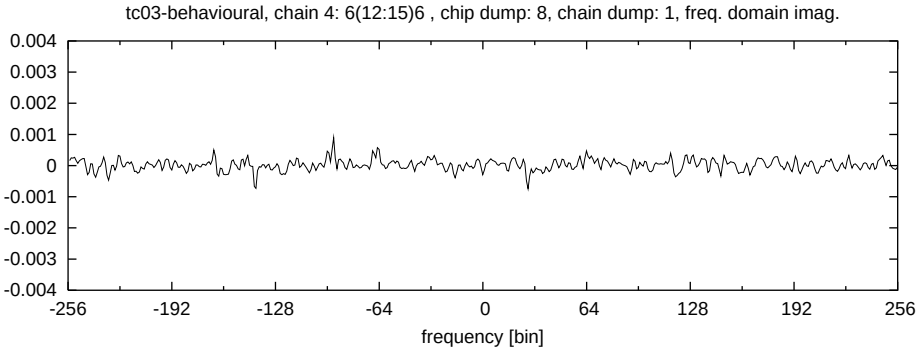

Illustration 2.5.59

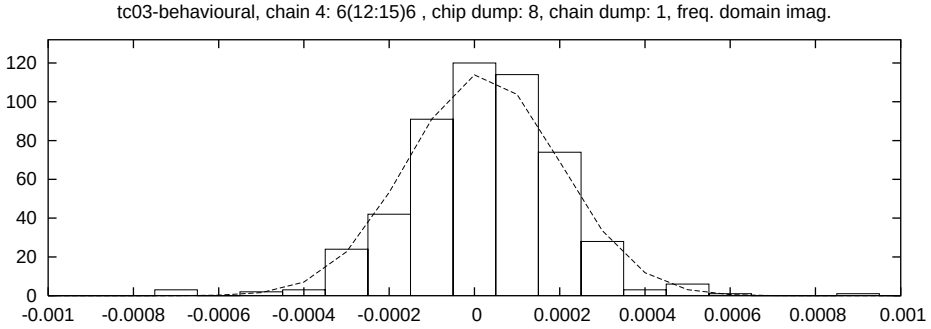

Illustration 2.5.60

2.5.4 test case 04

Synopsis: validation of the fully concatenated chip connected to stream 1 (as opposed to 0 for other fully concatenated test cases) was successful although a high frequency of applied input synchronization errors affected the RTL - behavioural model difference statistics, slightly reducing confidence in the validation. The high error rate corrupted some captured data which in turn produced the warnings reported in the log of *check_register_capture* (Text 2.5.6); these are not contrary to expectations. The extended post simulation script, *tc04_inspect_status_captured*, with output reported in the post simulation log (Text 2.5.5) lists captured data that likely triggered the warnings.

Notes:

(1) A very high peak count in Table 2.5.9 prompts the inclusion of plots in Illustration 2.5.61 to Illustration 2.5.72. This high count may be due to the high frequency of applied input synchronization errors.

(2) see the section on *on line error reporting*, 2.10.3 for additional notes.

```
***--------------------------------------------------------------------------***
tc04_postsim.m : V2.01 run time: Mon Oct 03 18:59:30 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 04.
test case version: v3,05apr21
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 1.
1(0:15)1

Initial (single dump) timestamp (word 0): 4294967295

WARNINGS encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.10: 05-10-02
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc04
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
octave:82> tc04_inspect_status_captured
XSyncerr: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
SID_X: 200 200 200 200 200 200 200 200 200 128 200 200 200 200 200
SBID_X: 1 1 1 1 1 1 1 1 1 28 1 1 1 1 1
BBID_X: 5 5 5 5 5 5 5 5 5 5 0 5 5 5 5 5

YSyncerr: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
SID_Y: 100 100 100 100 100 100 100 100 100 100 64 100 100 100 100 100
SBID_Y: 1 1 1 1 1 1 1 1 1 22 1 1 1 1 1
BBID_Y: 5 5 5 5 5 5 5 5 5 5 0 5 5 5 5 5
```

Text 2.5.5: console log for scripts tc04_postsim; tc04_inspect_status_captured

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 1.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 1.

WARNING from check_register_capture: Captured X-Station baseband
      ID (BBID-X) differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 1.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

WARNING from check_register_capture: status X_SE_CLK
                              differs from template.

WARNING from check_register_capture: status Y_SE_CLK
                              differs from template.
```

Text 2.5.6: tc04; check register and captured data log; WARNINGS noted.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	2.23e-04	2.16e-04	0.2893	0.3447	0.0404	0.0419	1.84	1.91	39	*

Table 2.5.9: test case 04; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	7.85e-05	2.26e-05	0.0205	0.0076	8.74e-04	6.72e-04	1.3	1.0	2	*

Table 2.5.10: test case 04; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: ! OVR*	*bias: OVR*	*applied dump - bias: OVR*
1	20000	19724	276	0	n-a

Table 2.5.11: test case 04; applied dump intervals and bias.

```
# 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.
#
# ------------------------------------------------------------------------------
01000101  MCSR: X/Y_DS=0 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000010  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=sec
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
10000000  CCQR_2_3: Q3: YSw1=sec YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

11111111  XSTATUS: X-SDATA1: 1111, X-SDATA0: 1111
11111111  YSTATUS: Y-SDATA1: 1111, Y-SDATA0: 1111
11111111  XSTATUS: X-SDATA3: 1111, X-SDATA2: 1111
11111111  YSTATUS: Y-SDATA3: 1111, Y-SDATA2: 1111
11111111  XSTATUS: X-SDATA5: 1111, X-SDATA4: 1111
11111111  YSTATUS: Y-SDATA5: 1111, Y-SDATA4: 1111
11111111  XSTATUS: X-SDATA7: 1111, X-SDATA6: 1111
11111111  YSTATUS: Y-SDATA7: 1111, Y-SDATA6: 1111
11111111  XSTATUS: X-PHASE1: 1111, X-PHASE0: 1111
11111111  YSTATUS: Y-PHASE1: 1111, Y-PHASE0: 1111
11111111  XSTATUS: X-PHASE3: 1111, X-PHASE2: 1111
11111111  YSTATUS: Y-PHASE3: 1111, Y-PHASE2: 1111
11111111  XSTATUS: X-PHASE5: 1111, X-PHASE4: 1111
11111111  YSTATUS: Y-PHASE5: 1111, Y-PHASE4: 1111
11111111  XSTATUS: X-PHASE7: 1111, X-PHASE6: 1111
11111111  YSTATUS: Y-PHASE7: 1111, Y-PHASE6: 1111
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
10111111  XSTATUS: X-SE_CLK: 10111111
10111111  YSTATUS: Y-SE_CLK: 10111111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.7: content of file mcb_reg04.txt

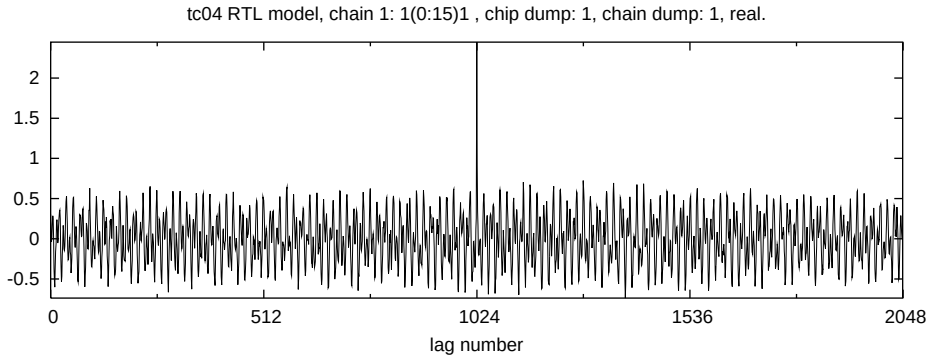

Illustration 2.5.61

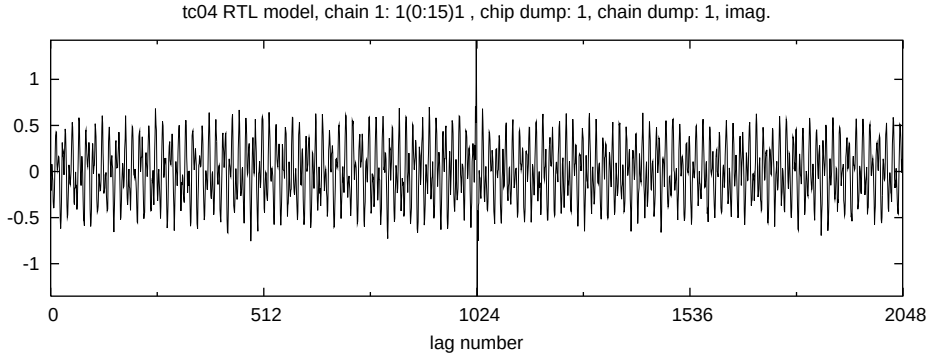

Illustration 2.5.62

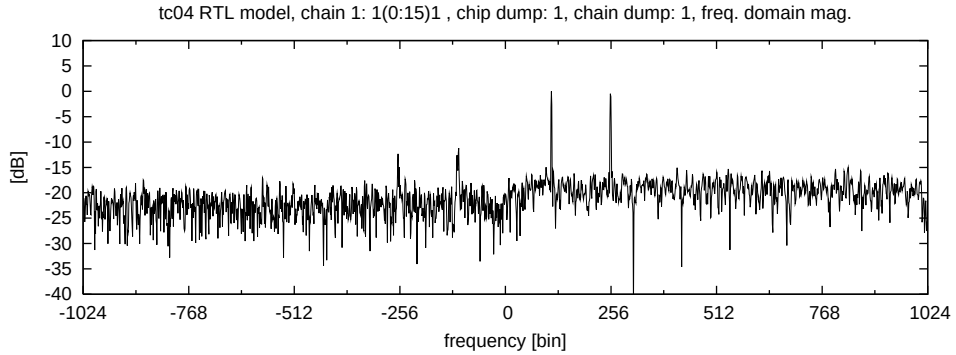

Illustration 2.5.63

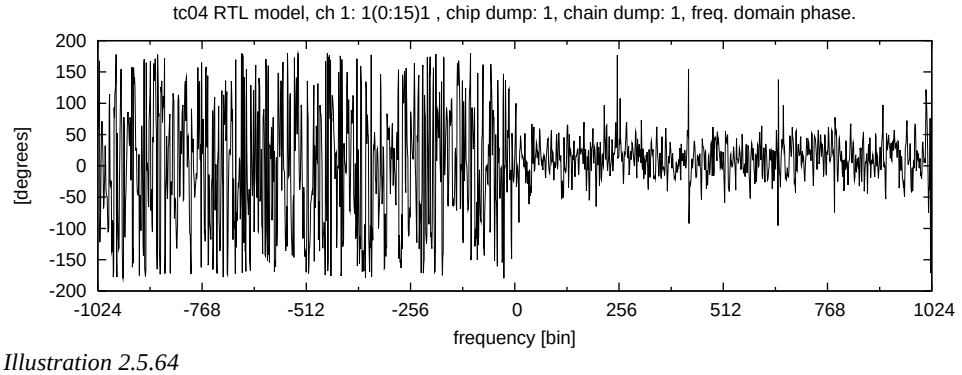

Illustration 2.5.64

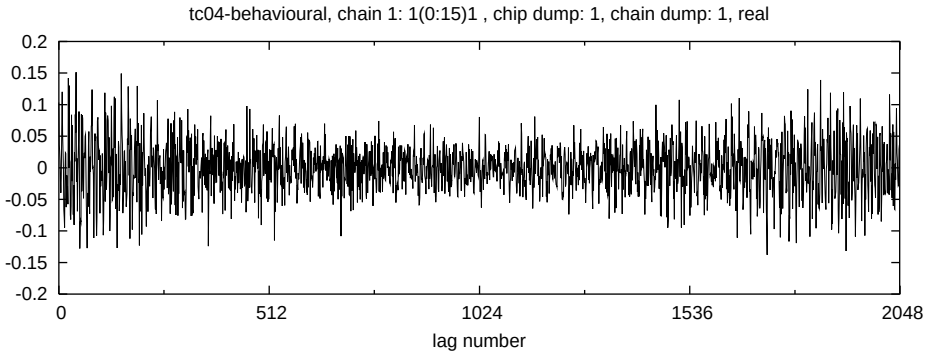

Illustration 2.5.65

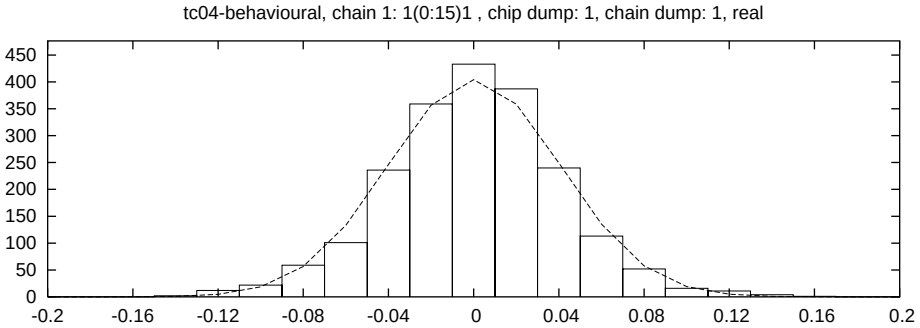

Illustration 2.5.66

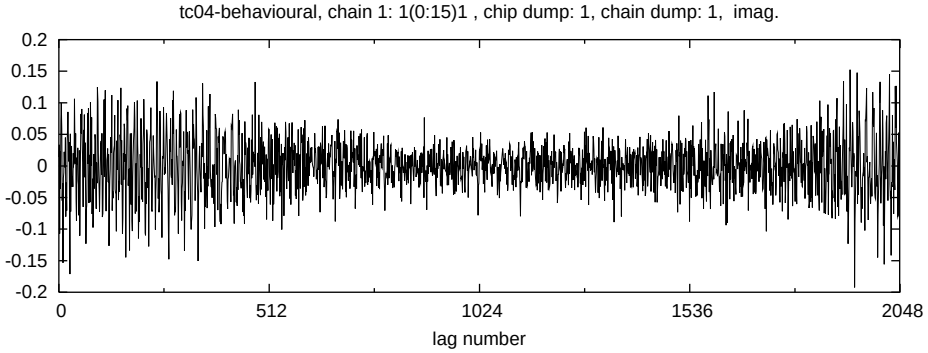

Illustration 2.5.67

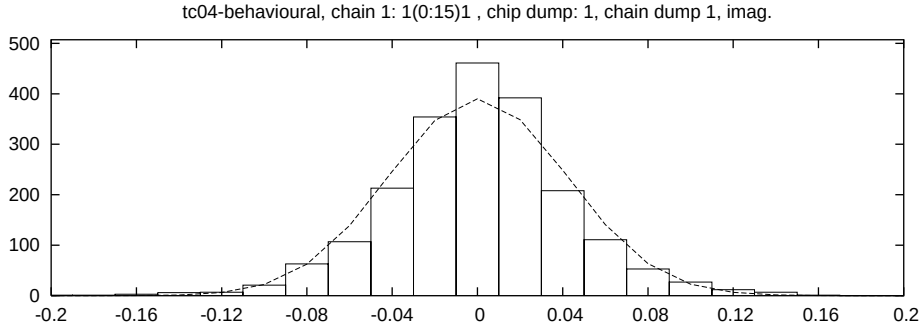

Illustration 2.5.68

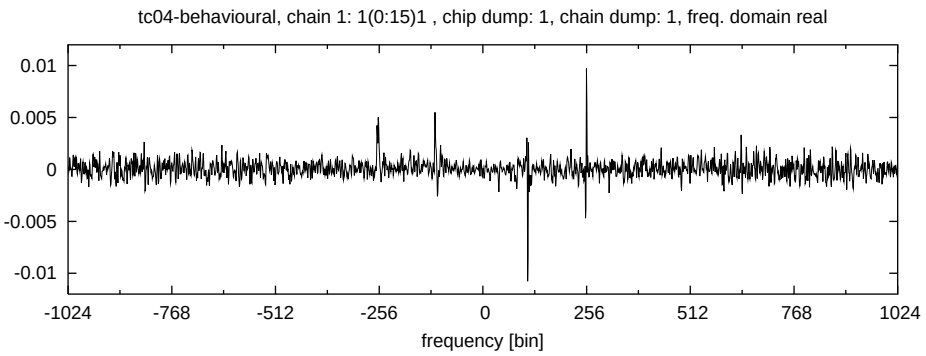

Illustration 2.5.69

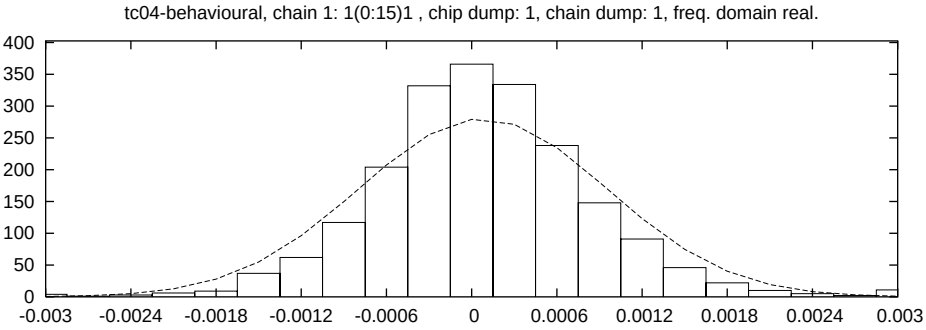

Illustration 2.5.70

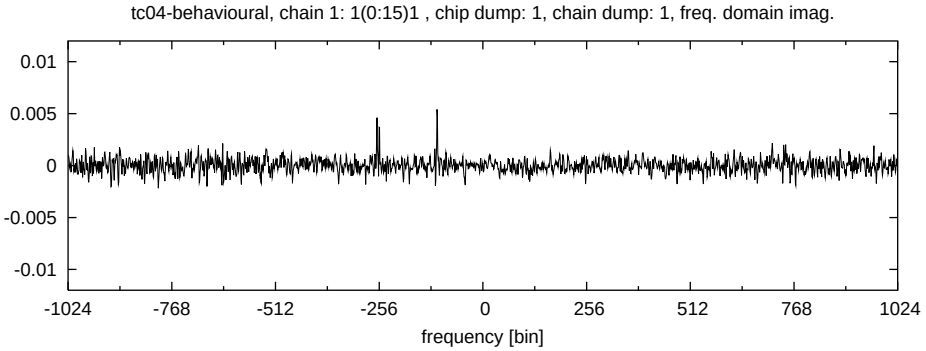

Illustration 2.5.71

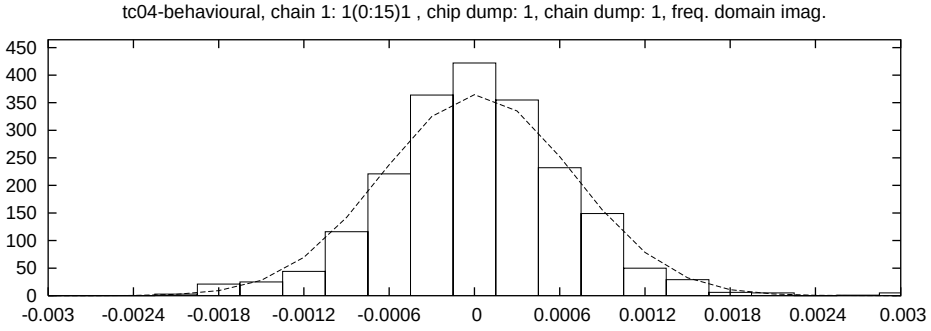

Illustration 2.5.72

2.5.5 test case 05

Synopsis: No extraordinary table entries are noted and no advisories or warnings regarding captured data or register content were logged. All 16 chains (single cell) share a common input data stream and passed a bit exact comparison between chains (logged in Text 2.5.8).

Notes: n-a

```
***--------------------------------------------------------------------------***
tc05_postsim.m : V2.00 run time: Sun Aug 28 15:42:41 2005
Using postsim_lib: 1.40: 05-08-18

48 frames read for test case 05.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
1(0:0)1  1(1:1)1  1(2:2)1  1(3:3)1
1(4:4)1  1(5:5)1  1(6:6)1  1(7:7)1
1(8:8)1  1(9:9)1  1(10:10)1  1(11:11)1
1(12:12)1  1(13:13)1  1(14:14)1  1(15:15)1

WARNING from make_dump_index: Times stamp rollover detected.
 Assume rollover in ordering.
ans = 91

Initial (dump= 1) timestamp (word 0): 4294967295
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc05
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***

No differences exceeding the floating point precision limit of this
machine were found between chains for any of the 3 dumps; all chains generated
identical lag data.
```

Text 2.5.8: console log for scripts tc05_postsim, tc05_chain_similarity_check

chain	dump	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	range $\Re[f-b]$	range $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	peak count	plot
1	1	-1.42e-03	1.90e-03	0.2247	0.1749	0.0348	0.0366	1.74	1.83	1	
1	2	-7.87e-04	-2.91e-03	0.1710	0.1774	0.0338	0.0333	1.69	1.66	0	
1	3	-1.14e-04	4.61e-04	0.1484	0.1587	0.0310	0.0327	1.55	1.63	0	

Table 2.5.12: test case 05; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	dump	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	range $\Re[f-f]$	range $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	co-dump peaks
1	1	1.14e-02	-1.83e-02	0.5666	0.6697	0.1141	0.1210	1.03	1.01	null
1	2	-7.63e-03	1.73e-02	0.5461	0.5956	0.1104	0.1224	0.97	0.98	null

Table 2.5.13: test case 05; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	dump	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	range $\Re[f^F-b^F]$	range $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	peak count	plot
1	1	2.37e-03	-1.16e-03	0.0213	0.0134	4.19e-03	3.09e-03	7.0	5.2	0	
1	2	-1.04e-03	1.41e-03	0.0224	0.0139	3.74e-03	2.80e-03	6.2	4.7	0	
1	3	3.73e-04	-8.65e-04	0.0166	0.0180	3.45e-03	3.60e-03	5.7	6.0	0	

Table 2.5.14: test case 05; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	applied dump	bias:!OVR	applied dump - bias: ! OVR	bias: OVR	applied dump - bias: OVR
1	12000	11724	276	0	n-a

Table 2.5.15: test case 05; applied dump intervals and bias.

```
# ------------------------------------------------------------------------------
# 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.
#
# ------------------------------------------------------------------------------
00000101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

01100110  CCCSCR_0_1:   CCC1: Y=pri X=sec   CCC0: Y=pri X=sec
01100110  CCCSCR_2_3:   CCC3: Y=pri X=sec   CCC2: Y=pri X=sec
01100110  CCCSCR_4_5:   CCC5: Y=pri X=sec   CCC4: Y=pri X=sec
01100110  CCCSCR_6_7:   CCC7: Y=pri X=sec   CCC6: Y=pri X=sec
01100110  CCCSCR_8_9:   CCC9: Y=pri X=sec   CCC8: Y=pri X=sec
01100110  CCCSCR_10_11: CCC11: Y=pri X=sec CCC10: Y=pri X=sec
01100110  CCCSCR_12_13: CCC13: Y=pri X=sec CCC12: Y=pri X=sec
01100110  CCCSCR_14_15: CCC15: Y=pri X=sec CCC14: Y=pri X=sec

10100000  CCQR_0_1: Q1: YSw1=sec YSw0=pri XSw1=sec XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
10101010  CCQR_2_3: Q3: YSw1=sec YSw0=pri XSw1=sec XSw0=pri  Q2: YSw1=sec YSw0=pri XSw1=sec XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111101  XSTATUS: X-SE_CLK: 11111101
11111101  YSTATUS: Y-SE_CLK: 11111101
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000100  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=1
00000100  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=1

11111111  DESSR_0_7: 11111111
11111111  DESSR_8_15: 11111111
```

Text 2.5.9: contents of file mcb_reg05.txt

2.5.6 test case 06

Synopsis: No extraordinary plot features were noted although warnings regarding consistency of captured data were logged as a result of applied input synchronization errors. Data captured from the *X* input stream that differs between cells common to a chain and triggered these warnings are tabulated in Table 2.5.16.

Notes: A single advisory in the log of *check_register_capture* (Text 2.5.11) was generated as a consequence of the **X|YSyncerr** bits of output frames differing between cells common to a chain. This behaviour is not specified within the RFS document and is not considered an issue.

```
***--------------------------------------------------------------------------***
tc06_postsim.m : V2.00 run time: Sun Aug 28 16:29:00 2005
Using postsim_lib: 1.40: 05-08-18

120 frames read for test case 06.
test case version: v3,05jul24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
1(0:3)1  3(4:7)3  5(8:11)5  7(12:15)7

WARNING from make_dump_index: Times stamp rollover detected.
 Assume rollover in ordering.
ans = 91

Initial (dump= 1) timestamp (word 0): 4294967295
Timestamps are sequential and increment by 1.

WARNINGS encountered during register status and data capture checks;
                        details in log file.

ADVISORIES encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc06
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
```

Text 2.5.10: console log for script tc06_postsim

chain	*dump*	*XSyncerr*	*SID_X*	*SBID_X*	*BBID_X*
1	11	1 1 1 1	128 200 200 200	28 1 1 1	0 7 7 7
1	12	1 0 0 0	200 200 200 200	1 1 1 1	7 7 7 7
3	16	1 1 1 1	128 200 200 200	28 5 5 5	2 3 3 3

Table 2.5.16: Test case 06; X status bits and X captured data which differ between cells of a chain. Obtained by executing script , tc06_inspect_status_captured.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

ADVISORY from check_register_capture: Synch. error status bit
           (X|YSyncer) settings in an output frame are not identical
            between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 1.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 1.

WARNING from check_register_capture: Captured X-Station baseband
      ID (BBID-X) differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 1.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      of chain 1 differs between dumps.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      of chain 1 differs between dumps.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) of chain 1 differs between dumps.

WARNING from check_register_capture: Captured Y-Station subband
      ID (SBID-Y) of chain 1 differs between dumps.

WARNING from check_register_capture: X-Station baseband
      ID (BBID-X) of chain 1 differs between dumps.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) of chain 1 differs between dumps.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      differs between cells of chain 3.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 3.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) differs between cells of chain 3.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 3.
```

Text 2.5.11: test case 06; register and captured data log, part 1

```
WARNING from check_register_capture: Captured X-Station baseband
      ID (BBID-X) differs between cells of chain 3.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 3.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      of chain 3 differs between dumps.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      of chain 3 differs between dumps.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) of chain 3 differs between dumps.

WARNING from check_register_capture: Captured Y-Station subband
      ID (SBID-Y) of chain 3 differs between dumps.

WARNING from check_register_capture: X-Station baseband
      ID (BBID-X) of chain 3 differs between dumps.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) of chain 3 differs between dumps.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                            template of expected states.
```

Text 2.5.12: test case 06; register and captured data log, part 2

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	2.17e-04	-1.19e-04	0.2189	0.2279	0.0306	0.0361	1.08	1.28	3	
1	2	-1.16e-03	1.83e-04	0.1972	0.2007	0.0338	0.0356	1.19	1.26	0	
1	3	1.82e-04	-2.67e-04	0.2205	0.2204	0.0339	0.0353	1.20	1.25	3	
1	4	1.04e-03	-5.08e-04	0.2257	0.2340	0.0364	0.0347	1.29	1.23	5	
1	5	6.83e-04	-1.38e-03	0.1839	0.1902	0.0278	0.0312	0.98	1.10	4	
1	6	5.04e-04	-6.99e-04	0.2711	0.1865	0.0395	0.0341	1.40	1.21	2	
1	7	3.61e-04	9.48e-04	0.1956	0.1956	0.0330	0.0340	1.17	1.20	3	
1	8	-2.10e-03	-7.82e-04	0.2130	0.1884	0.0342	0.0315	1.21	1.11	3	
1	9	1.46e-03	-7.72e-05	0.1927	0.1897	0.0332	0.0345	1.17	1.22	1	*
1	10	-4.62e-04	3.14e-04	0.2343	0.2027	0.0356	0.0347	1.26	1.23	5	
1	11	3.49e-04	-3.92e-04	0.1794	0.2031	0.0314	0.0347	1.11	1.23	0	
1	12	-6.59e-04	6.72e-05	0.2227	0.2220	0.0369	0.0360	1.31	1.27	3	
1	13	1.80e-03	-2.48e-03	0.2198	0.2268	0.0351	0.0320	1.24	1.13	3	
1	14	-9.39e-04	1.74e-03	0.2197	0.2005	0.0358	0.0372	1.27	1.32	2	
1	15	6.98e-04	2.01e-04	0.2193	0.2354	0.0383	0.0428	1.35	1.51	0	
1	16	8.77e-04	-7.03e-05	0.2272	0.2046	0.0344	0.0345	1.22	1.22	3	
2	2	-4.09e-05	-4.52e-04	0.0985	0.1032	0.0165	0.0178	0.83	0.89	3	
2	4	-9.80e-04	4.40e-04	0.1084	0.1533	0.0197	0.0224	0.99	1.12	3	*

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
2	6	3.44e-04	-4.54e-04	0.1269	0.1363	0.0206	0.0211	1.03	1.06	5	
2	8	-9.11e-04	-2.53e-04	0.1043	0.0927	0.0180	0.0166	0.90	0.83	1	
2	10	8.44e-04	5.34e-04	0.1001	0.1026	0.0183	0.0192	0.92	0.96	0	
2	12	-1.22e-04	-6.98e-04	0.1225	0.1156	0.0179	0.0192	0.89	0.96	4	*
2	14	8.87e-04	2.94e-04	0.1001	0.0928	0.0159	0.0156	0.79	0.78	3	
2	16	-7.31e-04	-1.65e-03	0.1049	0.1110	0.0171	0.0172	0.86	0.86	5	
3	4	1.02e-04	4.87e-04	0.0623	0.0667	0.0102	0.0105	0.72	0.74	2	
3	8	5.06e-04	1.81e-05	0.0786	0.0653	0.0116	0.0112	0.82	0.79	3	
3	12	-1.09e-04	-3.20e-05	0.0821	0.0973	0.0137	0.0142	0.97	1.01	4	
3	16	1.16e-03	-4.24e-04	0.0598	0.0673	0.0104	0.0103	0.74	0.73	2	
4	8	-9.90e-05	1.89e-04	0.0469	0.0454	0.0073	0.0075	0.73	0.75	4	*
4	16	4.83e-04	-1.30e-04	0.0603	0.0525	0.0095	0.0098	0.95	0.98	3	

Table 2.5.17: test case 06; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-9.65e-04	-7.75e-03	0.7864	0.7097	0.1228	0.1284	0.99	1.06	256
1	2	-3.37e-03	6.39e-03	0.6820	0.7316	0.1126	0.1252	0.96	1.07	null
1	3	3.37e-04	-1.49e-03	0.6859	0.6257	0.1234	0.1193	0.90	0.97	256 384
1	4	1.61e-03	-2.74e-03	0.6797	0.6096	0.1118	0.1078	0.93	0.96	488
1	5	1.07e-02	1.47e-03	0.6860	0.7553	0.1201	0.1176	1.04	1.01	null
1	6	-8.25e-03	6.47e-03	0.8018	0.8415	0.1304	0.1366	1.08	1.07	null
1	7	-5.05e-03	-4.39e-03	0.7462	0.9067	0.1186	0.1327	1.02	1.08	null
1	8	6.23e-04	-1.21e-03	0.7682	0.6638	0.1142	0.1148	1.04	1.04	null
1	9	1.57e-03	2.10e-03	0.8091	0.6816	0.1281	0.1110	0.99	0.94	null
1	10	3.33e-04	-3.93e-03	0.6925	0.8029	0.1171	0.1223	0.97	0.99	null
1	11	-3.30e-03	9.38e-04	0.8324	0.6549	0.1258	0.1208	1.05	1.05	null
1	12	8.21e-03	-8.33e-04	0.8774	0.5784	0.1164	0.1152	1.02	1.00	null
1	13	-3.51e-03	3.10e-03	0.5725	0.6425	0.1122	0.1179	1.00	1.04	null
1	14	3.94e-03	-8.06e-04	0.6525	0.8285	0.1144	0.1394	1.00	1.19	null
1	15	-6.31e-03	-2.25e-03	0.7490	0.6063	0.1228	0.1202	1.03	1.03	null
2	2	-1.51e-04	1.29e-03	0.3802	0.4751	0.0715	0.0762	0.96	0.95	null
2	4	2.22e-03	9.05e-04	0.5043	0.4951	0.0820	0.0844	0.96	1.01	null
2	6	-1.59e-03	-2.17e-03	0.4716	0.4857	0.0723	0.0790	1.01	1.03	null
2	8	-5.25e-04	2.66e-03	0.5305	0.4497	0.0816	0.0771	1.03	1.05	null
2	10	7.97e-04	-3.25e-03	0.4464	0.4741	0.0858	0.0851	1.04	1.06	null
2	12	-1.02e-03	-3.68e-04	0.4921	0.4806	0.0886	0.0785	1.02	1.02	null
2	14	1.65e-03	1.02e-03	0.4293	0.4797	0.0802	0.0807	1.01	1.02	null
3	4	2.28e-03	2.74e-03	0.3108	0.2905	0.0512	0.0478	0.99	1.00	null
3	8	-2.80e-04	-1.28e-03	0.3195	0.3023	0.0502	0.0505	0.97	0.98	null
3	12	5.59e-04	-1.39e-03	0.3292	0.3044	0.0532	0.0513	0.98	0.98	null
4	8	-3.90e-04	-9.98e-05	0.2207	0.2423	0.0374	0.0386	0.99	0.98	null

Table 2.5.18: test case 06; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	4.99e-04	1.49e-05	0.0249	0.0134	2.01e-03	1.60e-03	2.4	1.9	0	
1	2	-3.78e-04	-3.66e-05	0.0224	0.0197	1.95e-03	1.79e-03	2.3	2.1	0	
1	3	4.86e-04	1.63e-04	0.0168	0.0192	1.87e-03	1.93e-03	2.2	2.3	0	
1	4	-2.23e-04	1.63e-05	0.0211	0.0183	2.01e-03	1.74e-03	2.4	2.0	0	
1	5	-1.89e-04	-1.51e-04	0.0099	0.0113	1.59e-03	1.50e-03	1.9	1.8	0	
1	6	2.55e-04	-1.12e-05	0.0229	0.0120	2.13e-03	1.51e-03	2.5	1.8	0	
1	7	-2.88e-04	7.63e-05	0.0228	0.0106	2.10e-03	1.57e-03	2.5	1.8	0	
1	8	2.68e-04	-2.84e-04	0.0171	0.0160	1.85e-03	1.67e-03	2.2	2.0	0	
1	9	2.07e-05	1.87e-04	0.0120	0.0208	1.71e-03	1.96e-03	2.0	2.3	0	
1	10	-9.17e-05	-6.60e-05	0.0254	0.0129	2.11e-03	1.74e-03	2.5	2.1	0	*
1	11	8.61e-05	-6.59e-05	0.0228	0.0118	1.98e-03	1.60e-03	2.3	1.9	0	
1	12	-9.47e-05	1.72e-04	0.0249	0.0120	2.04e-03	1.68e-03	2.4	2.0	0	
1	13	1.76e-04	-4.43e-04	0.0175	0.0182	1.90e-03	1.80e-03	2.2	2.1	0	
1	14	3.37e-05	1.82e-04	0.0270	0.0147	2.12e-03	1.82e-03	2.5	2.1	0	
1	15	-3.16e-04	-7.01e-06	0.0452	0.0170	2.88e-03	1.65e-03	3.4	1.9	1	
1	16	3.86e-04	-2.63e-05	0.0238	0.0096	1.87e-03	1.46e-03	2.2	1.7	0	
2	2	8.90e-05	-1.59e-04	0.0100	0.0095	9.88e-04	9.49e-04	1.6	1.6	0	
2	4	-1.14e-04	7.83e-05	0.0161	0.0120	1.29e-03	9.97e-04	2.2	1.7	0	*
2	6	-1.80e-05	3.17e-05	0.0113	0.0083	1.17e-03	8.88e-04	1.9	1.5	0	
2	8	-5.46e-05	1.96e-05	0.0086	0.0082	9.46e-04	9.00e-04	1.6	1.5	0	
2	10	1.05e-04	-8.11e-05	0.0138	0.0108	1.11e-03	1.01e-03	1.8	1.7	0	
2	12	1.69e-05	8.30e-05	0.0085	0.0066	9.74e-04	8.51e-04	1.6	1.4	0	*
2	14	-6.42e-06	6.91e-06	0.0074	0.0091	8.55e-04	8.65e-04	1.4	1.4	0	
2	16	1.25e-04	-1.39e-04	0.0082	0.0094	9.96e-04	9.38e-04	1.7	1.6	0	
3	4	3.39e-05	4.23e-05	0.0074	0.0044	6.27e-04	4.81e-04	1.5	1.1	0	
3	8	9.21e-06	-4.92e-05	0.0056	0.0048	6.21e-04	5.04e-04	1.5	1.2	0	
3	12	1.43e-05	-3.48e-05	0.0091	0.0043	8.13e-04	5.42e-04	1.9	1.3	0	
3	16	-1.61e-05	1.26e-05	0.0071	0.0049	6.25e-04	4.67e-04	1.5	1.1	0	
4	8	3.09e-05	-2.14e-06	0.0041	0.0025	4.29e-04	2.81e-04	1.4	0.9	0	*
4	16	-4.03e-05	-1.47e-05	0.0058	0.0030	6.22e-04	3.20e-04	2.1	1.1	0	

Table 2.5.19: test case 06; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: ! OVR*	*bias: OVR*	*applied dump - bias: OVR*
1	12000	11724	276	0	n-a
3	24000	23724	276	0	n-a
5	48000	47724	276	0	n-a
7	96000	95724	276	0	n-a

Table 2.5.20: test case 06: applied dump interval and bias.

```
--------------------------------------------------------------------------------
# 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.
#
# ------------------------------------------------------------------------------
01000101  MCSR: X/Y_DS=0 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000010  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=sec
01000000  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=adj
00000010  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=sec
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
01000000  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

01010000  CCQR_0_1: Q1: YSw1=pri YSw0=sec XSw1=pri XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01010101  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=sec  Q2: YSw1=pri YSw0=sec XSw1=pri XSw0=sec

11111111  XSTATUS: X-SDATA1: 1111, X-SDATA0: 1111
11111111  YSTATUS: Y-SDATA1: 1111, Y-SDATA0: 1111
11111111  XSTATUS: X-SDATA3: 1111, X-SDATA2: 1111
11111111  YSTATUS: Y-SDATA3: 1111, Y-SDATA2: 1111
11111111  XSTATUS: X-SDATA5: 1111, X-SDATA4: 1111
11111111  YSTATUS: Y-SDATA5: 1111, Y-SDATA4: 1111
11111111  XSTATUS: X-SDATA7: 1111, X-SDATA6: 1111
11111111  YSTATUS: Y-SDATA7: 1111, Y-SDATA6: 1111
11111111  XSTATUS: X-PHASE1: 1111, X-PHASE0: 1111
11111111  YSTATUS: Y-PHASE1: 1111, Y-PHASE0: 1111
11111111  XSTATUS: X-PHASE3: 1111, X-PHASE2: 1111
11111111  YSTATUS: Y-PHASE3: 1111, Y-PHASE2: 1111
11111111  XSTATUS: X-PHASE5: 1111, X-PHASE4: 1111
11111111  YSTATUS: Y-PHASE5: 1111, Y-PHASE4: 1111
11111111  XSTATUS: X-PHASE7: 1111, X-PHASE6: 1111
11111111  YSTATUS: Y-PHASE7: 1111, Y-PHASE6: 1111
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111111  XSTATUS: X-SE_CLK: 11111111
11111111  YSTATUS: Y-SE_CLK: 11111111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.13: content of file mcb_reg06.txt

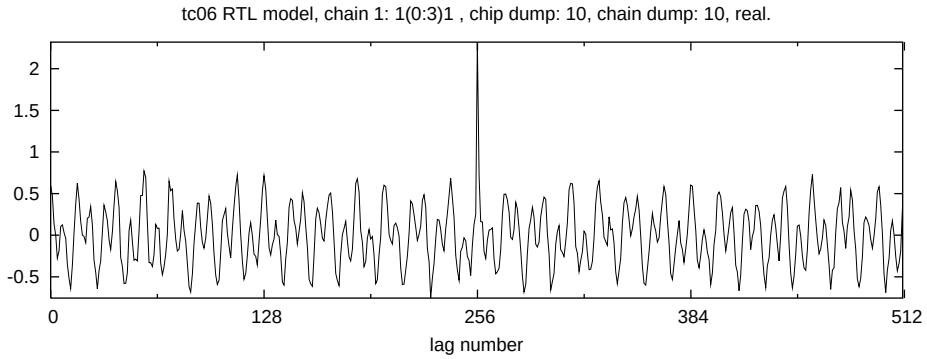

Illustration 2.5.73

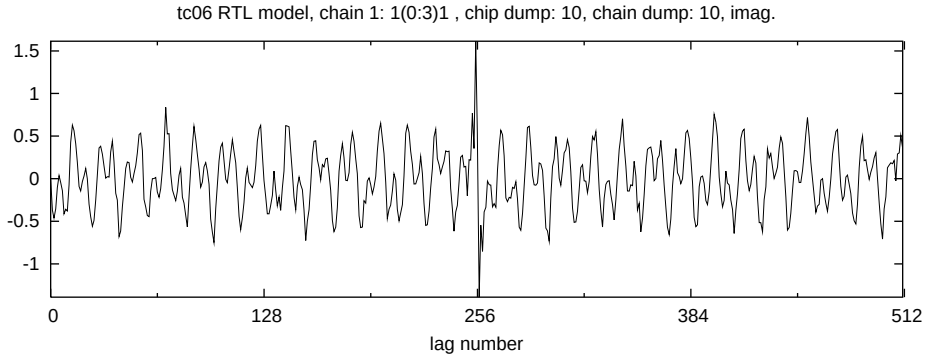

Illustration 2.5.74

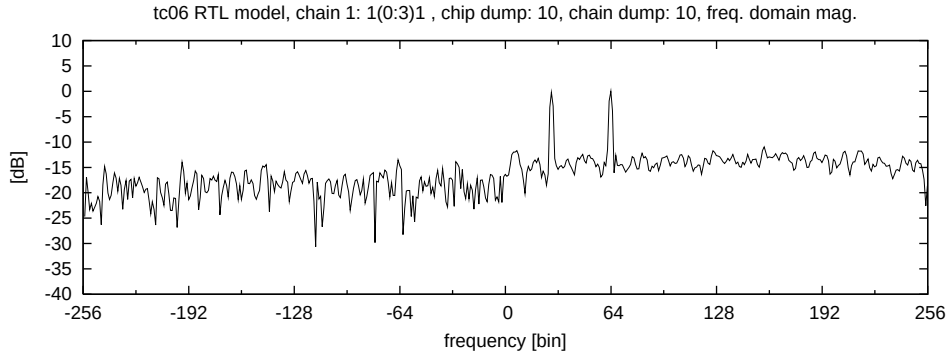

Illustration 2.5.75

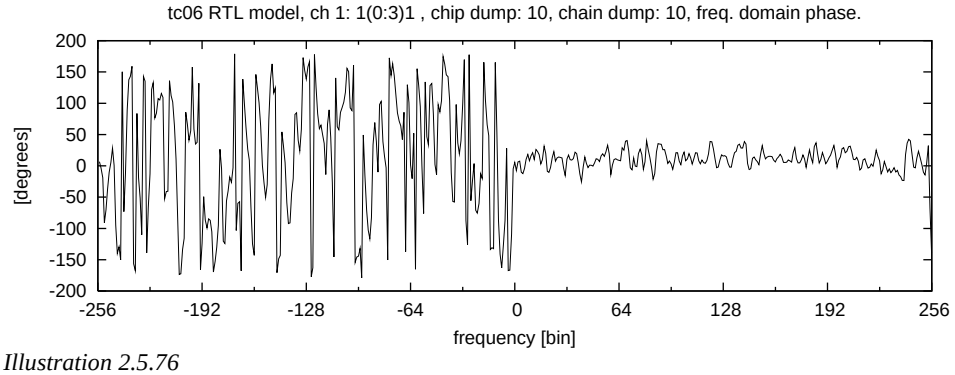

Illustration 2.5.76

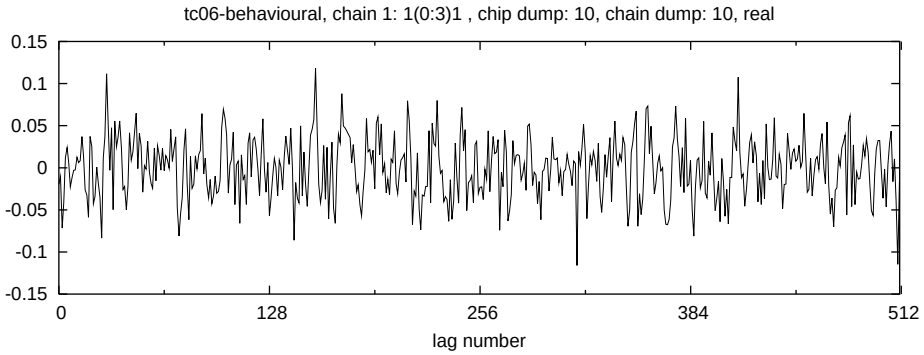

Illustration 2.5.77

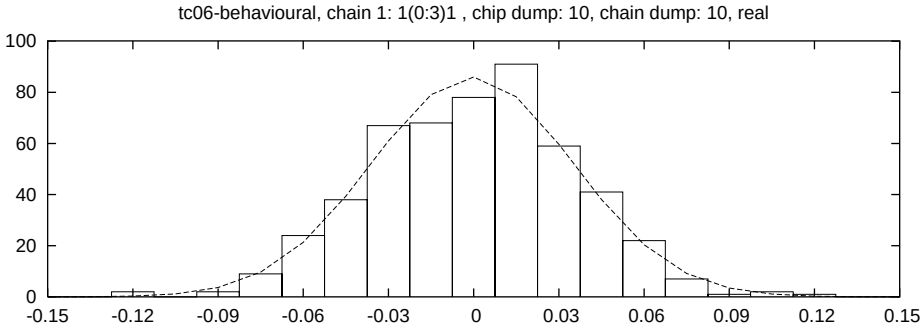

Illustration 2.5.78

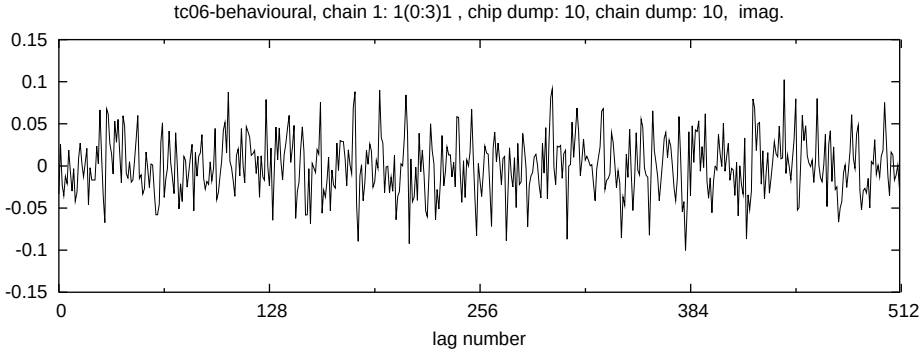

Illustration 2.5.79

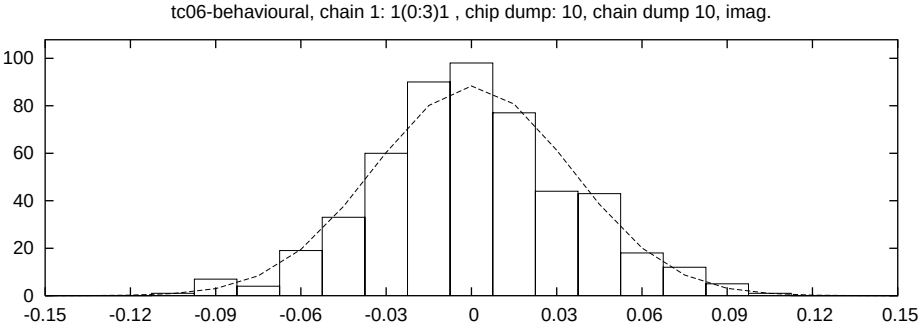

Illustration 2.5.80

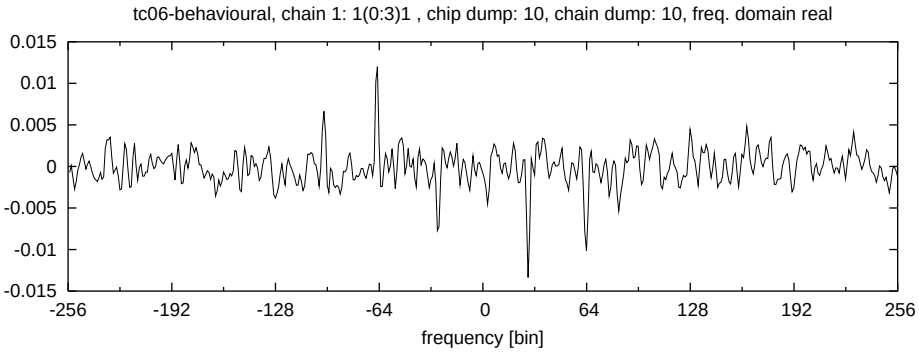

Illustration 2.5.81

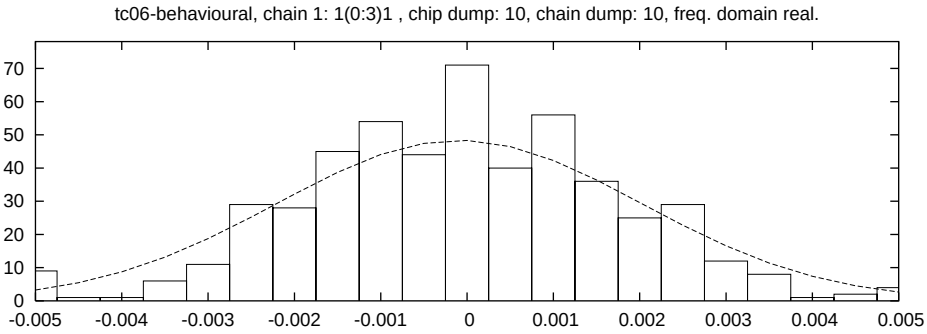

Illustration 2.5.82

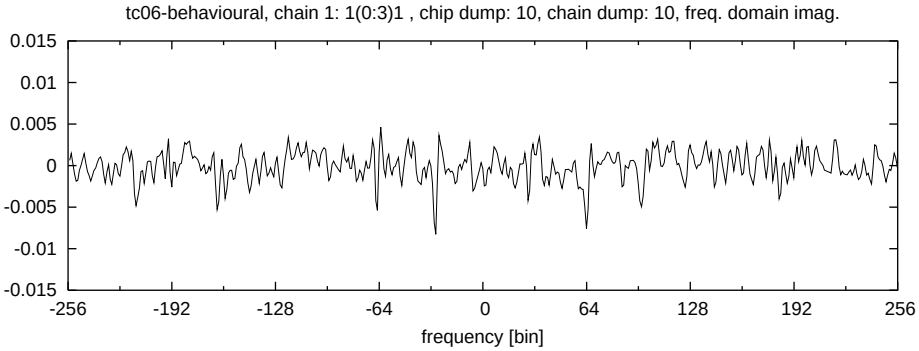

Illustration 2.5.83

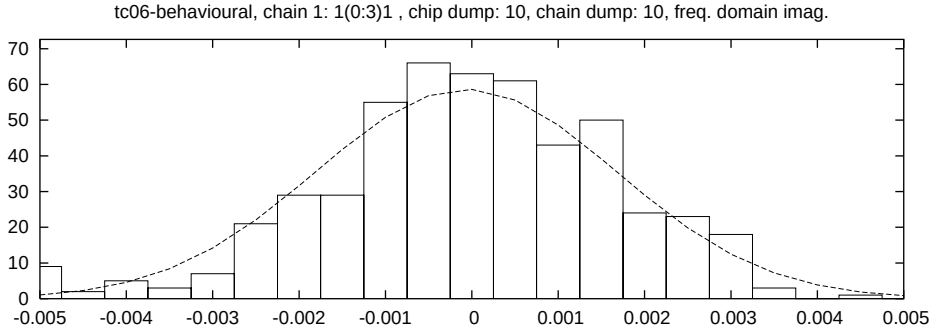

Illustration 2.5.84

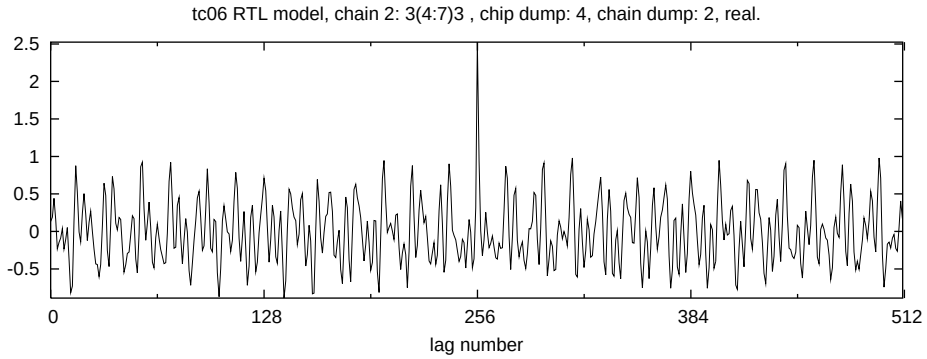

Illustration 2.5.85

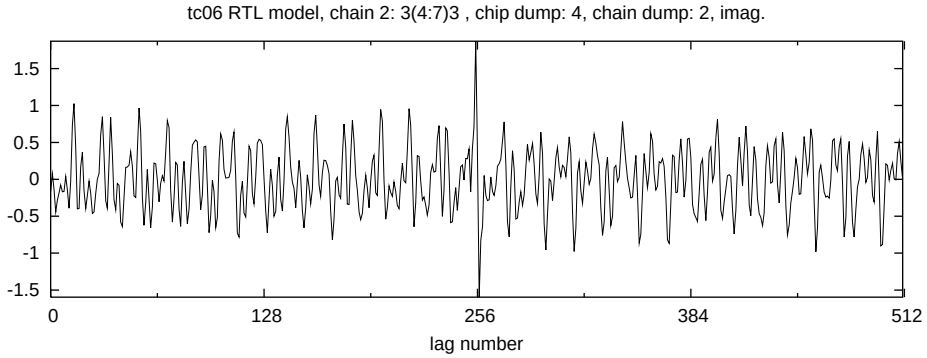

Illustration 2.5.86

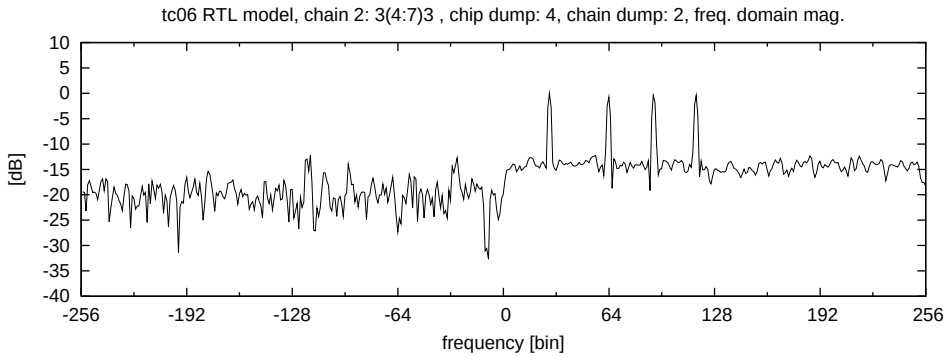

Illustration 2.5.87

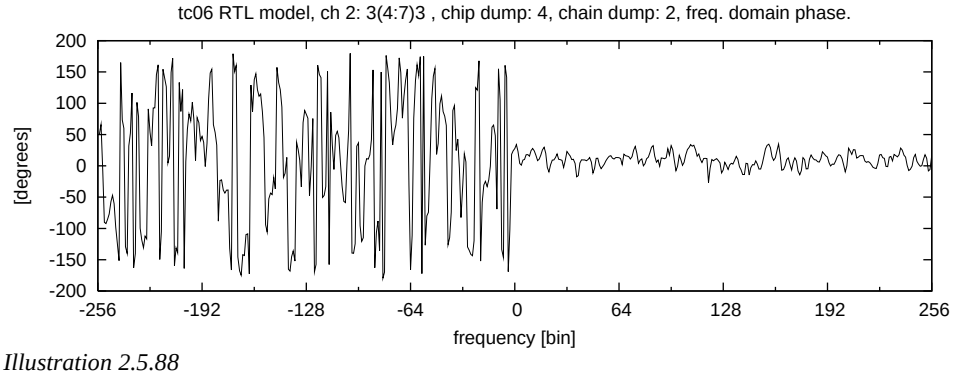

Illustration 2.5.88

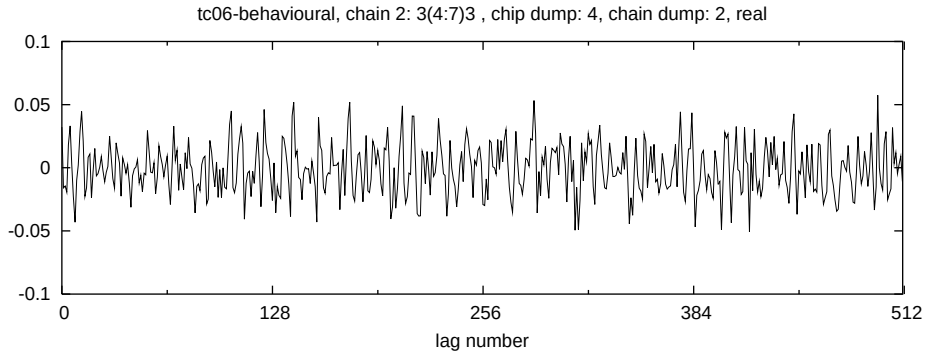

Illustration 2.5.89

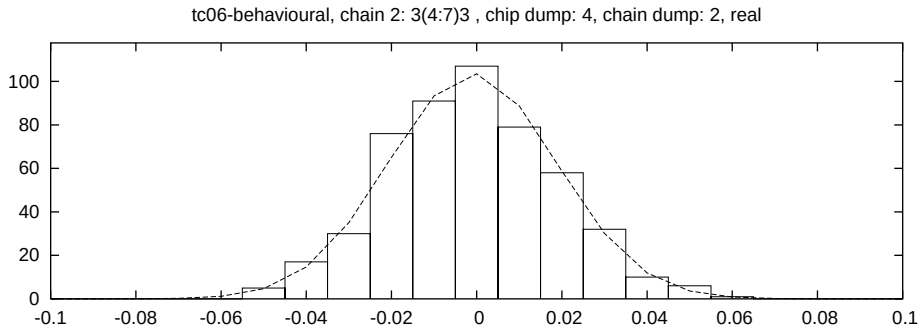

Illustration 2.5.90

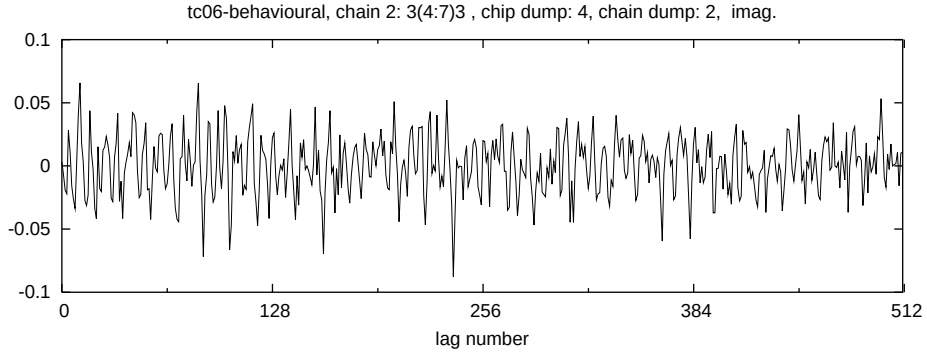

Illustration 2.5.91

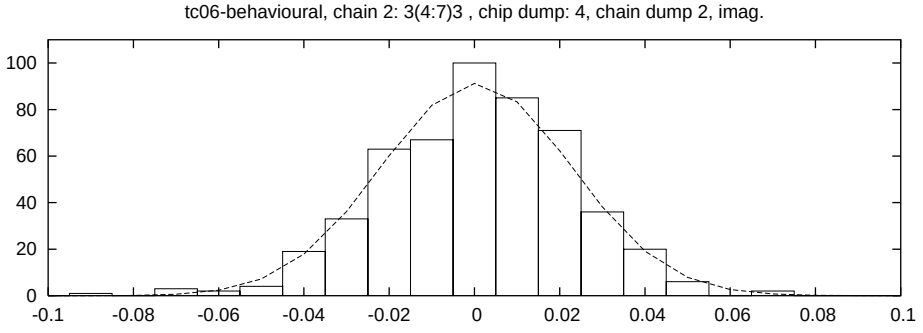

Illustration 2.5.92

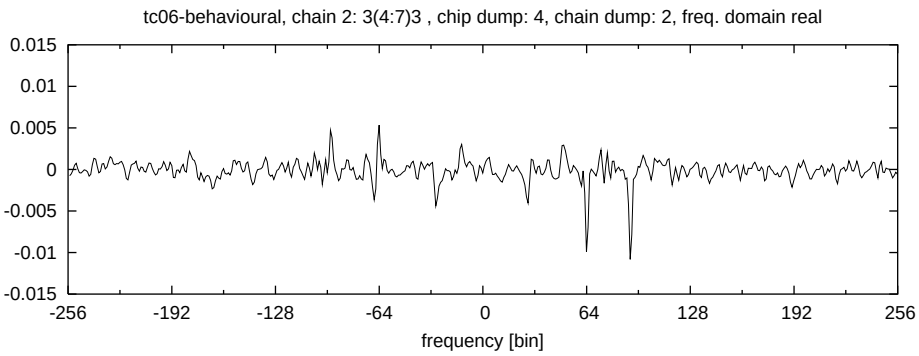

Illustration 2.5.93

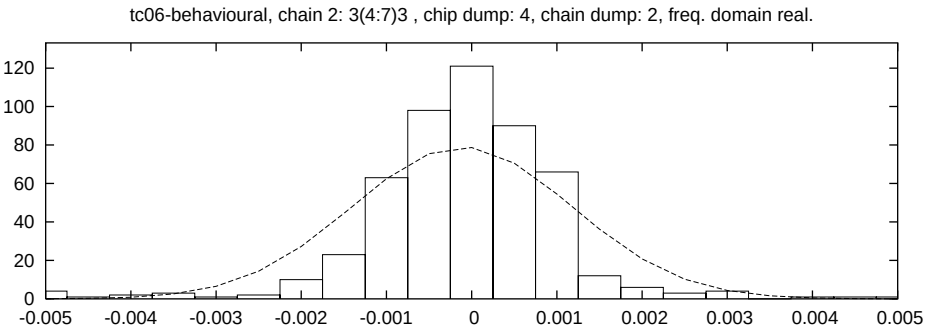

Illustration 2.5.94

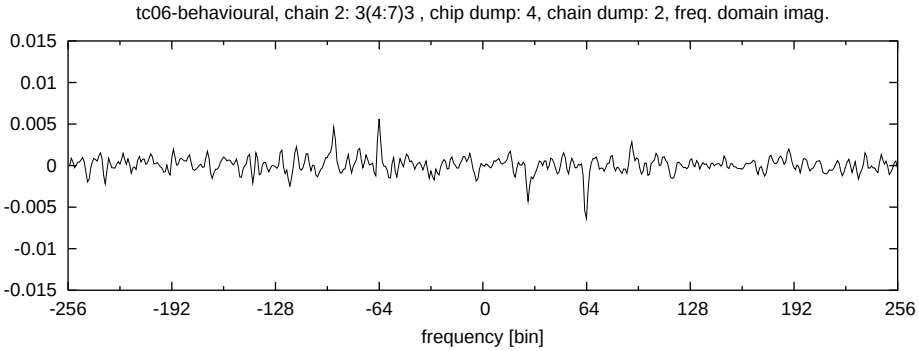

Illustration 2.5.95

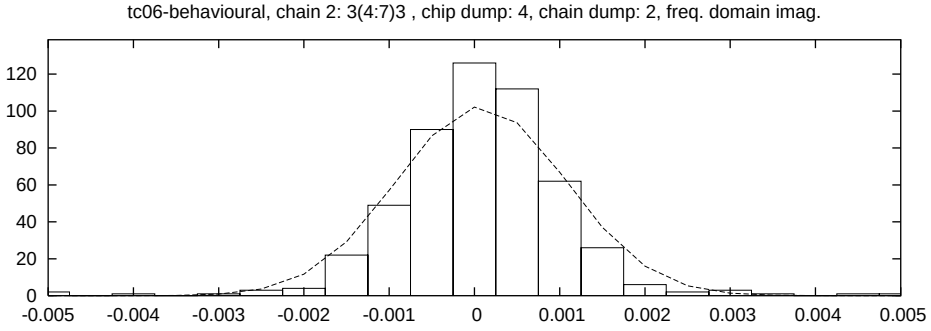

Illustration 2.5.96

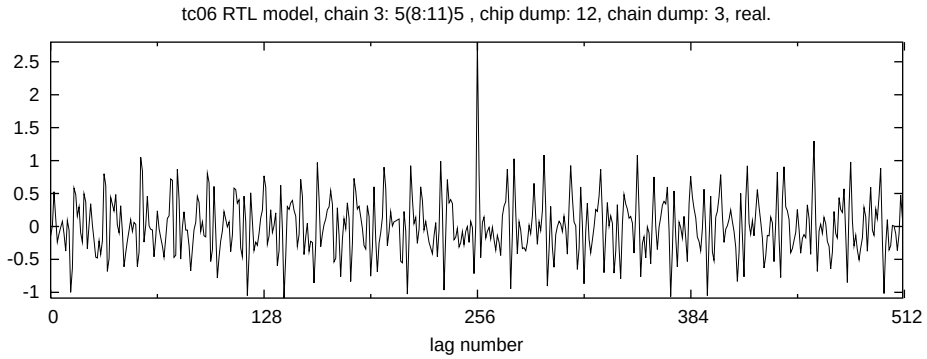

Illustration 2.5.97

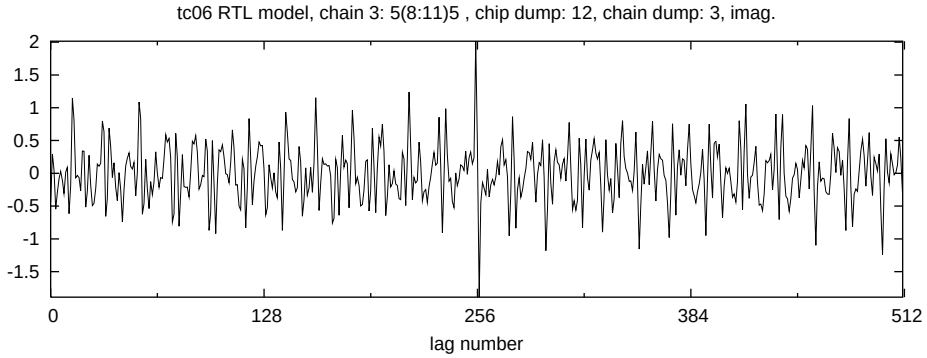

Illustration 2.5.98

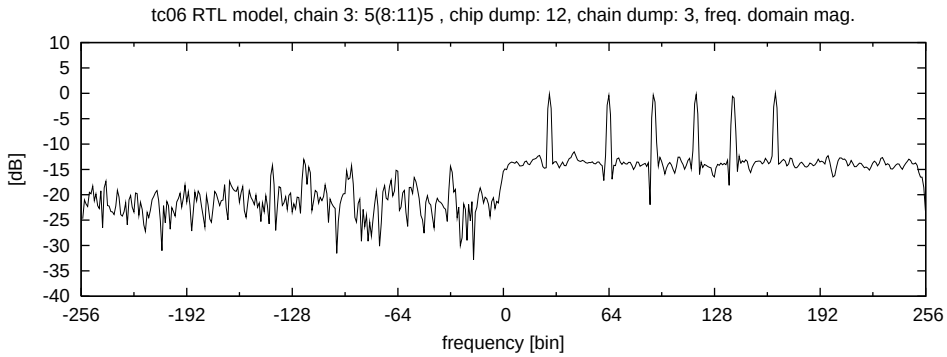

Illustration 2.5.99

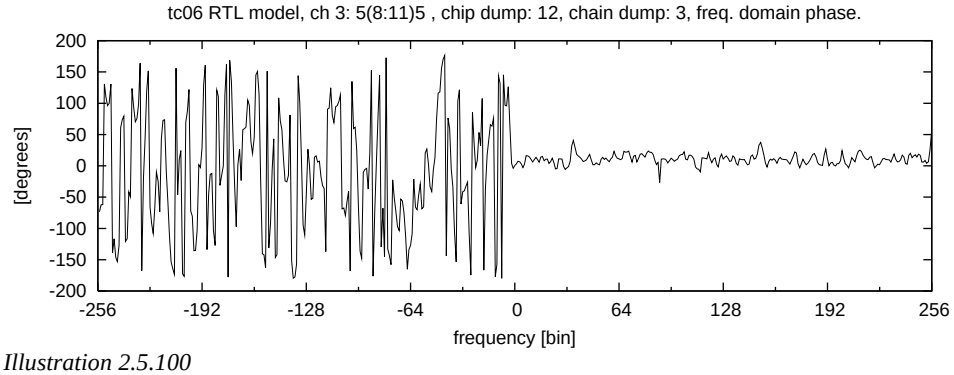

Illustration 2.5.100

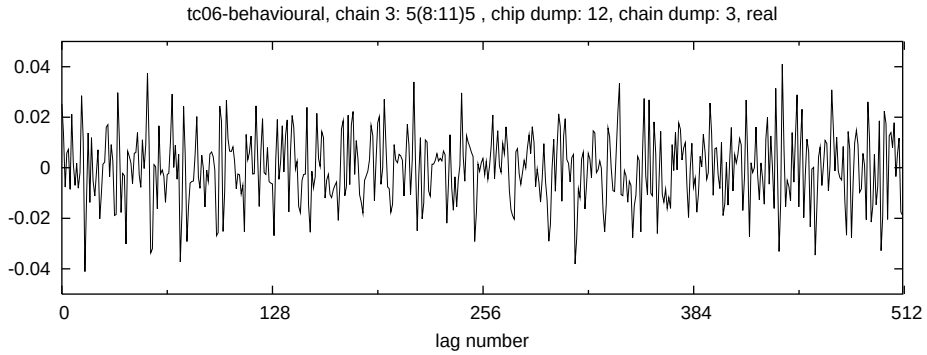

Illustration 2.5.101

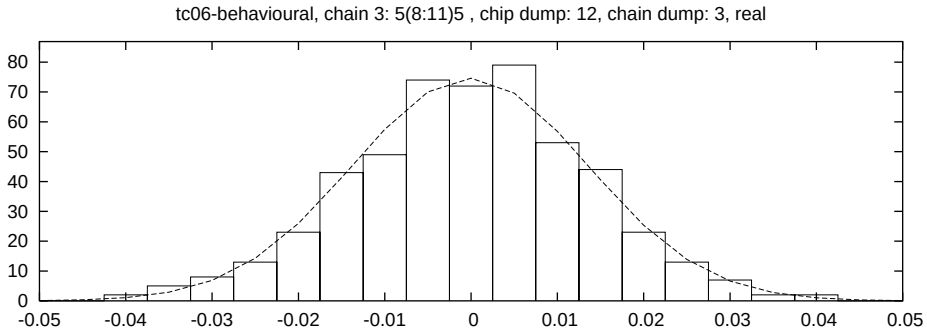

Illustration 2.5.102

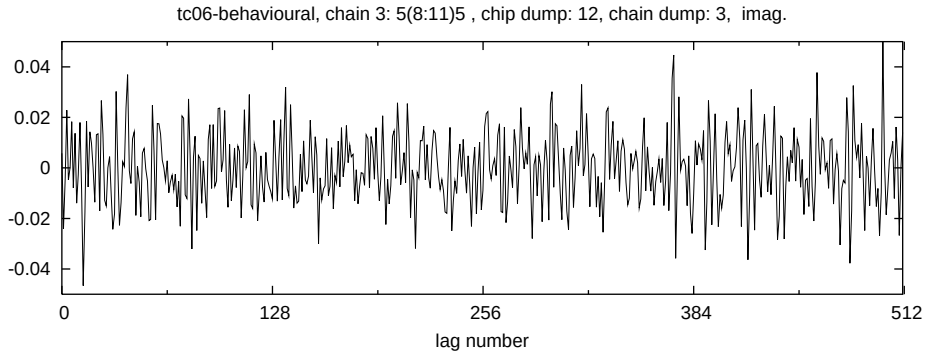

Illustration 2.5.103

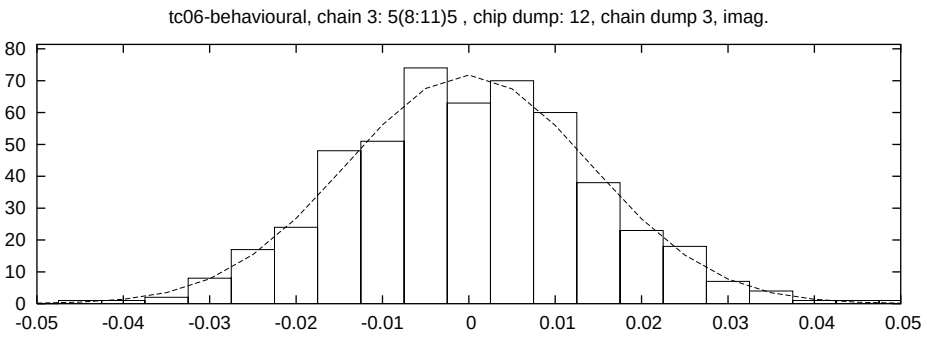

Illustration 2.5.104

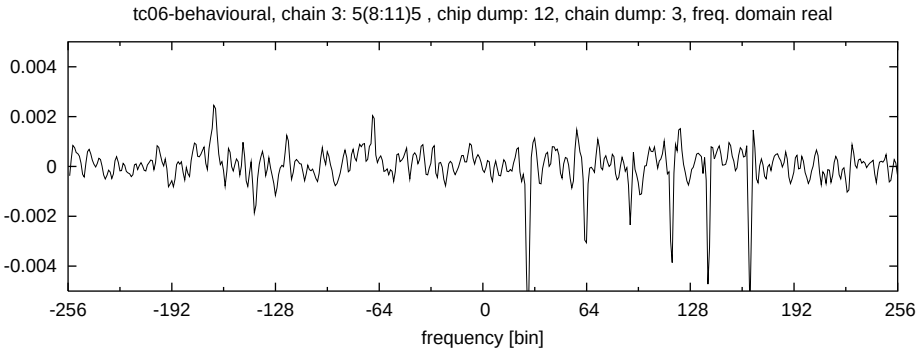

Illustration 2.5.105

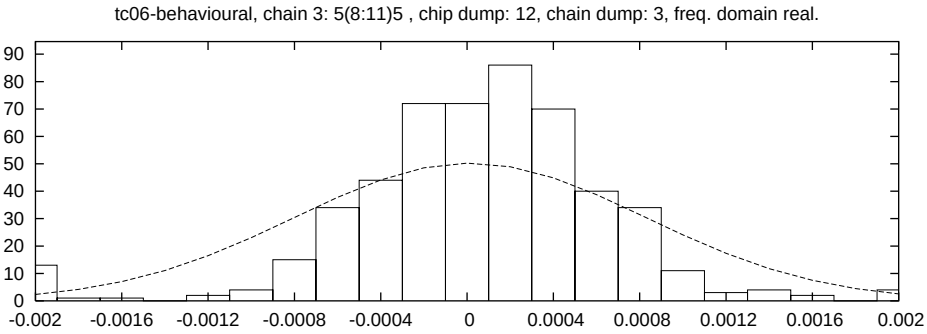

Illustration 2.5.106

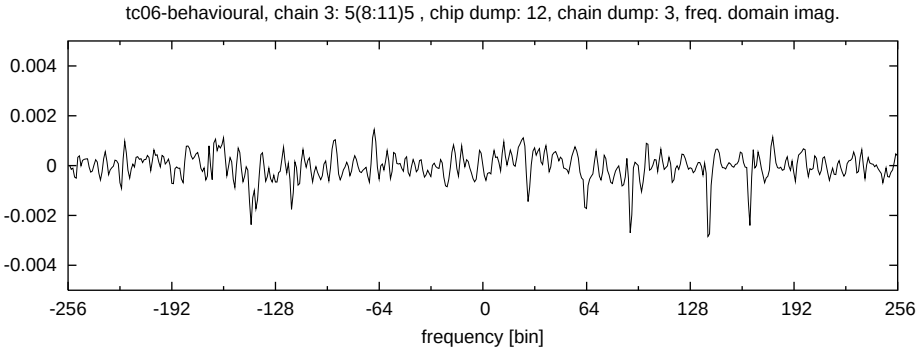

Illustration 2.5.107

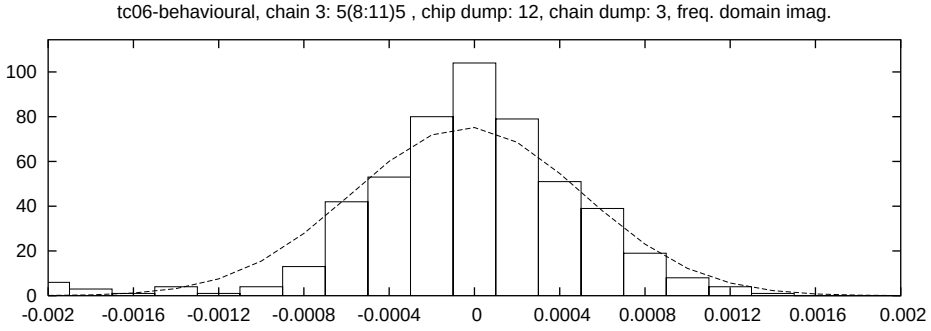

Illustration 2.5.108

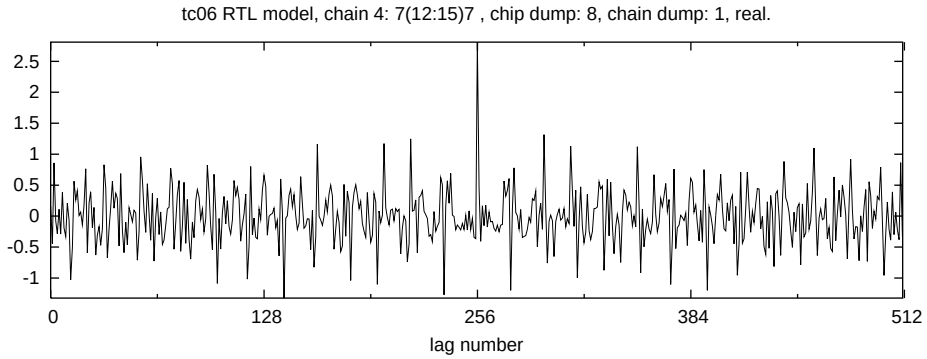

Illustration 2.5.109

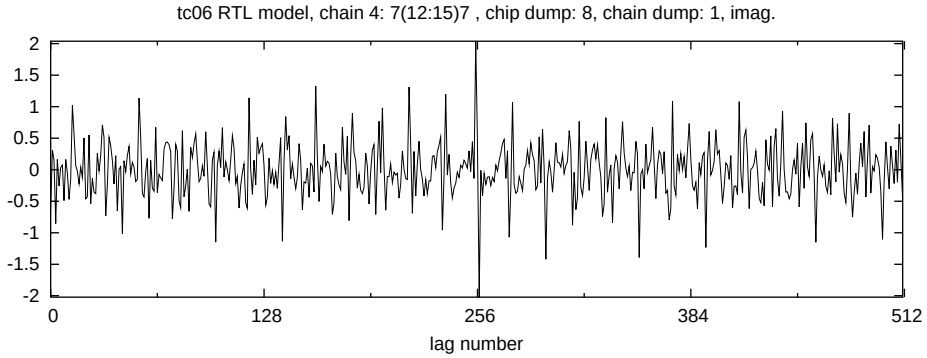

Illustration 2.5.110

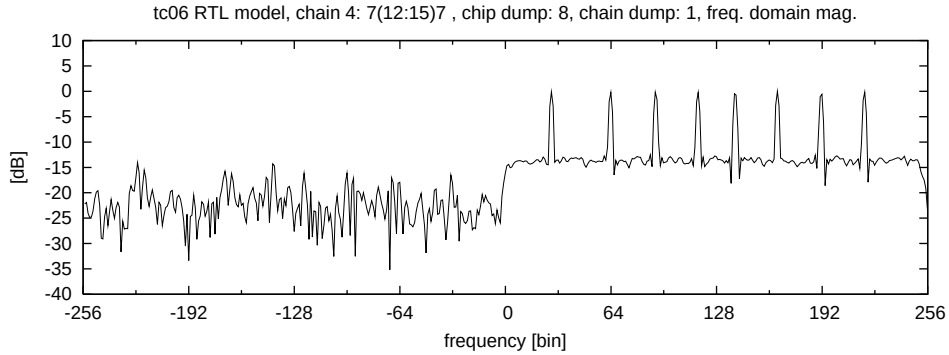

Illustration 2.5.111

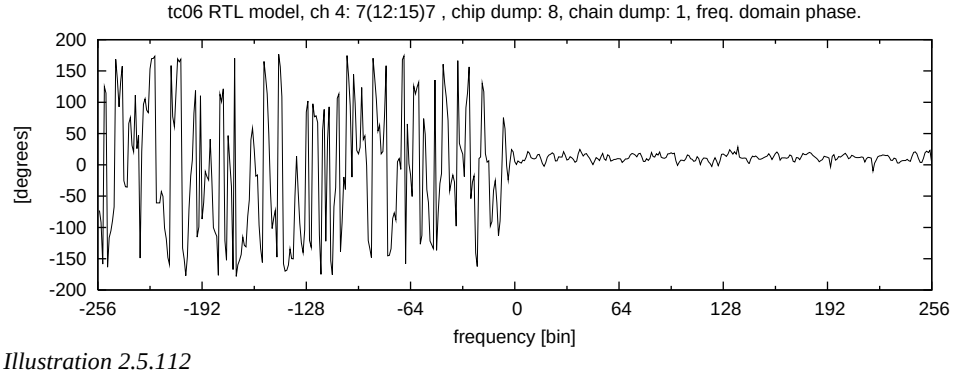

Illustration 2.5.112

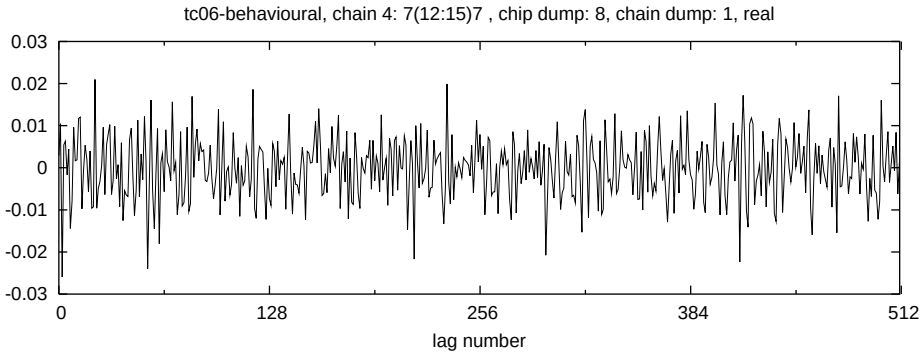

Illustration 2.5.113

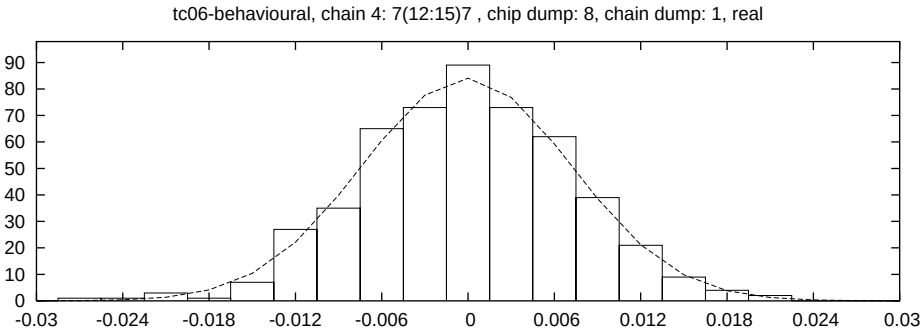

Illustration 2.5.114

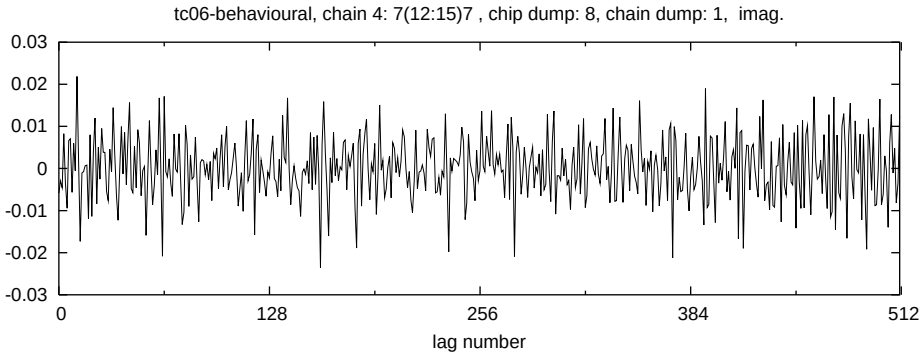

Illustration 2.5.115

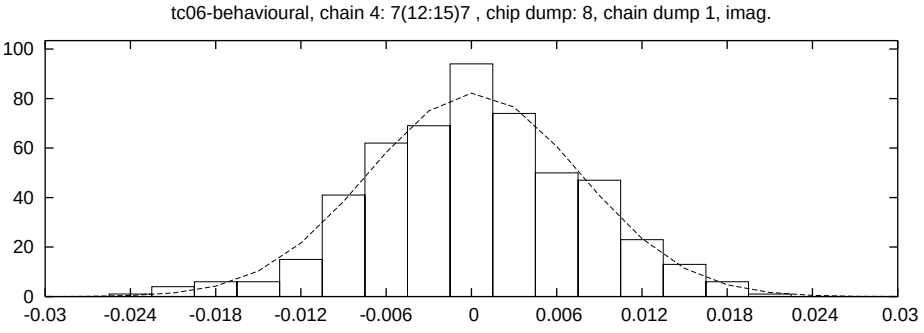

Illustration 2.5.116

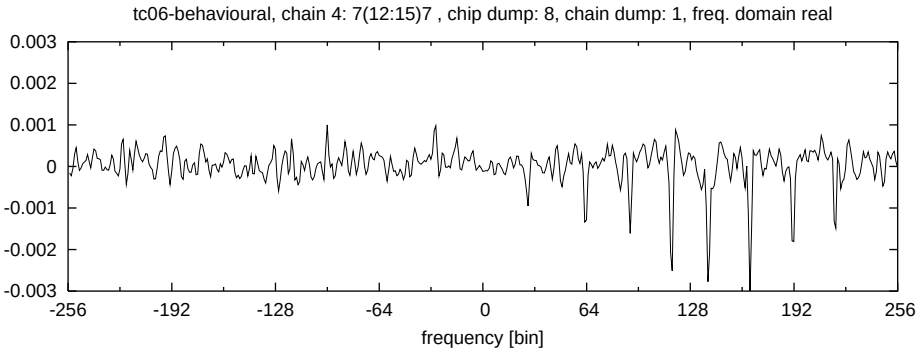

Illustration 2.5.117

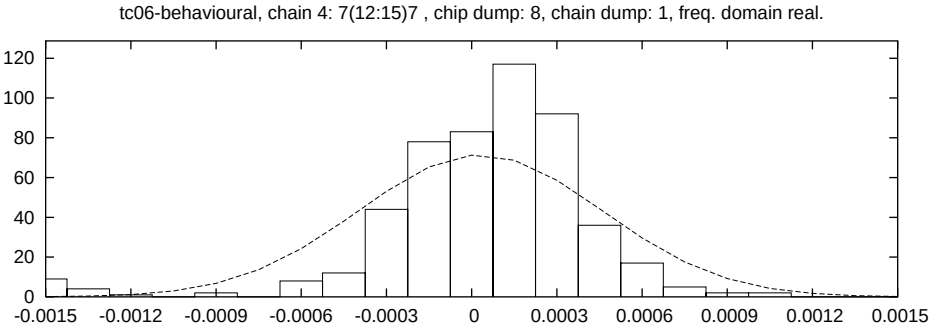

Illustration 2.5.118

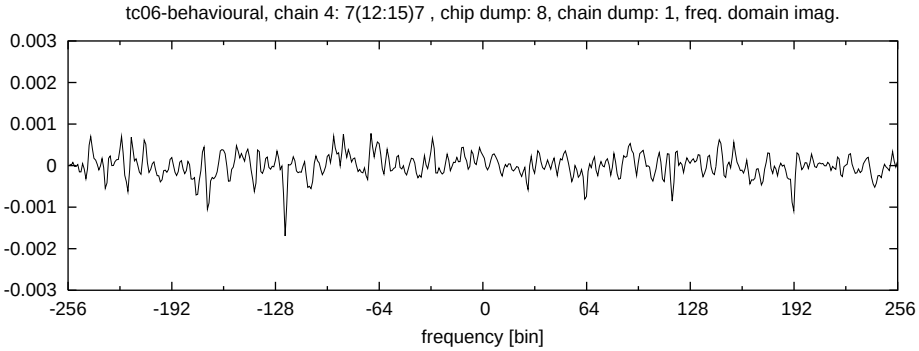

Illustration 2.5.119

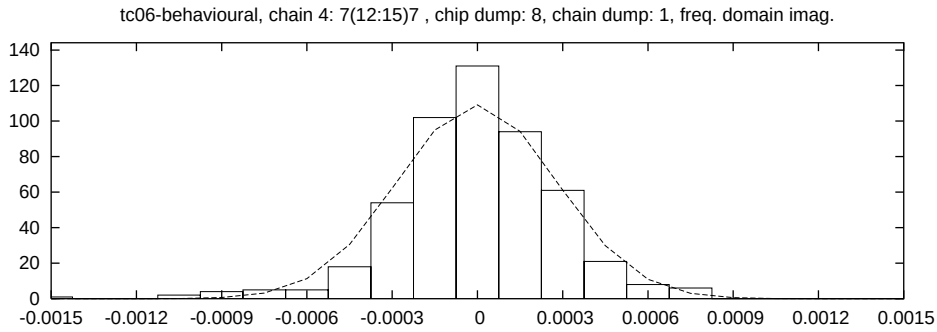

Illustration 2.5.120

2.5.7 test cases 08, 09

Synopsis: Output lag data are bit exact identical, as expected, between test cases 08 and 09 (see Text 2.5.15). Correspondence between cell numbers of the two test cases is tabulated in Table 2.5.21. Frequency domain plots show the expected number of dominant lines for the configuration. No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc08_postsim.m : V2.00 run time: Thu Sep 01 11:20:08 2005
Using postsim_lib: 1.40: 05-08-18

24 frames read for test case 08.
test case version: v3,05apr21
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
0(0:0)0  1(3:3)1  2(4:4)2  3(7:7)3
4(8:8)4  5(11:11)5  6(12:12)6  7(15:15)7


Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc08
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

List all cells represented in the output frame data for the first dump.
(
  [1] = 0
  [2] = 3
  [3] = 4
  [4] = 7
  [5] = 8
  [6] = 11
  [7] = 12
  [8] = 15
)
***--------------------------------------------------------------------------***
```

Text 2.5.14 console log for scripts tc08_postsim

```
***-------------------------------------------------------------------------***
tc09_postsim.m : V2.00 run time: Thu Sep 01 11:21:18 2005
Using postsim_lib: 1.40: 05-08-18

24 frames read for test case 09.
test case version: v3,05apr21
Checksum is OK for all frames and corrupt data was not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
0(1:1)0  1(2:2)1  2(5:5)2  3(6:6)3
4(9:9)4  5(10:10)5  6(13:13)6  7(14:14)7

WARNING from make_dump_index: Times stamp rollover detected.
 Assume rollover in ordering.
ans = 91

Initial (dump= 1) timestamp (word 0): 4294967295
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc09
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

List all cells represented in the output frame data for the first dump.
(
  [1] = 1
  [2] = 2
  [3] = 5
  [4] = 6
  [5] = 9
  [6] = 10
  [7] = 13
  [8] = 14
)
***-------------------------------------------------------------------------***
octave:6> tc08_09_similarity_check

No differences exceeding the floating point precision limit of this
machine were found between corresponding chains and dumps of test
cases 08 and 09.
```

Text 2.5.15 console log for scripts tc08_postsim, tc08_09_similarity_check

chain number	1	2	3	4	5	6	7	8
test case 08	0	3	4	7	8	11	12	15
test case 09	1	2	5	6	9	10	13	14

Table 2.5.21 mapping of similarly configured (corresponding) cells between test cases 08 and 09.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	1.08e-03	-6.91e-04	0.0808	0.1085	0.0150	0.0155	0.97	1.00	1	
1	2	1.59e-04	7.66e-04	0.0823	0.0998	0.0135	0.0161	0.87	1.04	2	
1	3	1.93e-03	7.73e-04	0.0894	0.1045	0.0144	0.0138	0.93	0.89	3	*
2	1	2.09e-03	2.28e-04	0.0888	0.0886	0.0143	0.0182	0.75	0.96	1	
2	2	1.42e-03	-3.40e-04	0.1085	0.1100	0.0183	0.0203	0.96	1.07	1	*
2	3	-1.96e-03	-2.69e-04	0.1045	0.0610	0.0184	0.0129	0.97	0.68	1	
3	1	1.75e-04	-7.85e-04	0.0662	0.0920	0.0151	0.0208	0.69	0.95	0	
3	2	-2.39e-03	-2.37e-04	0.0902	0.0883	0.0184	0.0195	0.84	0.89	0	*
3	3	1.03e-03	-1.50e-03	0.0827	0.1179	0.0167	0.0245	0.76	1.12	0	
4	1	2.17e-03	-2.05e-03	0.1296	0.1103	0.0208	0.0216	0.85	0.88	1	
4	2	3.25e-03	2.63e-04	0.1188	0.1623	0.0229	0.0309	0.94	1.26	0	*
4	3	-2.21e-03	-1.12e-03	0.1134	0.1266	0.0237	0.0258	0.97	1.05	0	
5	1	-7.89e-05	1.43e-03	0.1195	0.1154	0.0217	0.0206	0.81	0.77	0	
5	2	-1.40e-03	-1.01e-03	0.1712	0.1500	0.0293	0.0242	1.09	0.90	1	*
5	3	1.31e-03	-2.29e-03	0.1292	0.1396	0.0275	0.0250	1.02	0.93	0	
6	1	1.25e-03	-8.41e-04	0.1299	0.1291	0.0241	0.0221	0.83	0.76	2	
6	2	2.27e-03	3.12e-03	0.1349	0.1149	0.0270	0.0230	0.93	0.79	0	*
6	3	-2.71e-03	3.78e-04	0.1732	0.1529	0.0303	0.0262	1.04	0.90	2	
7	1	1.13e-03	1.33e-05	0.1108	0.1495	0.0255	0.0263	0.82	0.85	1	
7	2	3.54e-04	6.00e-04	0.1098	0.1186	0.0224	0.0247	0.72	0.80	0	*
7	3	2.93e-04	1.75e-04	0.1552	0.1140	0.0246	0.0228	0.79	0.74	1	
8	1	1.41e-04	1.75e-03	0.1040	0.1042	0.0191	0.0212	0.58	0.64	0	
8	2	1.33e-03	7.92e-04	0.1367	0.1268	0.0252	0.0251	0.77	0.76	1	
8	3	9.65e-04	1.69e-03	0.1573	0.1538	0.0266	0.0281	0.81	0.85	1	*

Table 2.5.22 test case 08,09; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-4.09e-03	8.07e-03	0.3727	0.2685	0.0621	0.0605	0.98	0.83	64
1	2	3.97e-03	-4.89e-03	0.3630	0.4115	0.0694	0.0796	0.96	0.94	63 64
2	1	-6.73e-05	1.03e-03	0.4067	0.4953	0.0777	0.0852	1.08	1.05	64
2	2	7.94e-04	3.94e-03	0.3244	0.4477	0.0697	0.0899	0.96	1.02	null
3	1	-1.31e-02	9.19e-03	0.4388	0.4401	0.0924	0.0840	1.03	0.96	null
3	2	1.03e-02	-2.25e-03	0.4923	0.4450	0.0905	0.0851	0.98	0.95	null
4	1	-4.18e-03	-3.10e-03	0.5136	0.4776	0.0939	0.1048	0.97	0.95	null
4	2	-6.68e-03	5.33e-03	0.5627	0.4140	0.1050	0.0945	0.98	1.02	null
5	1	-7.41e-03	-2.21e-03	0.4791	0.5158	0.1032	0.0971	1.09	1.04	null
5	2	1.25e-02	3.45e-03	0.5087	0.4947	0.0990	0.1019	0.97	1.01	null
6	1	7.62e-04	2.10e-03	0.5526	0.6236	0.1018	0.1132	0.96	0.99	null
6	2	-7.40e-03	-3.10e-03	0.4936	0.7070	0.1085	0.1145	0.96	0.95	null
7	1	9.73e-05	-8.65e-03	0.5604	0.4866	0.1170	0.1039	0.96	0.94	null
7	2	3.84e-03	1.31e-02	0.5758	0.5957	0.1190	0.1116	1.03	0.96	null
8	1	-3.96e-03	1.78e-03	0.6738	0.5817	0.1201	0.1327	0.98	1.02	null
8	2	-3.16e-03	5.23e-04	0.5434	0.6924	0.1242	0.1245	1.01	0.99	null

Table 2.5.23 test case 08,09; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	dump	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	range $\Re[f^F - b^F]$	range $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	peak count	plot
1	1	1.20e-04	1.15e-03	0.0064	0.0157	1.28e-03	2.00e-03	2.7	4.3	0	
1	2	-2.02e-04	-9.64e-04	0.0066	0.0160	1.15e-03	2.06e-03	2.5	4.4	0	
1	3	4.31e-04	1.17e-03	0.0079	0.0144	1.27e-03	2.02e-03	2.7	4.3	0	*
2	1	7.55e-04	-2.84e-04	0.0176	0.0089	2.11e-03	1.54e-03	3.7	2.7	0	
2	2	9.00e-04	9.27e-04	0.0155	0.0159	2.14e-03	2.32e-03	3.8	4.1	0	*
2	3	-2.94e-04	3.83e-04	0.0113	0.0106	1.88e-03	1.70e-03	3.3	3.0	0	
3	1	-3.52e-04	1.26e-04	0.0153	0.0135	2.24e-03	1.87e-03	3.4	2.8	0	
3	2	2.58e-04	-4.30e-04	0.0164	0.0092	2.14e-03	1.82e-03	3.2	2.8	0	*
3	3	7.84e-05	-1.65e-04	0.0167	0.0161	2.44e-03	2.17e-03	3.7	3.3	0	
4	1	-4.05e-04	-6.58e-04	0.0135	0.0120	2.38e-03	2.16e-03	3.2	2.9	0	
4	2	-2.70e-05	-3.88e-04	0.0217	0.0226	2.98e-03	3.09e-03	4.1	4.2	0	*
4	3	6.51e-04	5.13e-04	0.0154	0.0158	2.42e-03	2.63e-03	3.3	3.6	0	
5	1	1.24e-04	-9.86e-05	0.0181	0.0089	2.74e-03	1.69e-03	3.4	2.1	0	
5	2	-5.64e-04	5.58e-04	0.0158	0.0220	3.06e-03	2.95e-03	3.8	3.7	0	*
5	3	6.53e-04	7.13e-04	0.0213	0.0131	3.02e-03	2.58e-03	3.8	3.2	0	
6	1	6.80e-05	-3.52e-04	0.0178	0.0121	2.88e-03	2.10e-03	3.3	2.4	0	
6	2	-2.01e-04	7.28e-04	0.0150	0.0180	2.66e-03	2.78e-03	3.1	3.2	0	*
6	3	5.69e-04	-3.58e-04	0.0191	0.0174	3.10e-03	2.50e-03	3.6	2.9	0	
7	1	-7.72e-04	-7.90e-04	0.0180	0.0125	2.91e-03	2.38e-03	3.1	2.6	0	
7	2	9.40e-04	1.60e-04	0.0125	0.0106	2.54e-03	2.13e-03	2.7	2.3	0	*
7	3	-2.93e-04	2.17e-04	0.0145	0.0105	2.52e-03	2.20e-03	2.7	2.4	0	
8	1	2.15e-04	6.57e-04	0.0115	0.0121	2.23e-03	2.17e-03	2.3	2.2	0	
8	2	-1.34e-04	-2.81e-04	0.0145	0.0170	2.85e-03	2.58e-03	2.9	2.6	0	
8	3	-5.38e-05	-5.90e-04	0.0159	0.0186	2.84e-03	3.18e-03	2.9	3.2	0	*

Table 2.5.24 test case 08,09; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	applied dump	bias:!OVR	applied dump - bias: ! OVR	bias: OVR	applied dump - bias: OVR
0	40000	39724	276	0	n-a
1	40000	39724	276	0	n-a
2	40000	39724	276	0	n-a
3	40000	39724	276	0	n-a
4	40000	39724	276	0	n-a
5	40000	39724	276	0	n-a
6	40000	39724	276	0	n-a
7	40000	39724	276	0	n-a

Table 2.5.25 test case 08,09; applied dump and bias.

```
# ------------------------------------------------------------------------------
# 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.
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

11111001  CCCSCR_0_1:   CCC1: Y=n/c X=n/c   CCC0: Y=sec X=pri
01101111  CCCSCR_2_3:   CCC3: Y=pri X=sec   CCC2: Y=n/c X=n/c
11111001  CCCSCR_4_5:   CCC5: Y=n/c X=n/c   CCC4: Y=sec X=pri
01101111  CCCSCR_6_7:   CCC7: Y=pri X=sec   CCC6: Y=n/c X=n/c
11111001  CCCSCR_8_9:   CCC9: Y=n/c X=n/c   CCC8: Y=sec X=pri
01101111  CCCSCR_10_11: CCC11: Y=pri X=sec CCC10: Y=n/c X=n/c
11111001  CCCSCR_12_13: CCC13: Y=n/c X=n/c CCC12: Y=sec X=pri
01101111  CCCSCR_14_15: CCC15: Y=pri X=sec CCC14: Y=n/c X=n/c

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.16: content of file mcb_reg08.txt.

```
# ------------------------------------------------------------------------------
# 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.
#
# ------------------------------------------------------------------------------
00000101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10011111  CCCSCR_0_1:   CCC1: Y=sec X=pri   CCC0: Y=n/c X=n/c
11110110  CCCSCR_2_3:   CCC3: Y=n/c X=n/c   CCC2: Y=pri X=sec
10011111  CCCSCR_4_5:   CCC5: Y=sec X=pri   CCC4: Y=n/c X=n/c
11110110  CCCSCR_6_7:   CCC7: Y=n/c X=n/c   CCC6: Y=pri X=sec
10011111  CCCSCR_8_9:   CCC9: Y=sec X=pri   CCC8: Y=n/c X=n/c
11110110  CCCSCR_10_11: CCC11: Y=n/c X=n/c CCC10: Y=pri X=sec
10011111  CCCSCR_12_13: CCC13: Y=sec X=pri CCC12: Y=n/c X=n/c
11110110  CCCSCR_14_15: CCC15: Y=n/c X=n/c CCC14: Y=pri X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.17: content of file mcb_reg09.txt.

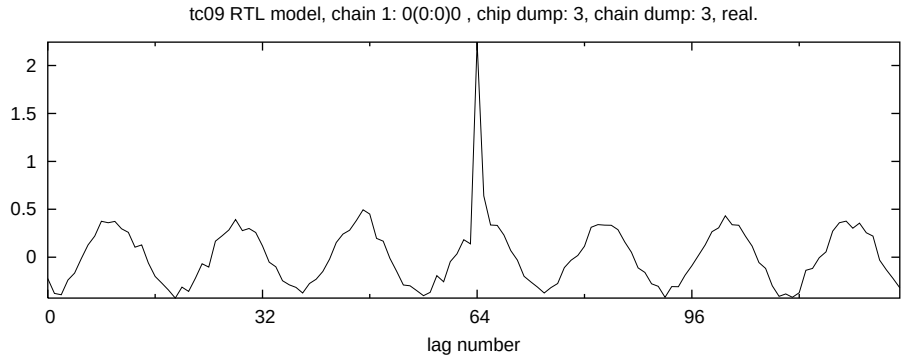

Illustration 2.5.121

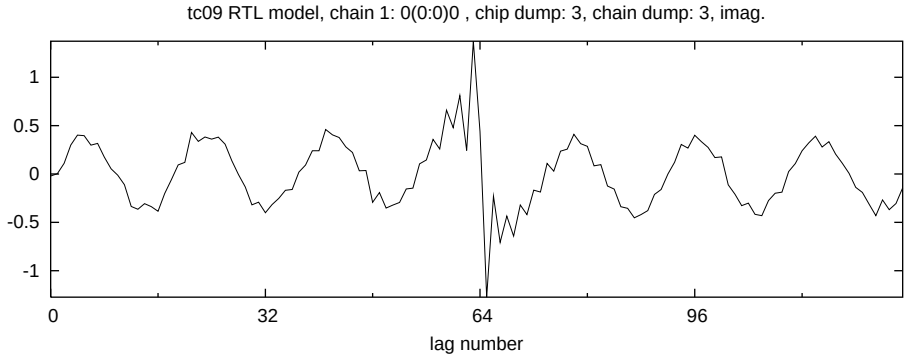

Illustration 2.5.122

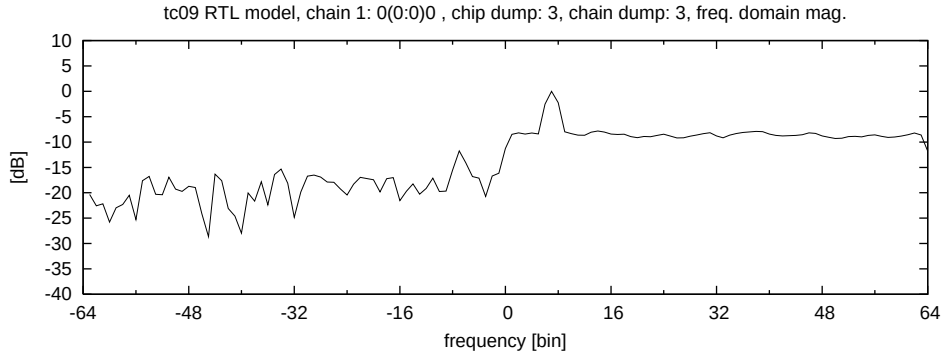

Illustration 2.5.123

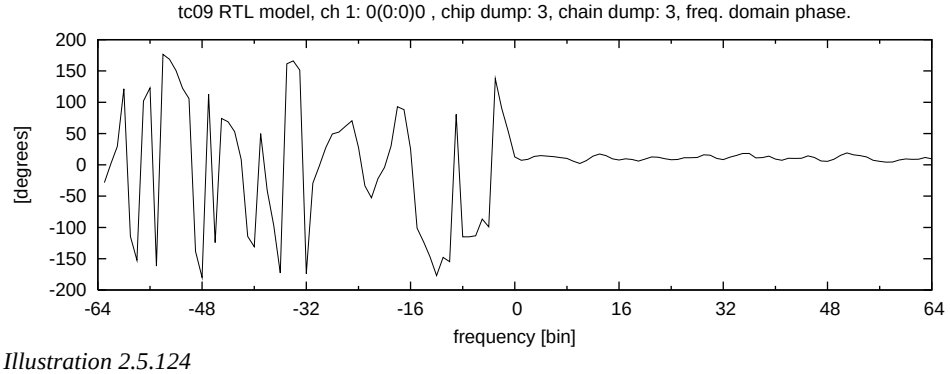

Illustration 2.5.124

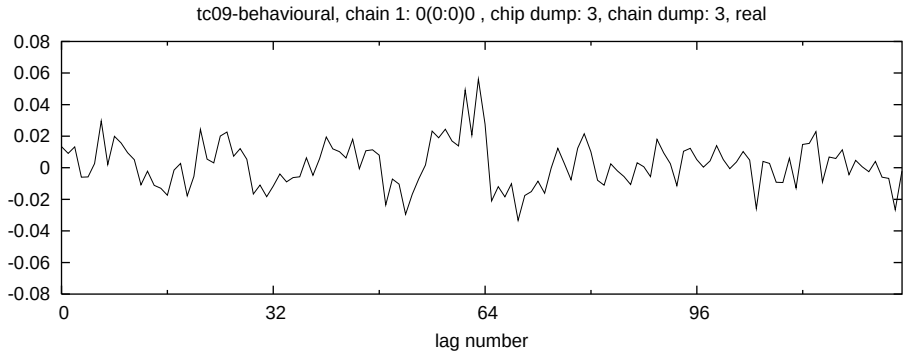

Illustration 2.5.125

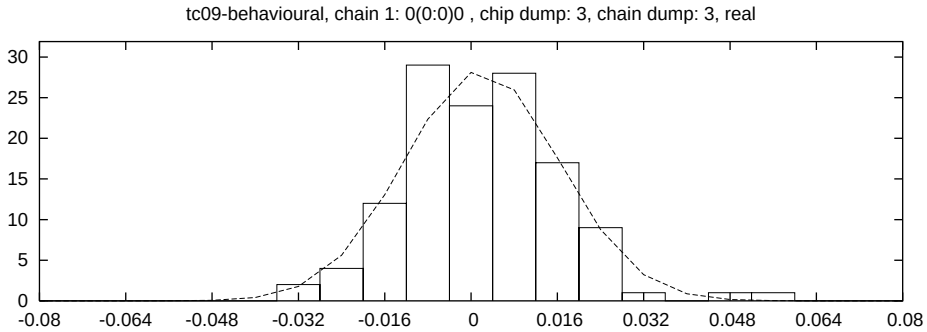

Illustration 2.5.126

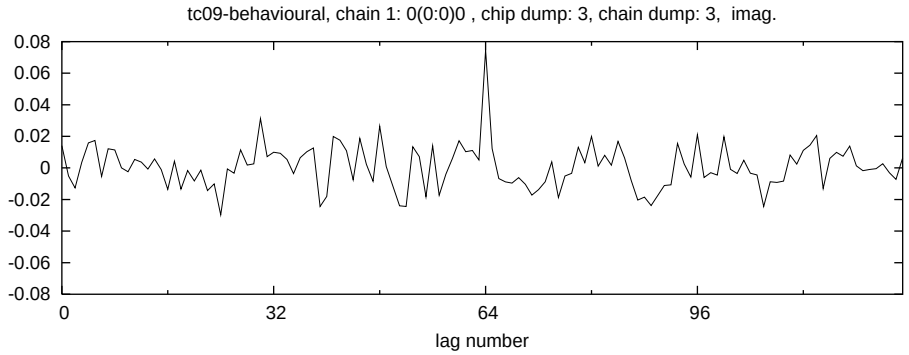

Illustration 2.5.127

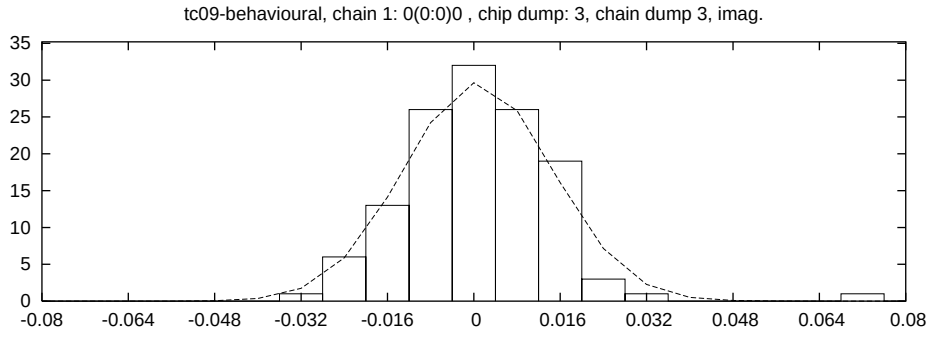

Illustration 2.5.128

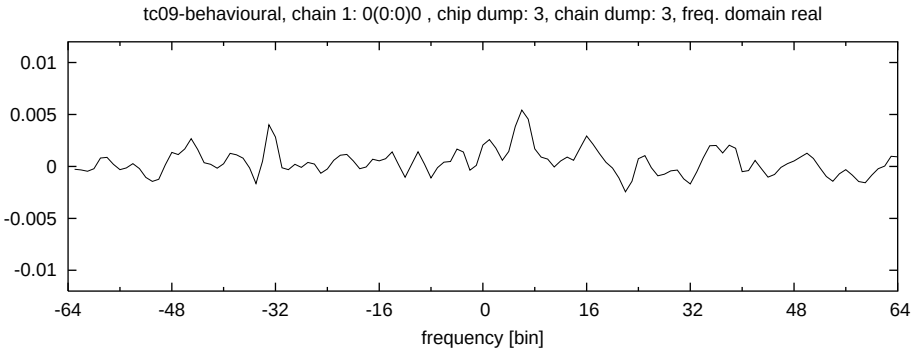

Illustration 2.5.129

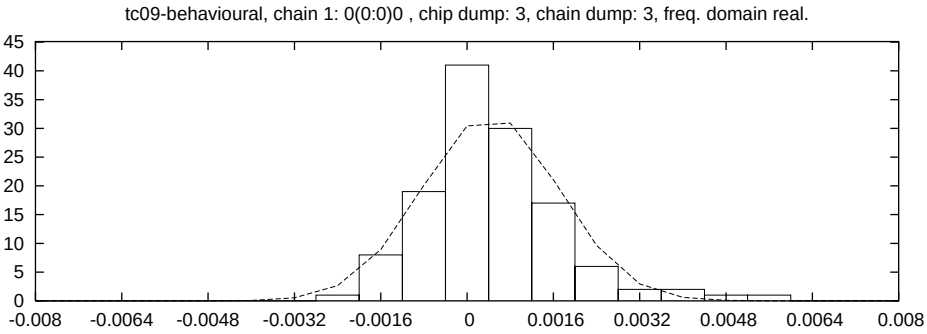

Illustration 2.5.130

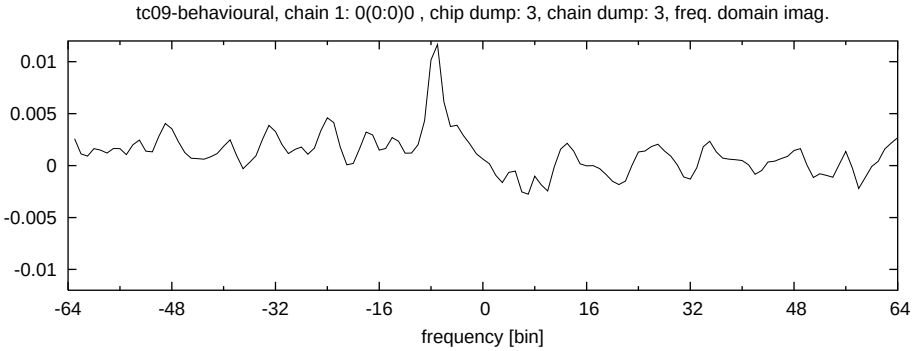

Illustration 2.5.131

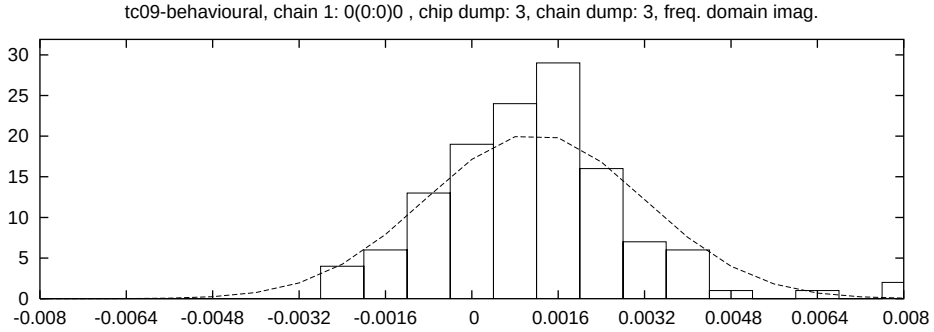

Illustration 2.5.132

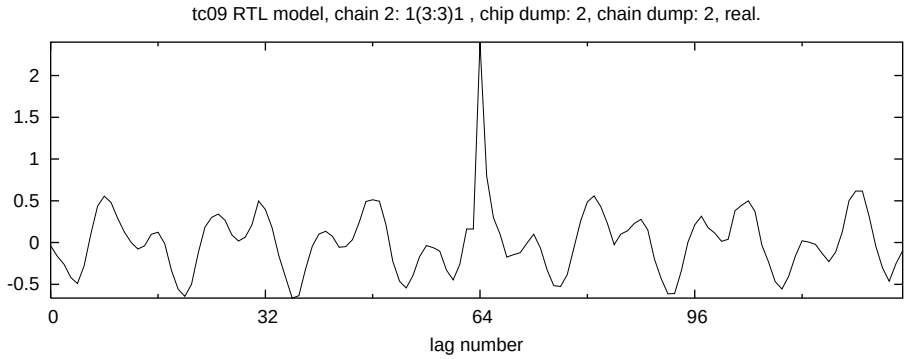

Illustration 2.5.133

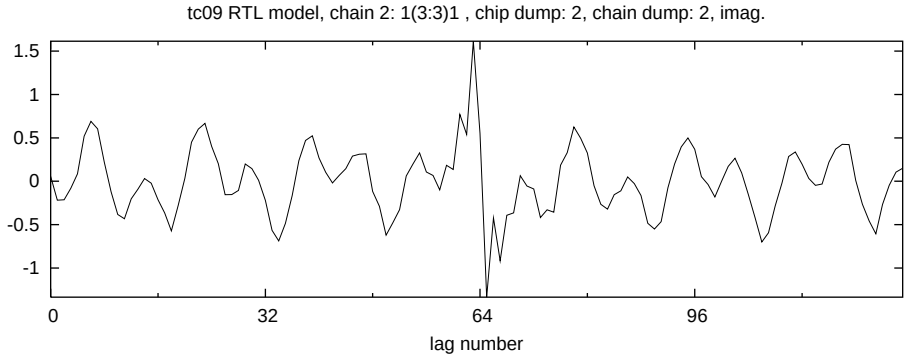

Illustration 2.5.134

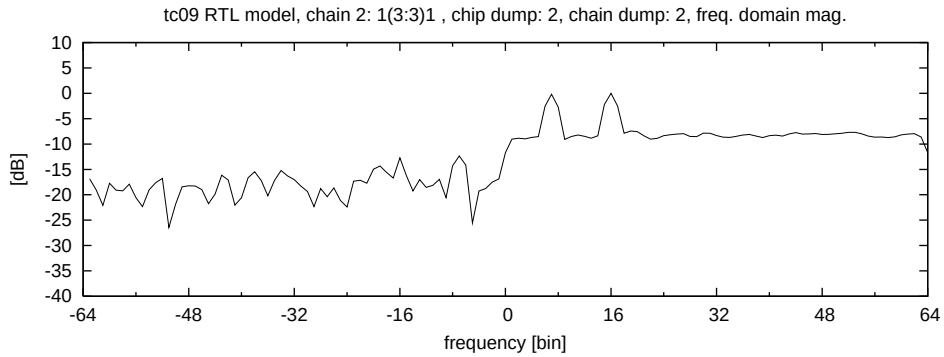

Illustration 2.5.135

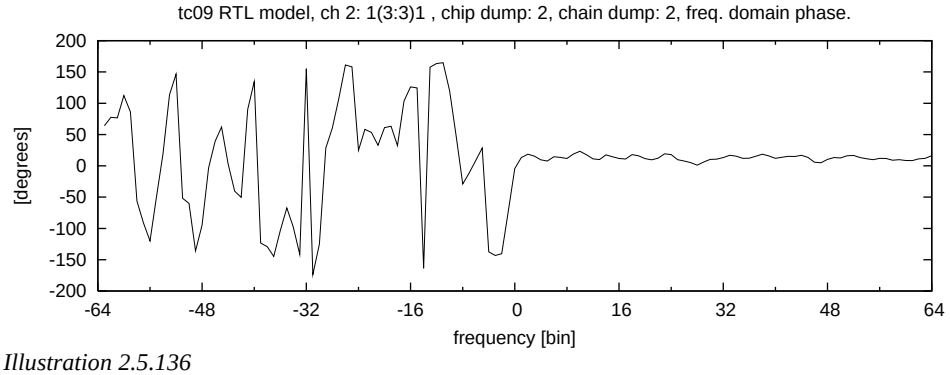

Illustration 2.5.136

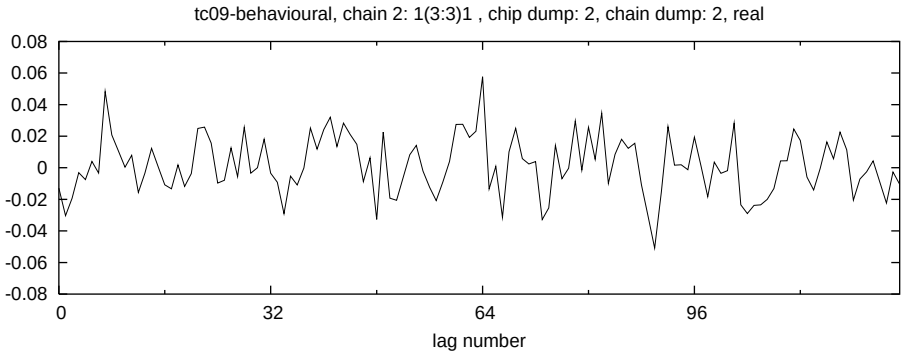

Illustration 2.5.137

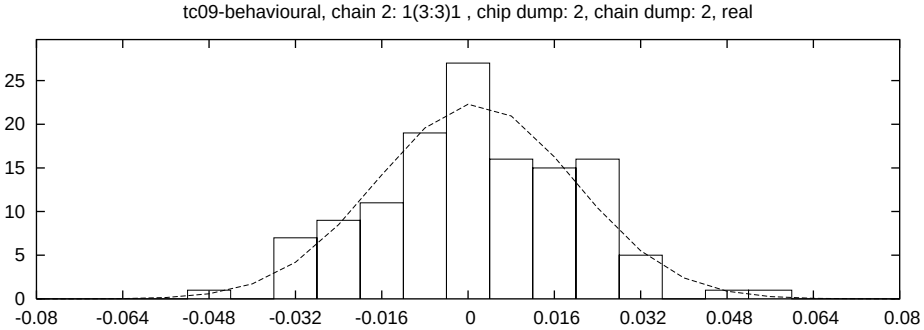

Illustration 2.5.138

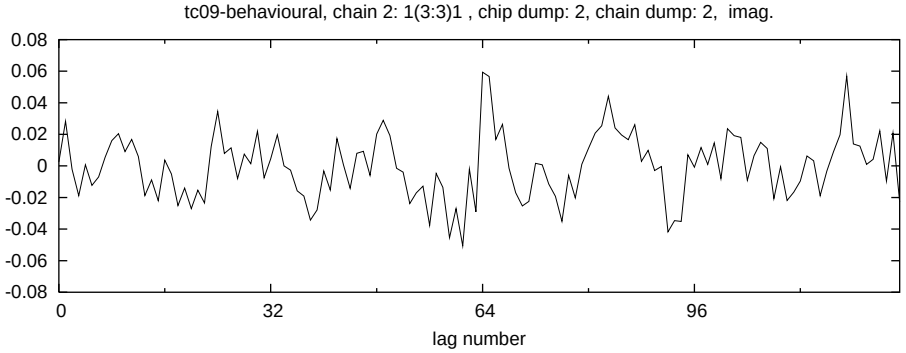

Illustration 2.5.139

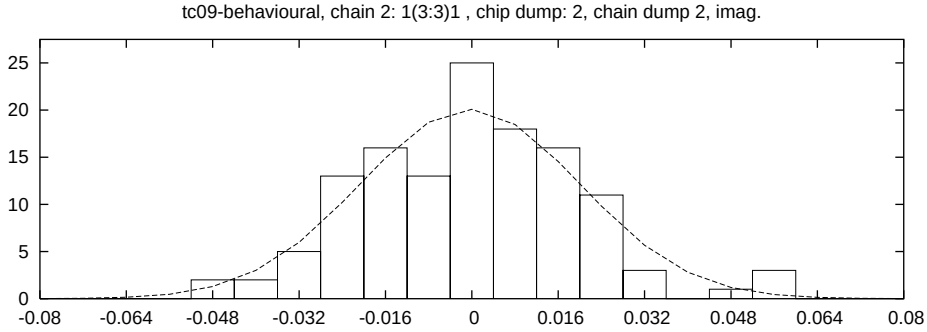

Illustration 2.5.140

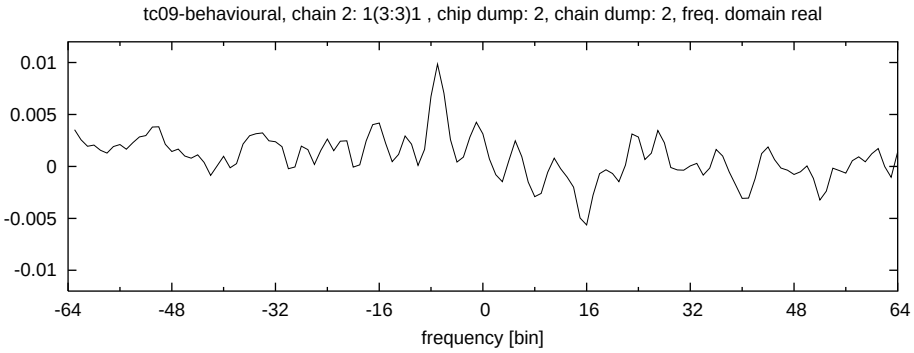

Illustration 2.5.141

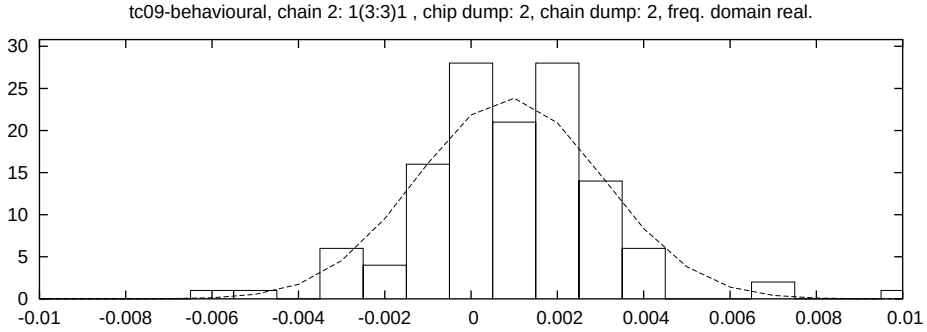

Illustration 2.5.142

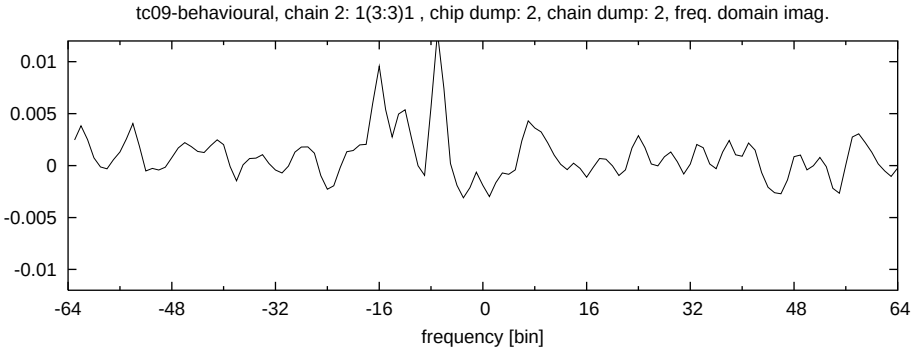

Illustration 2.5.143

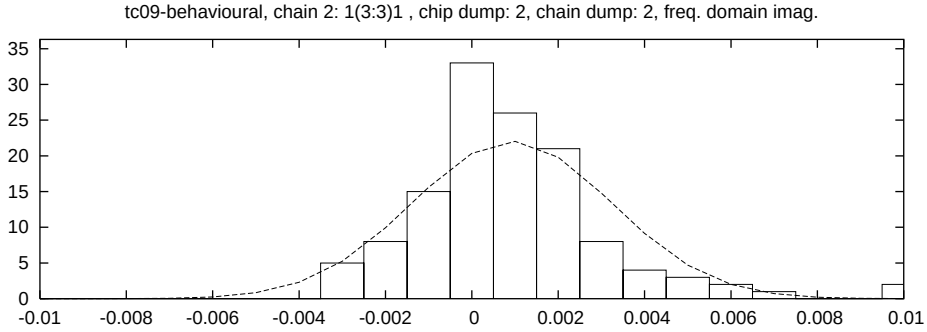

Illustration 2.5.144

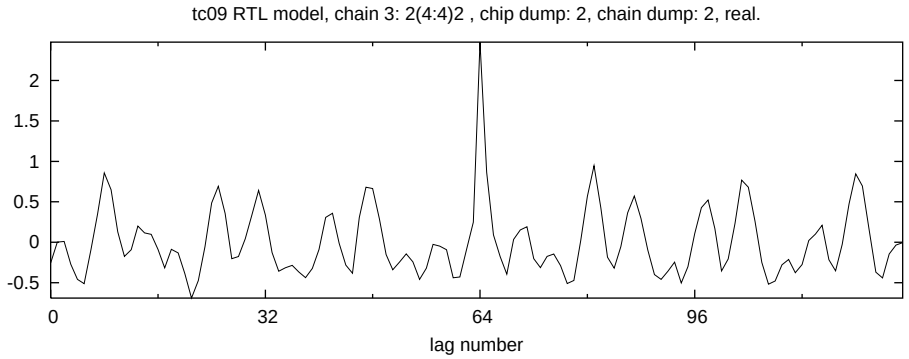

Illustration 2.5.145

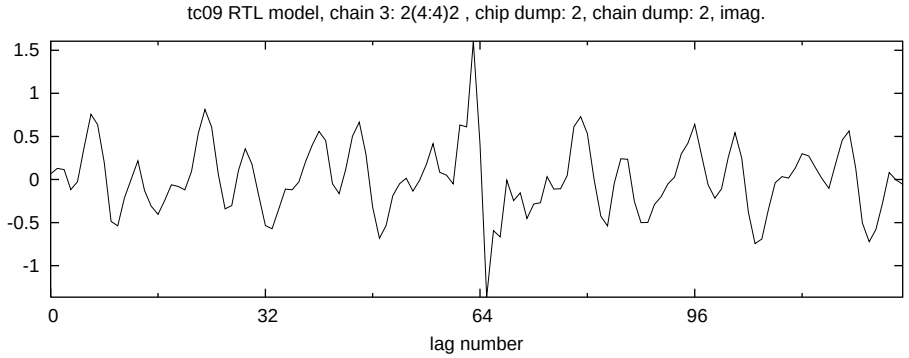

Illustration 2.5.146

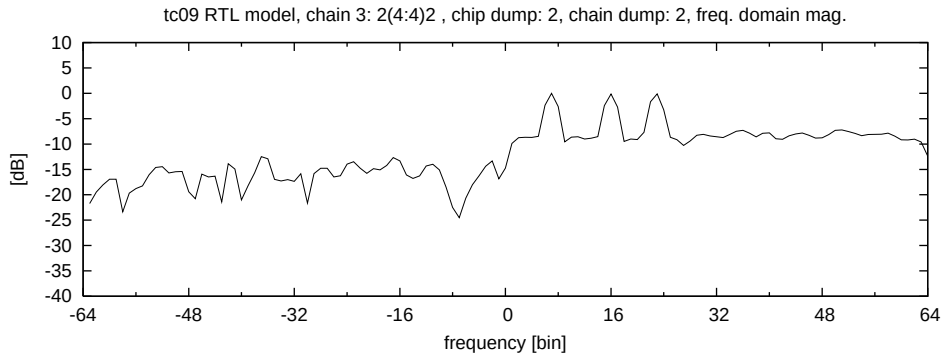

Illustration 2.5.147

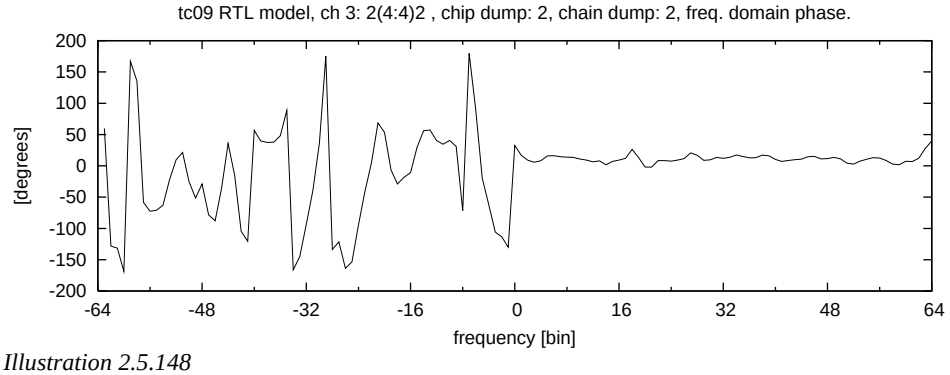

Illustration 2.5.148

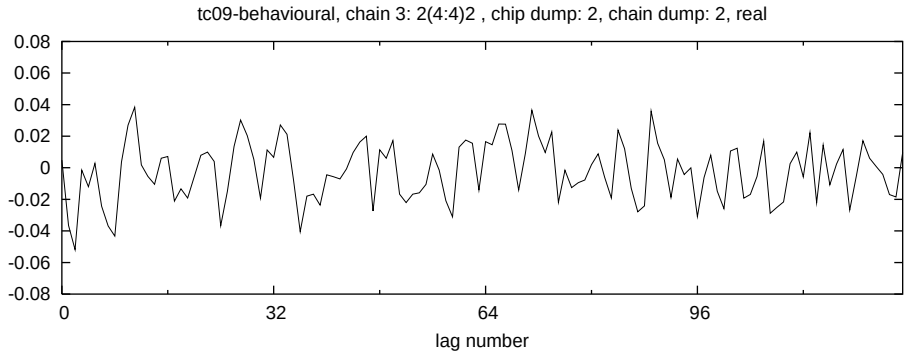

Illustration 2.5.149

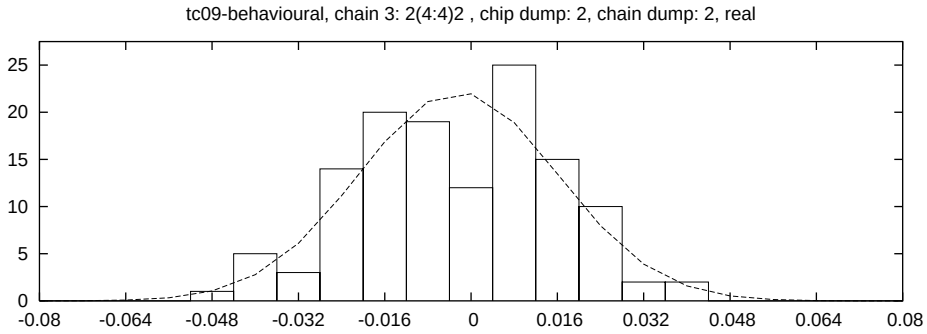

Illustration 2.5.150

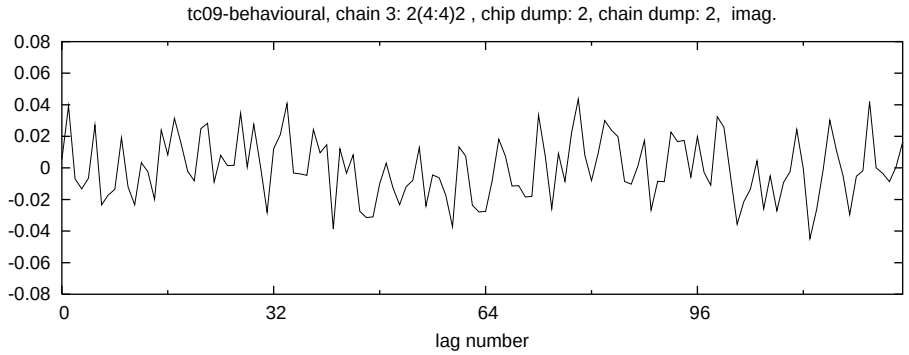

Illustration 2.5.151

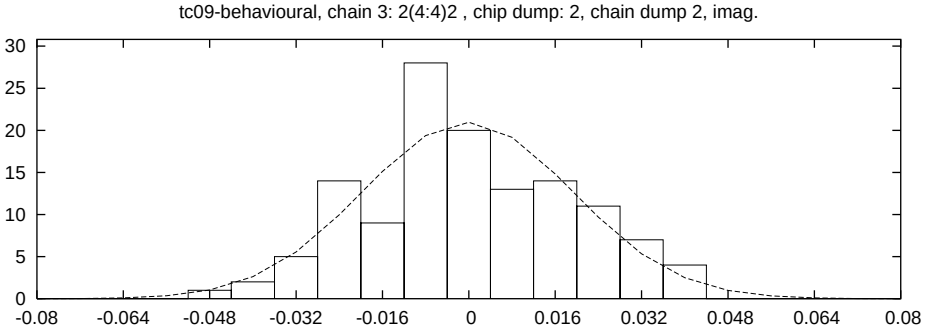

Illustration 2.5.152

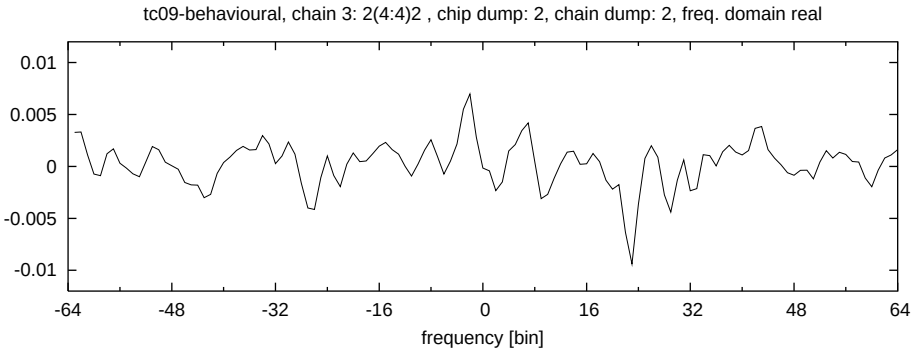

Illustration 2.5.153

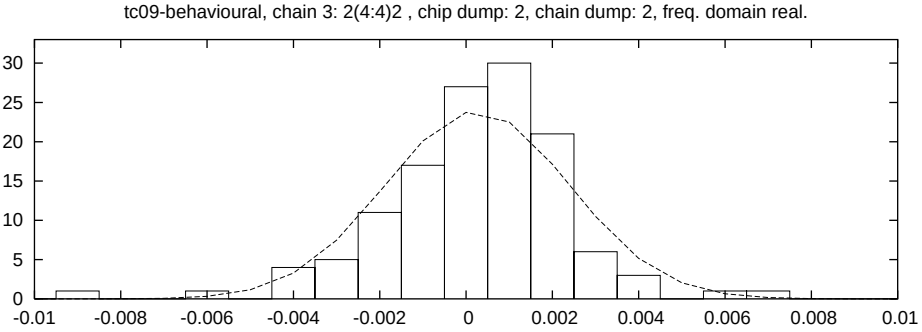

Illustration 2.5.154

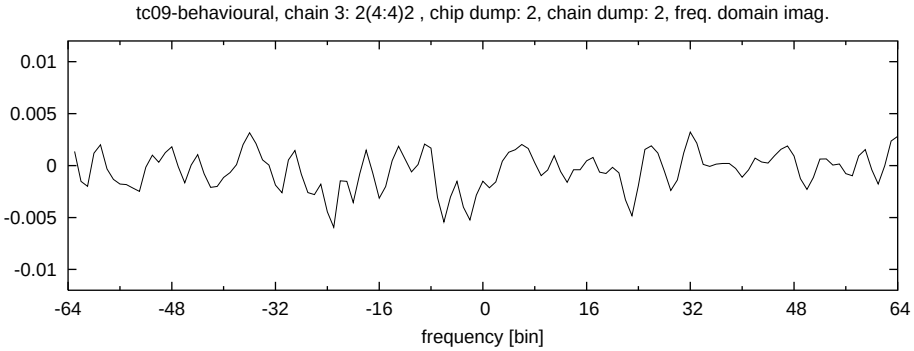

Illustration 2.5.155

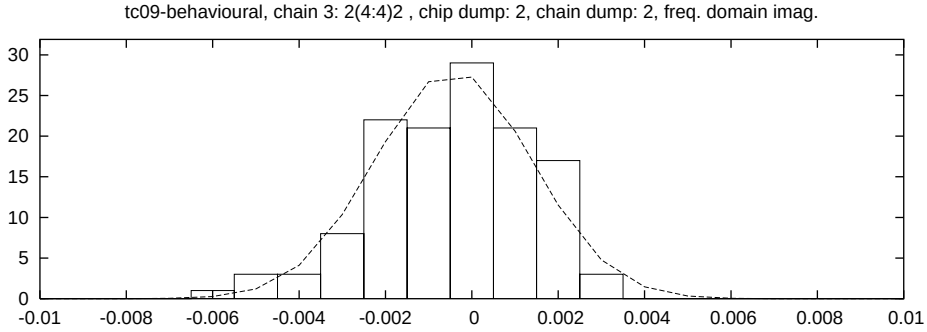

Illustration 2.5.156

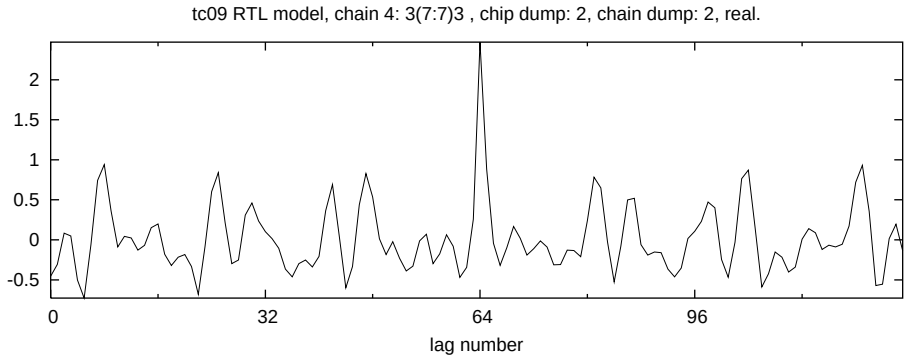

Illustration 2.5.157

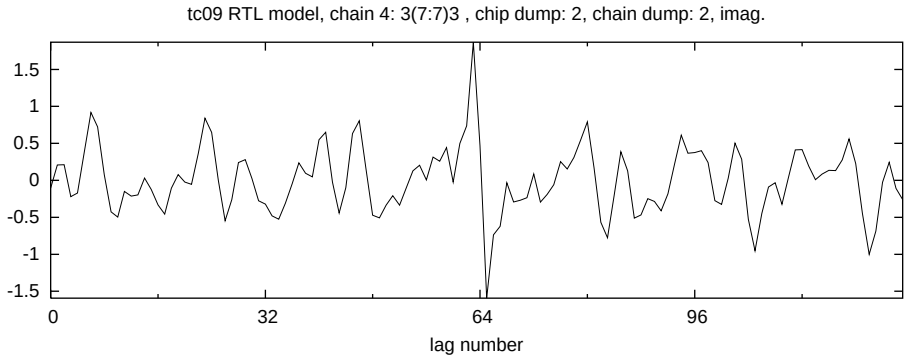

Illustration 2.5.158

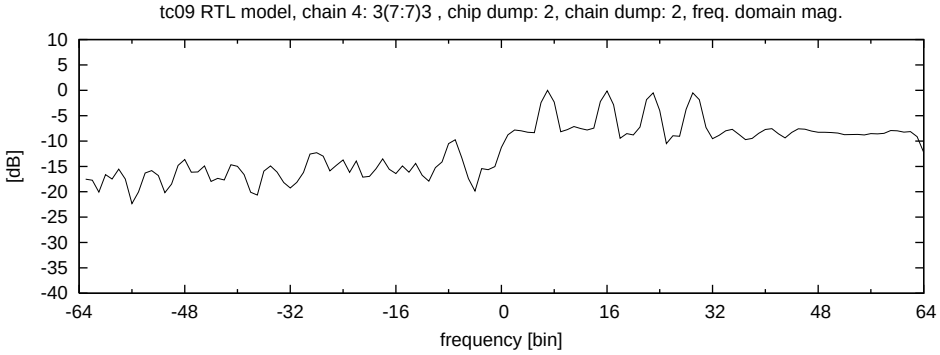

Illustration 2.5.159

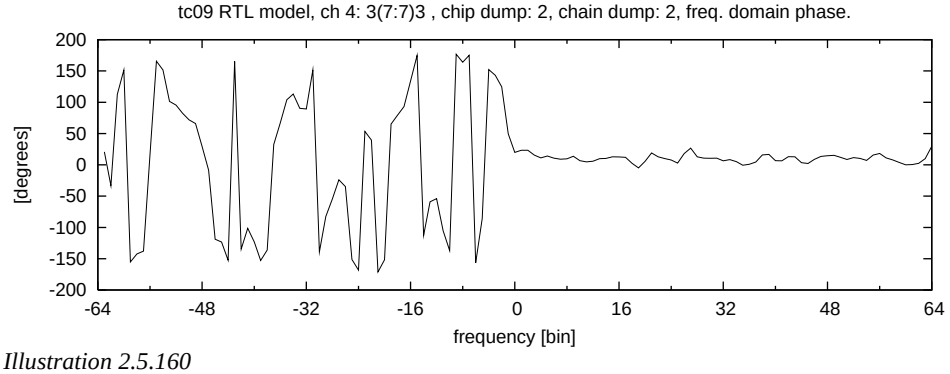

Illustration 2.5.160

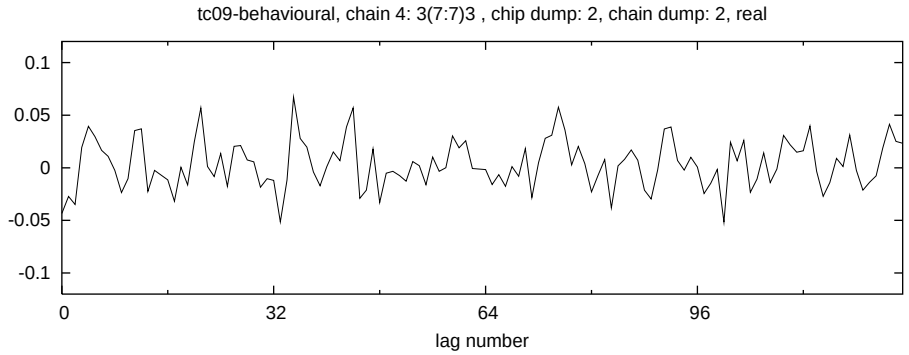

Illustration 2.5.161

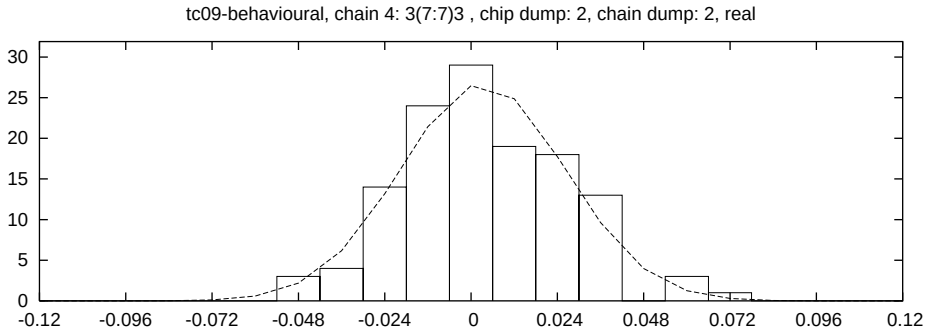

Illustration 2.5.162

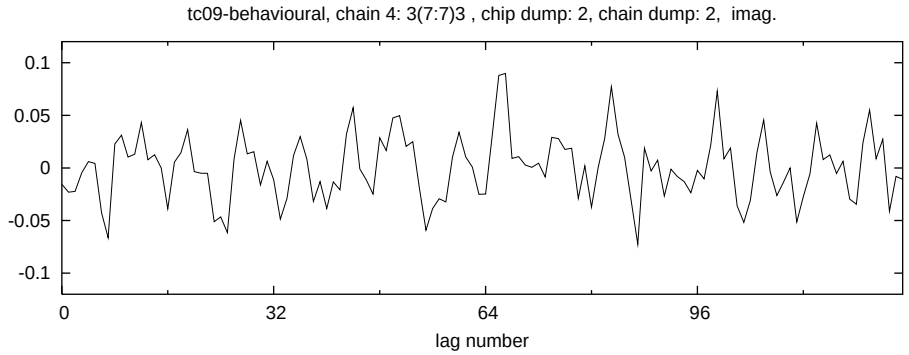

Illustration 2.5.163

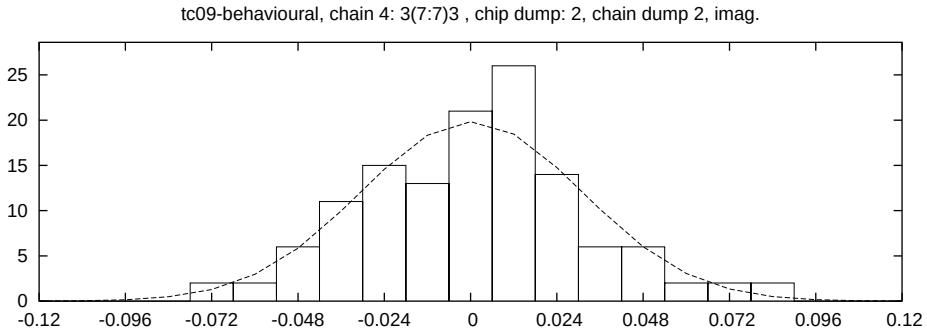

Illustration 2.5.164

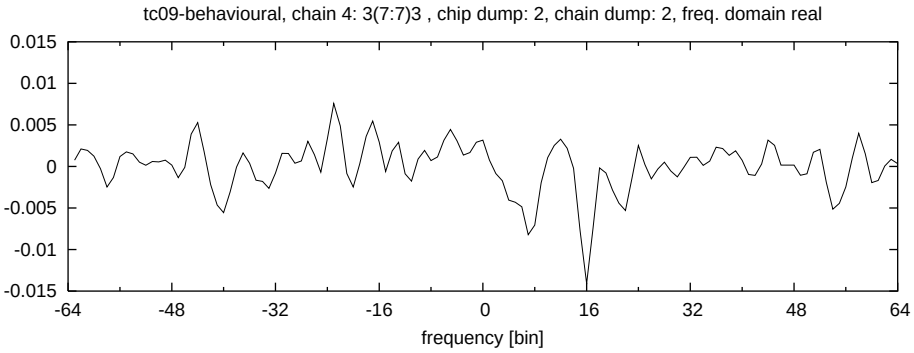

Illustration 2.5.165

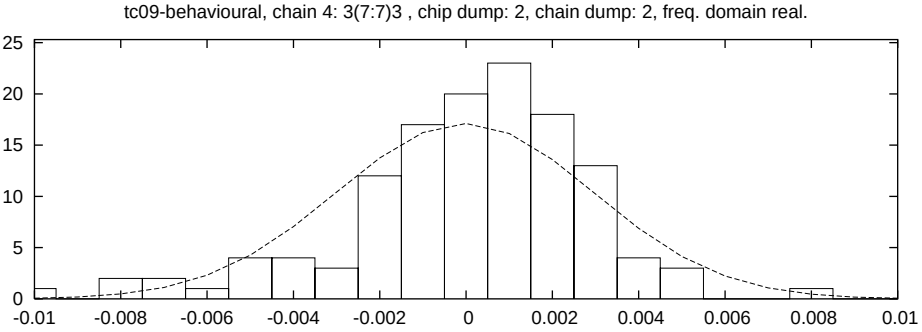

Illustration 2.5.166

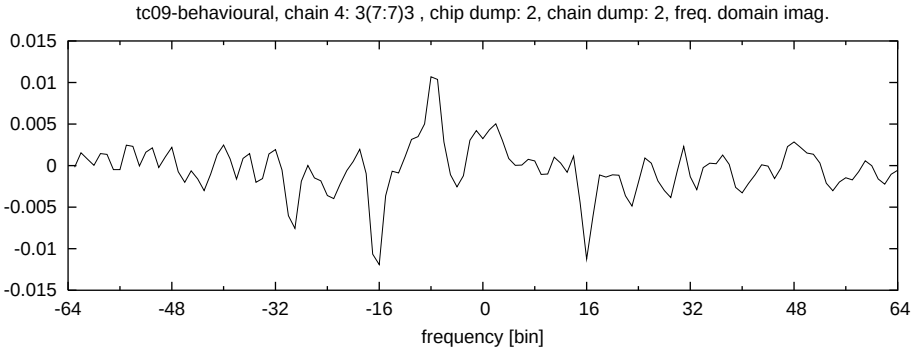

Illustration 2.5.167

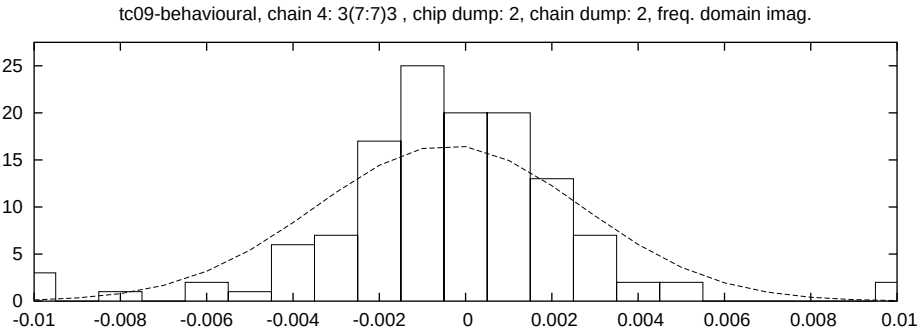

Illustration 2.5.168

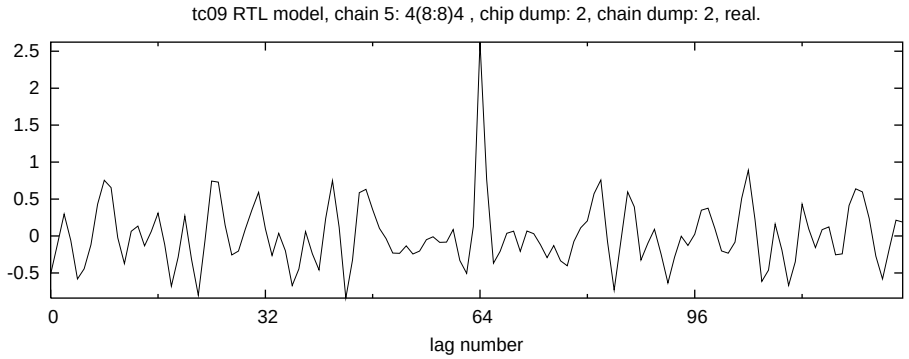

Illustration 2.5.169

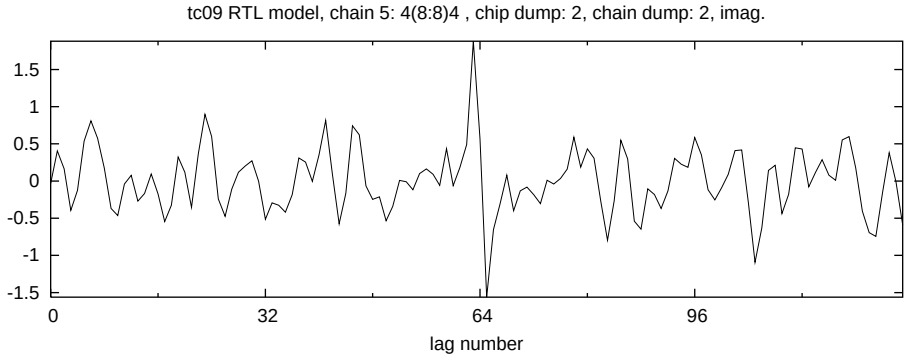

Illustration 2.5.170

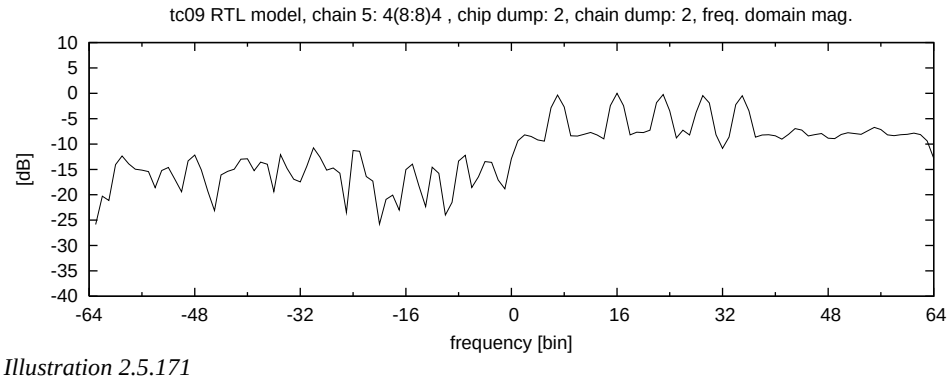

Illustration 2.5.171

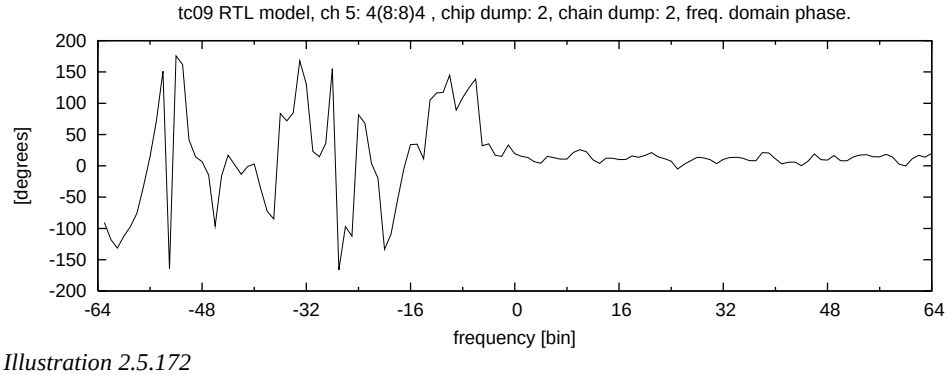

Illustration 2.5.172

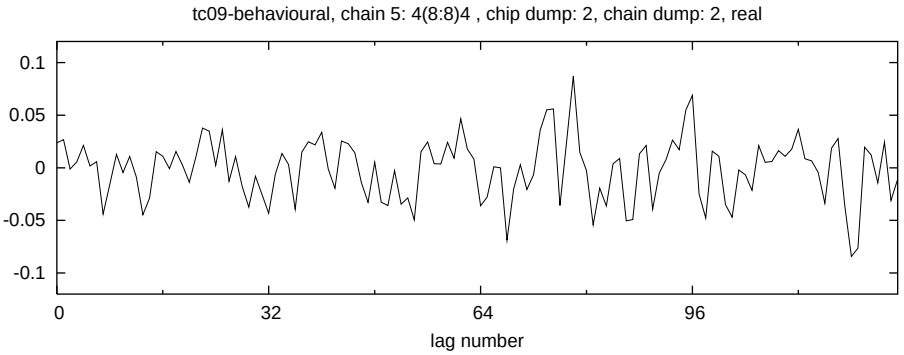

Illustration 2.5.173

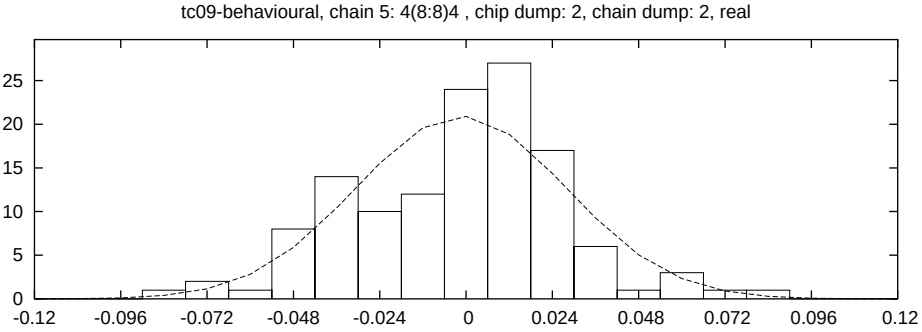

Illustration 2.5.174

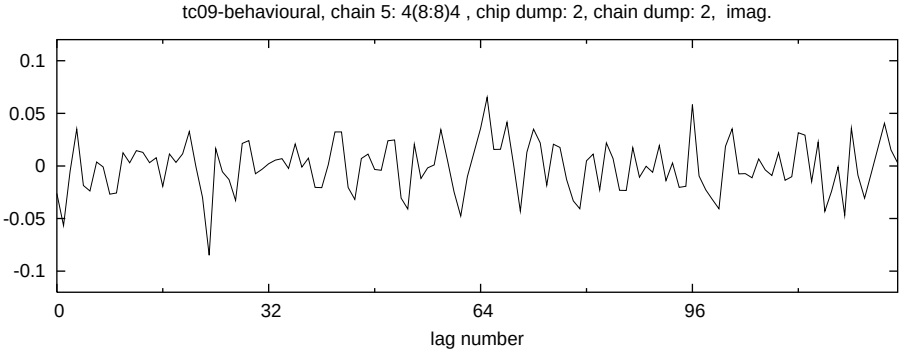

Illustration 2.5.175

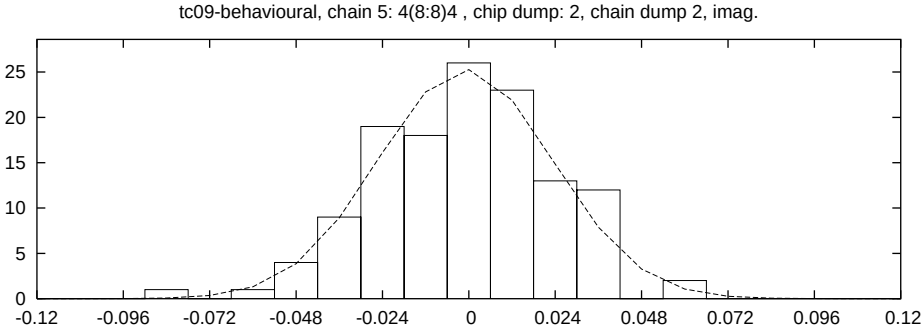

Illustration 2.5.176

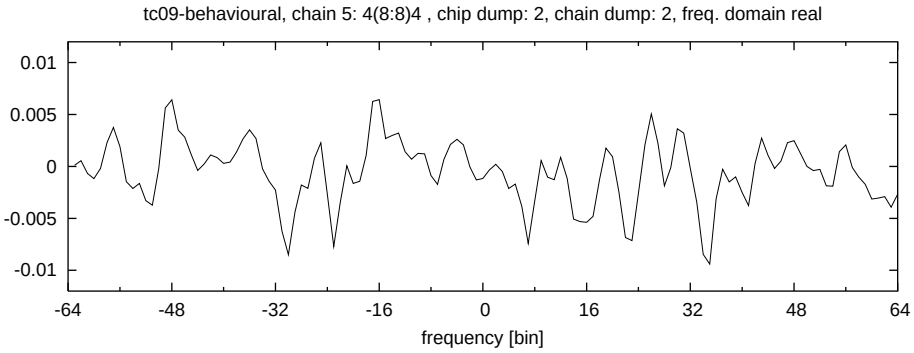

Illustration 2.5.177

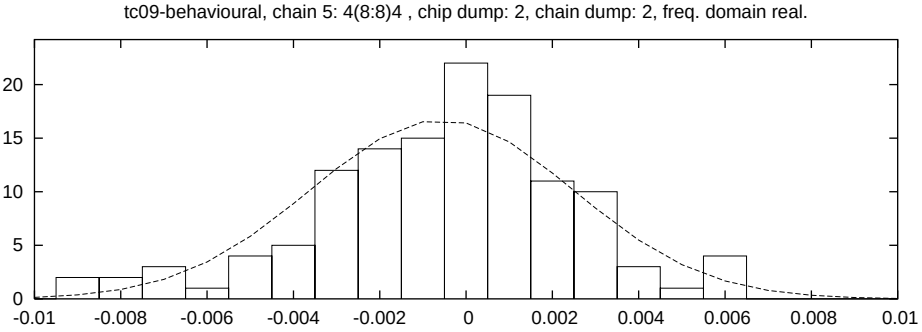

Illustration 2.5.178

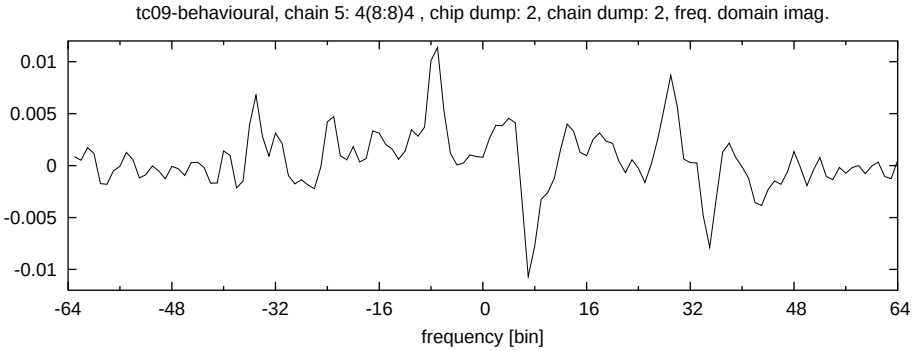

Illustration 2.5.179

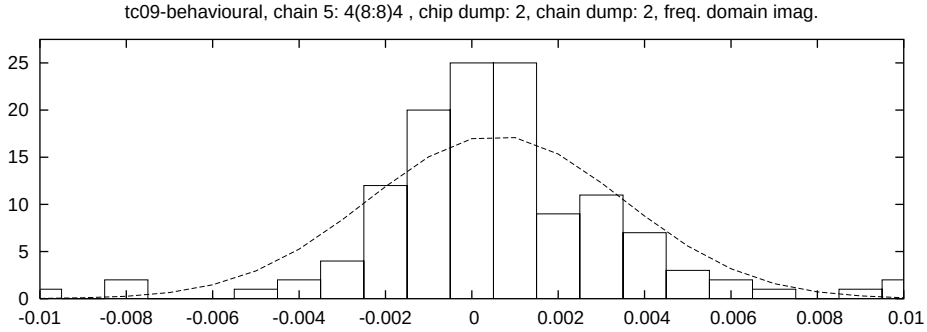

Illustration 2.5.180

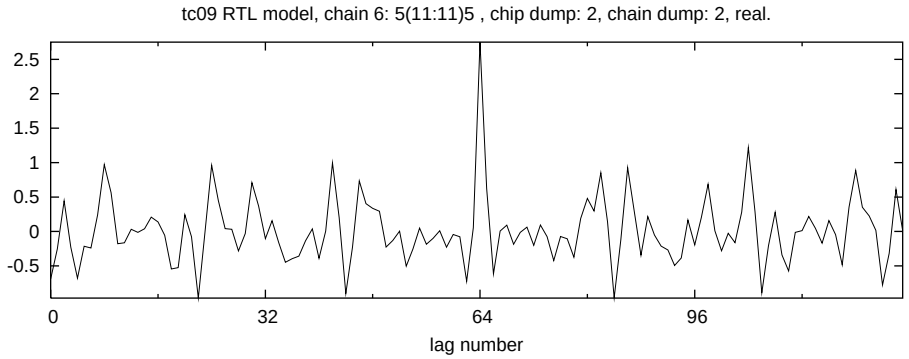

Illustration 2.5.181

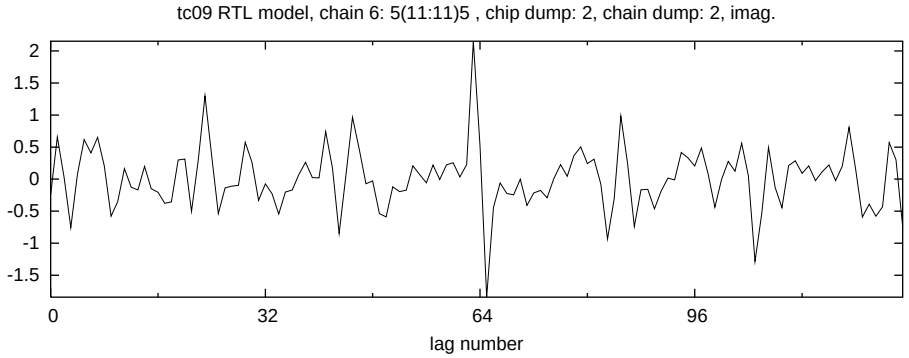

Illustration 2.5.182

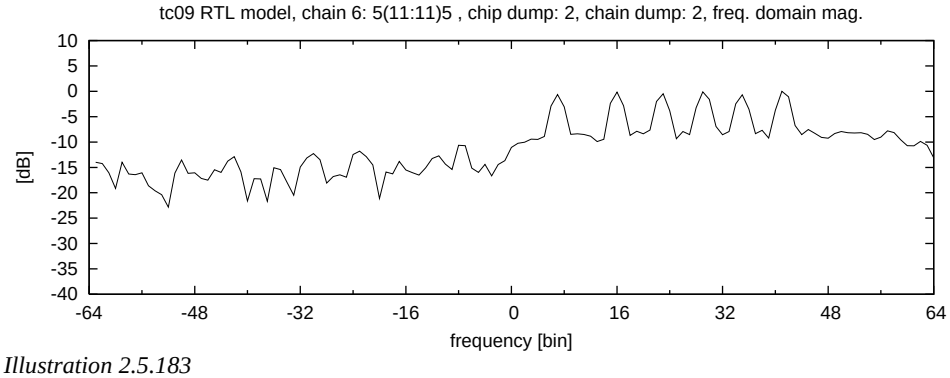

Illustration 2.5.183

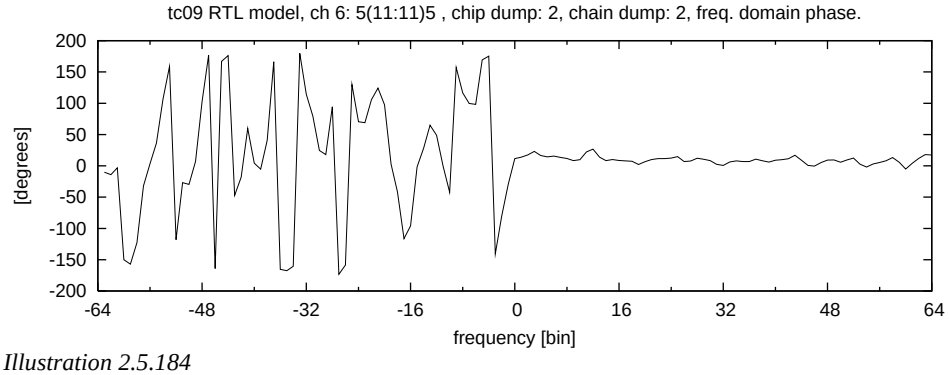

Illustration 2.5.184

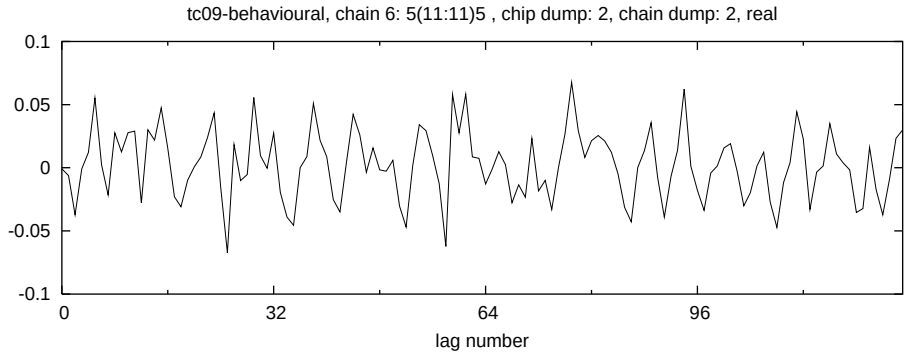

Illustration 2.5.185

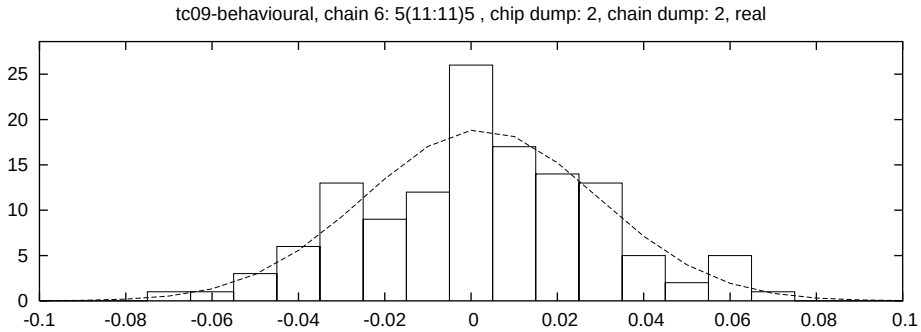

Illustration 2.5.186

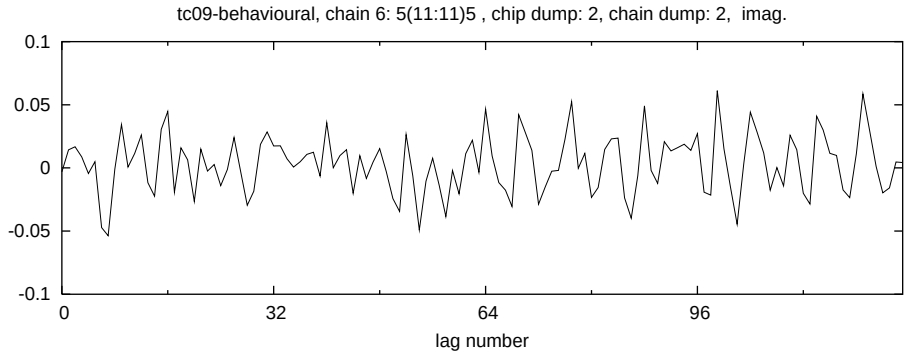

Illustration 2.5.187

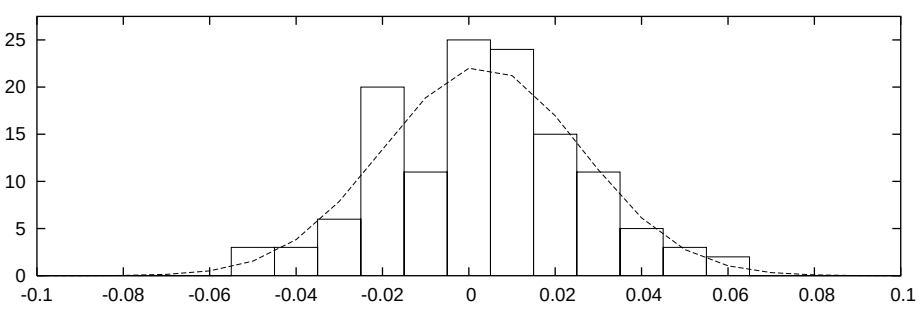

Illustration 2.5.188

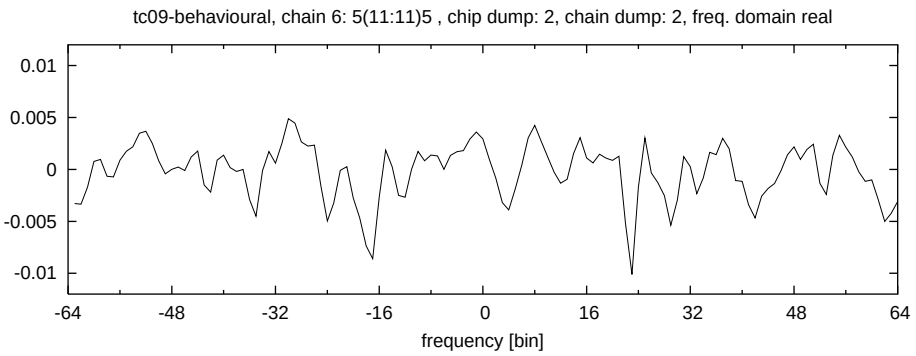

Illustration 2.5.189

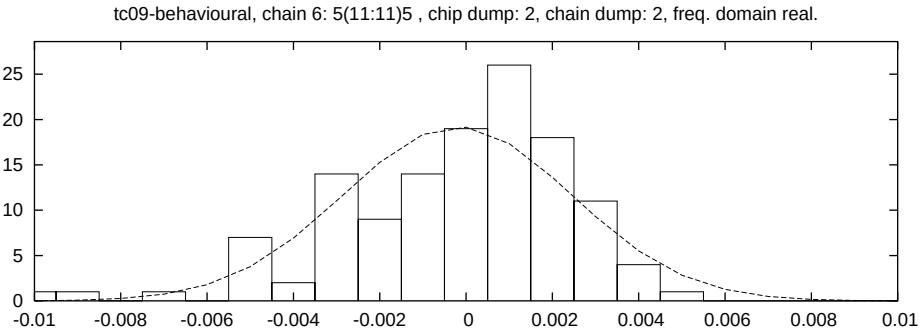

Illustration 2.5.190

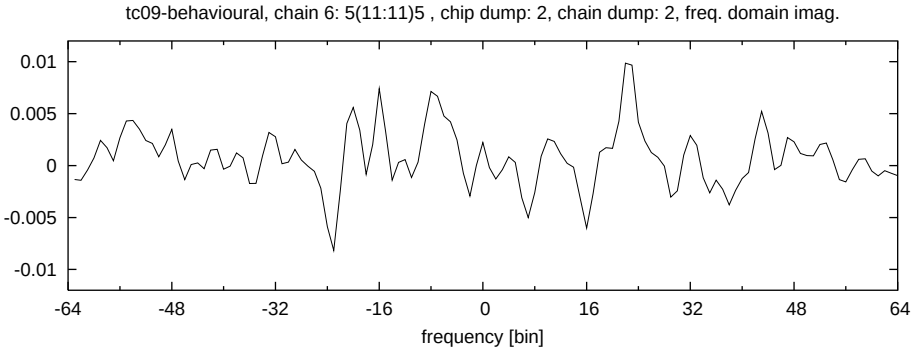

Illustration 2.5.191

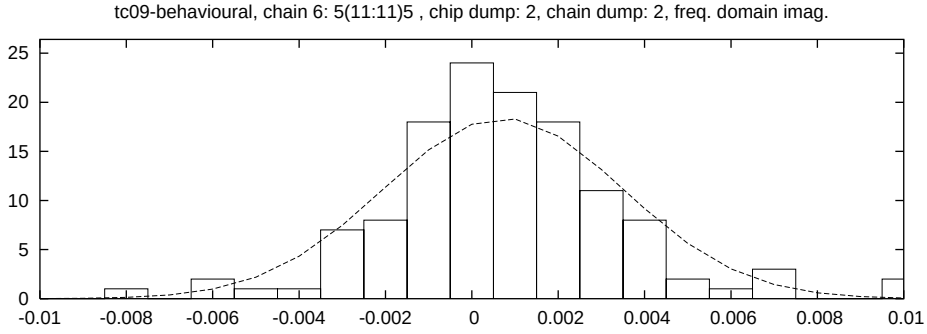

Illustration 2.5.192

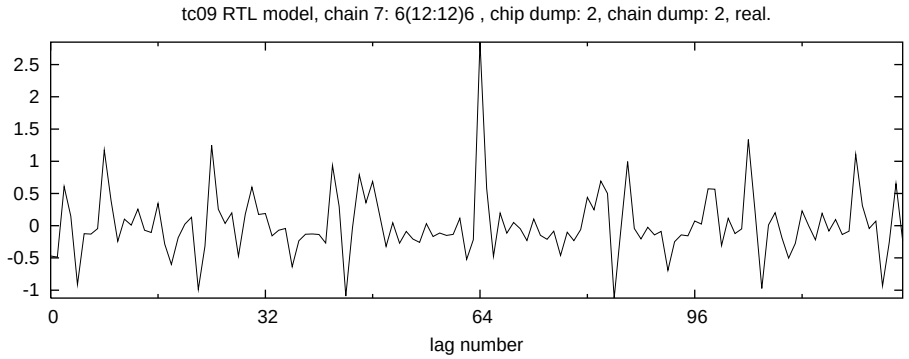

Illustration 2.5.193

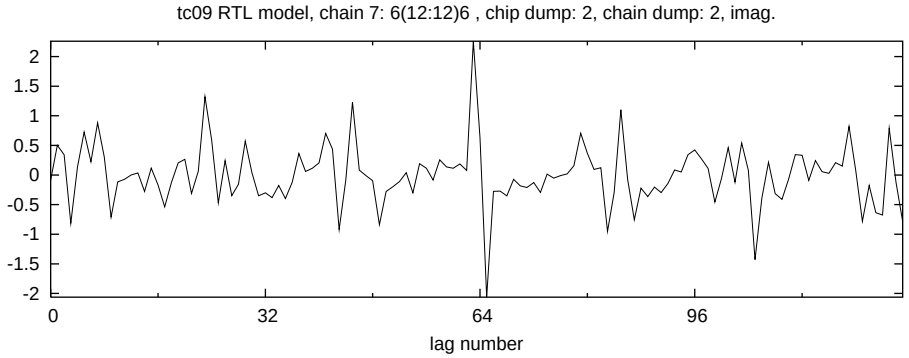

Illustration 2.5.194

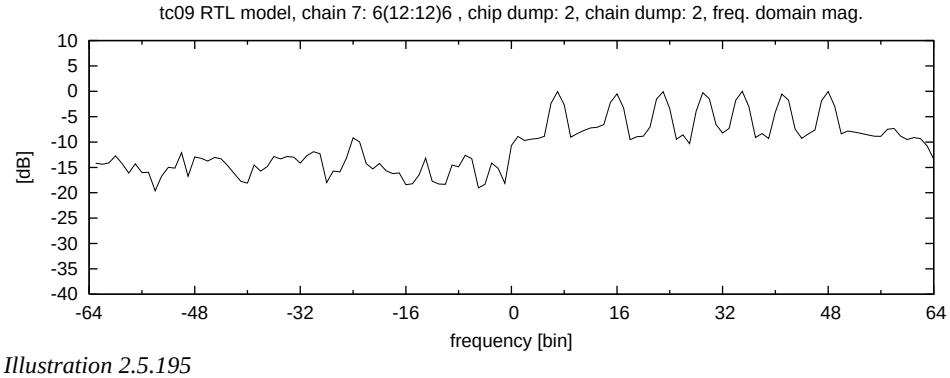

Illustration 2.5.195

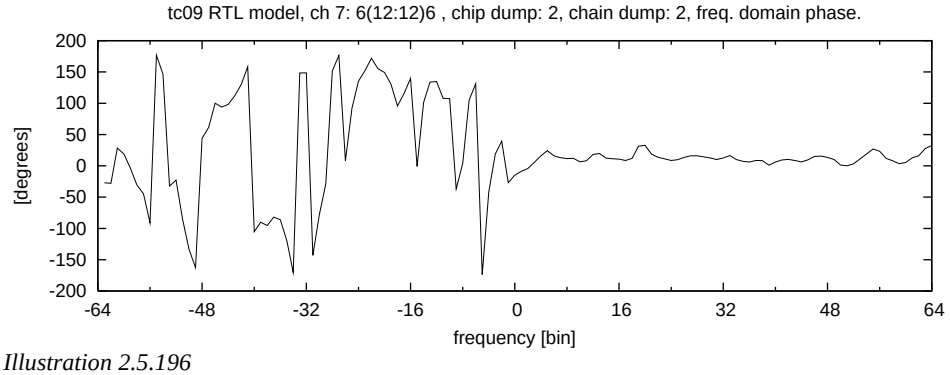

Illustration 2.5.196

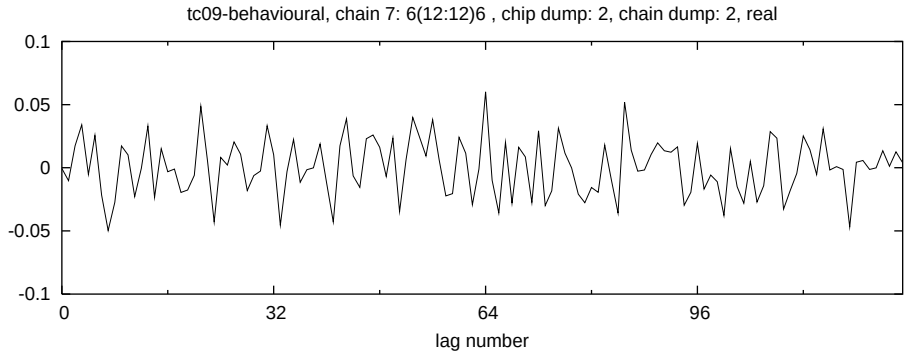

Illustration 2.5.197

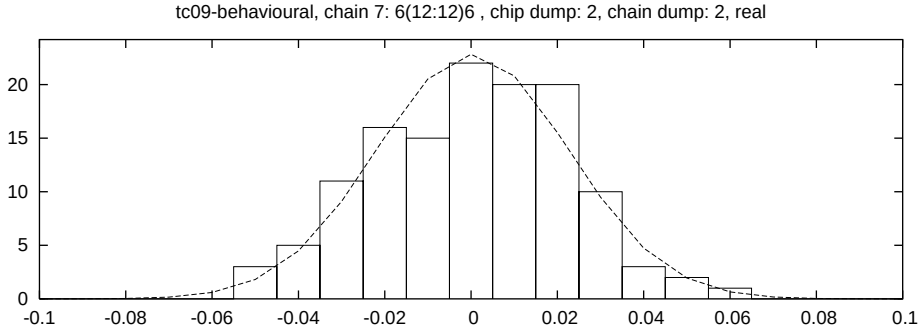

Illustration 2.5.198

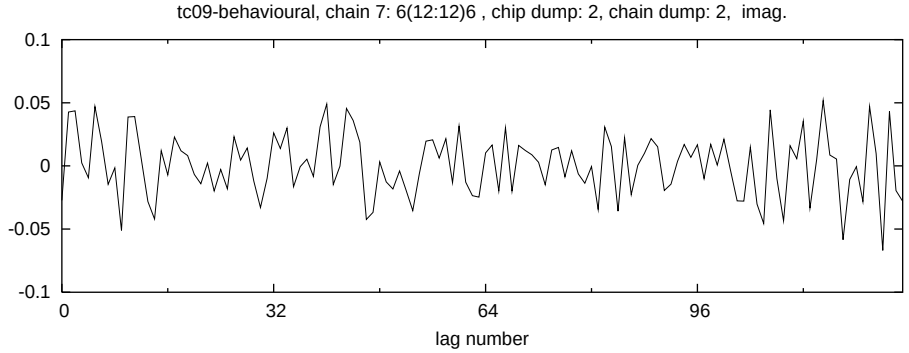

Illustration 2.5.199

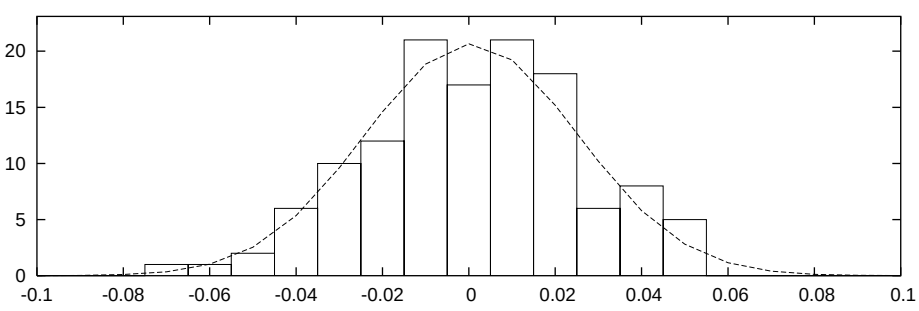

Illustration 2.5.200

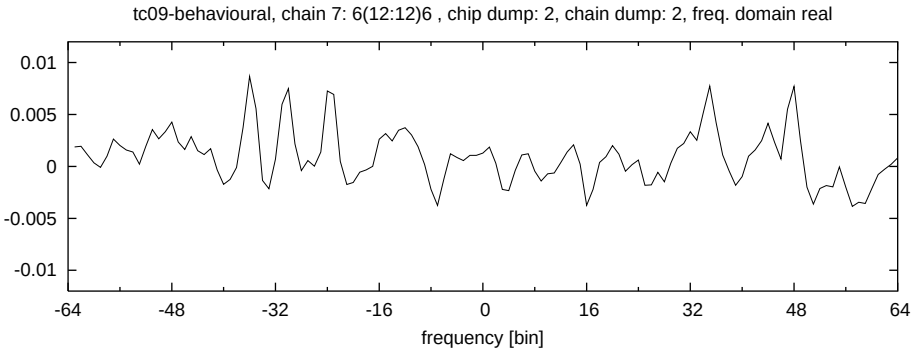

Illustration 2.5.201

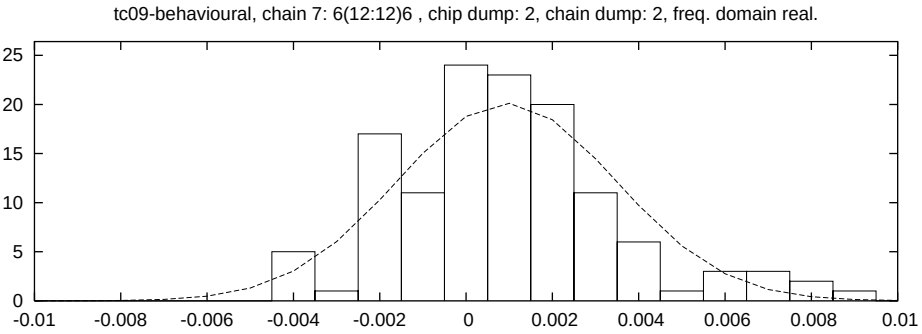

Illustration 2.5.202

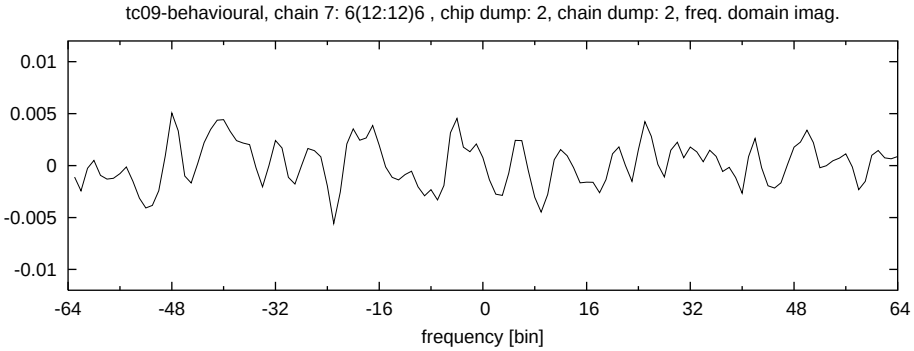

Illustration 2.5.203

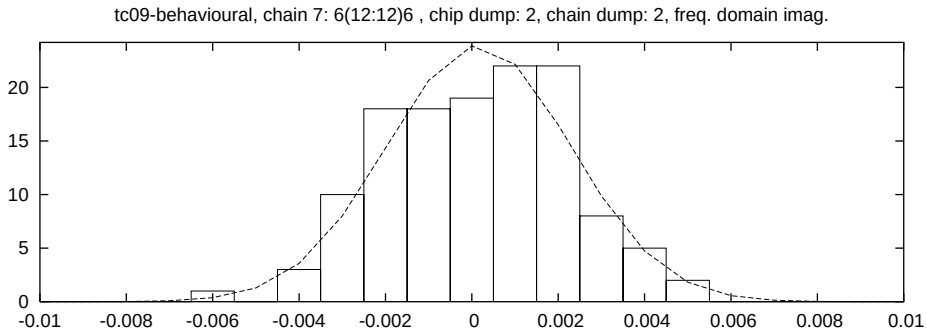

Illustration 2.5.204

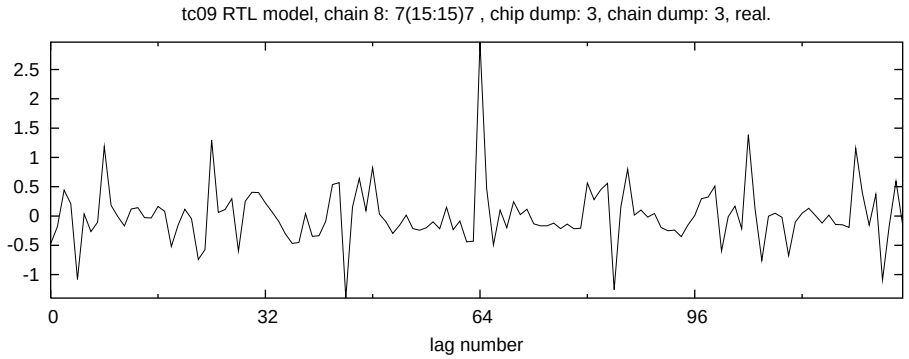

Illustration 2.5.205

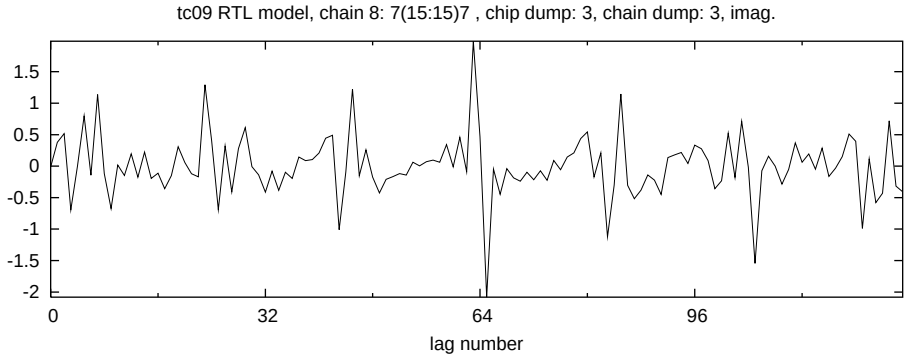

Illustration 2.5.206

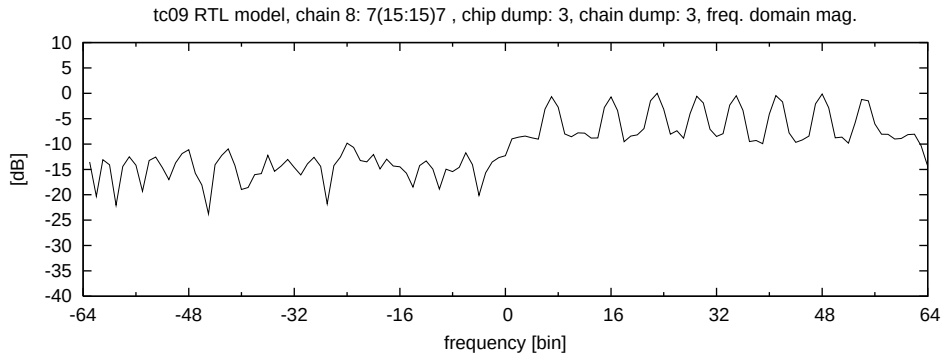

Illustration 2.5.207

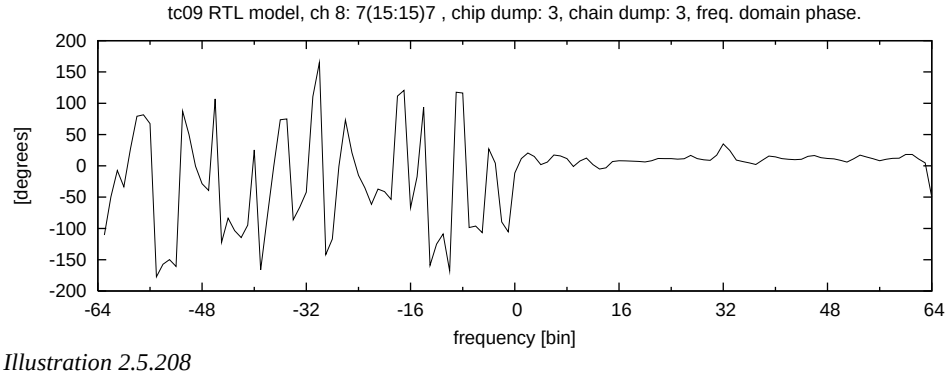

Illustration 2.5.208

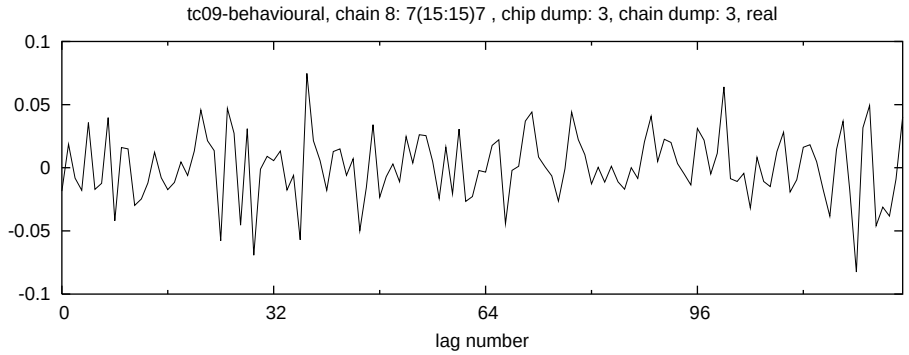

Illustration 2.5.209

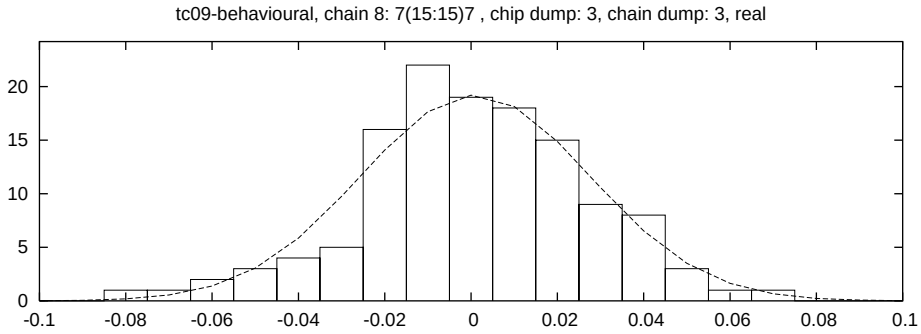

Illustration 2.5.210

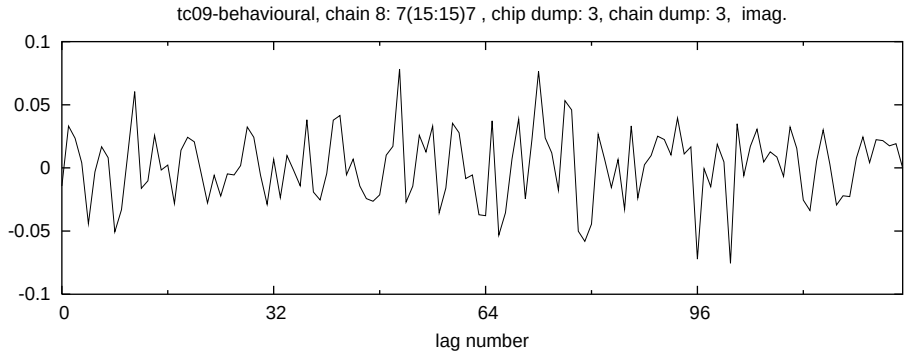

Illustration 2.5.211

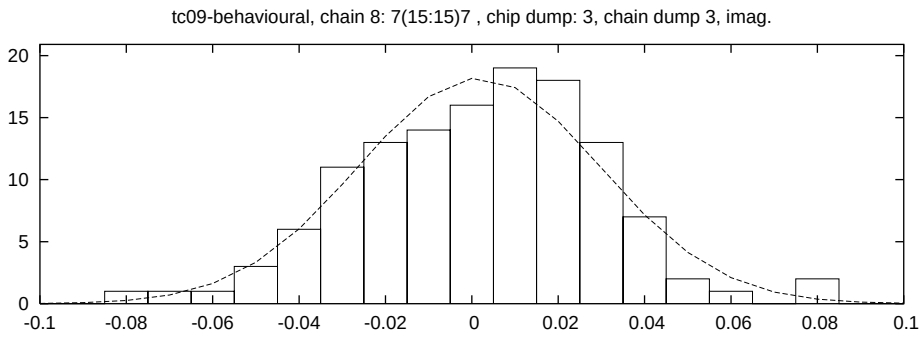

Illustration 2.5.212

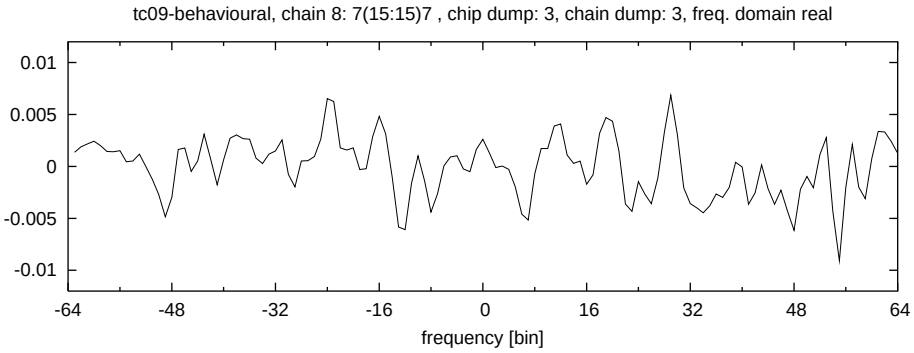

Illustration 2.5.213

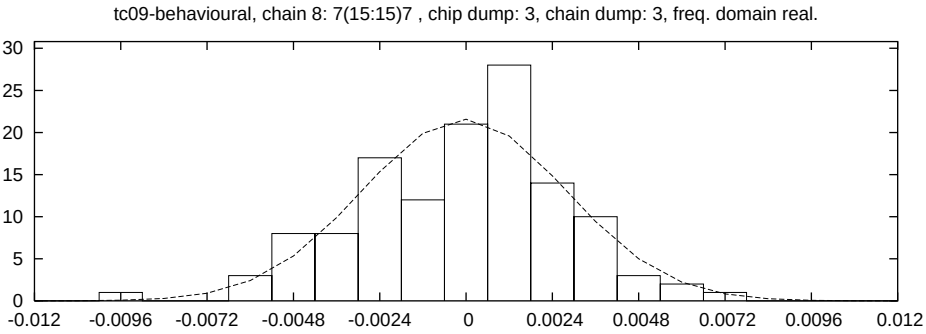

Illustration 2.5.214

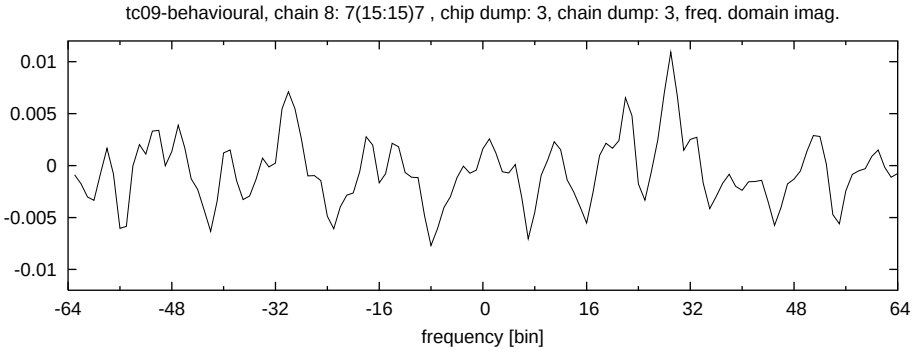

Illustration 2.5.215

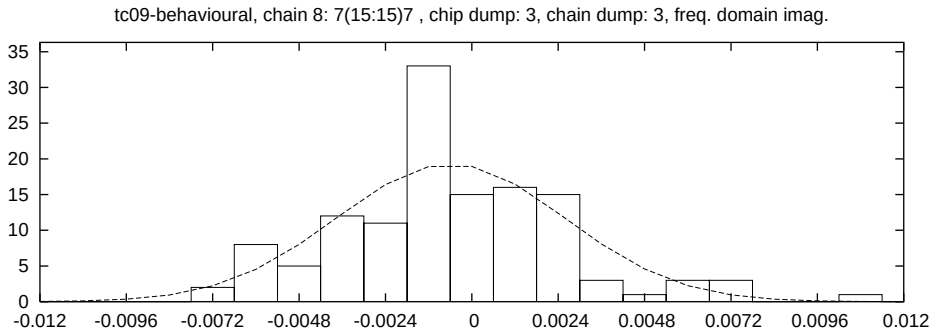

Illustration 2.5.216

2.5.8 test case 10

Synopsis: No extraordinary table entries (RTL and behavioural model difference) were noted and no advisories or warnings regarding captured data or register content were logged. A second set of model difference tables, created by applying a 12000 sample test vector to all chains of the behavioural model, is provided as a demonstration of statistical parameter sensitivity to a mismatch of model input data.

Notes: chain 2 of the RTL model dumps at 3.33 times the standard 12000 sample dump interval but the shift enable clock (**SE_CLK**) runs at modulo 3, hence, the statistics between the two sets of tables provided should be similar for this chain (at least for the first dump); this is observed. The standard deviation ratio of RTL and behavioural models between adjacent dumps columns 9 and 10 of tables Table 2.5.27 and Table 2.5.31 is seen to be a sensitive indicator of mismatched model data.

```
***--------------------------------------------------------------------------***
tc10_postsim.m : V2.00 run time: Sat Sep 03 12:49:29 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 10.
test case version: v3,05jul24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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

WARNING from make_dump_index: Times stamp rollover detected.
 Assume rollover in ordering.
ans = 91

Initial (dump= 1) timestamp (word 0): 4294967295
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc10
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***

octave:15> tc10_postsim_validation_1
Generating a second table of model lag and frequency domain
difference statistics with 12000 samples of the behavioural
model for all chains; using name: tc10a
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.
```

Text 2.5.18 console log for scripts tc10_postsim, tc10_postsim_validation_1.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	-2.02e-04	1.77e-04	0.0794	0.1052	0.0148	0.0149	0.96	0.96	2	
1	2	5.91e-04	-3.38e-04	0.0815	0.1047	0.0131	0.0168	0.85	1.08	3	
2	1	1.04e-03	1.48e-04	0.1019	0.0904	0.0135	0.0170	0.71	0.90	2	
2	2	3.05e-04	2.67e-04	0.0959	0.1183	0.0170	0.0189	0.90	1.00	1	
3	1	1.27e-03	3.65e-04	0.1195	0.1168	0.0185	0.0207	0.84	0.94	3	
3	2	-8.31e-04	2.61e-04	0.1028	0.0898	0.0186	0.0173	0.85	0.79	0	
4	1	5.71e-04	-1.66e-03	0.1284	0.1056	0.0203	0.0197	0.83	0.80	1	
4	2	1.20e-03	6.44e-04	0.1416	0.1706	0.0241	0.0292	0.98	1.19	3	
5	1	7.03e-04	-1.15e-05	0.1081	0.1301	0.0197	0.0206	0.73	0.77	1	
5	2	-9.48e-04	1.43e-04	0.1337	0.1450	0.0264	0.0226	0.98	0.84	1	
6	1	4.23e-05	2.73e-04	0.1274	0.1409	0.0235	0.0221	0.81	0.76	2	
6	2	1.37e-03	1.09e-03	0.1644	0.1576	0.0264	0.0258	0.91	0.89	2	
7	1	4.79e-04	-8.64e-04	0.1180	0.1385	0.0254	0.0271	0.82	0.88	0	
7	2	-5.30e-04	1.18e-03	0.1468	0.1186	0.0234	0.0224	0.76	0.72	1	
8	1	-6.51e-04	1.72e-03	0.1155	0.1289	0.0211	0.0232	0.64	0.70	1	
8	2	2.11e-04	3.86e-04	0.1384	0.1444	0.0262	0.0249	0.80	0.76	1	

Table 2.5.26 test case 10; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-1.06e-03	6.75e-03	0.4117	0.3592	0.0647	0.0601	0.98	0.84	128
2	1	-4.70e-03	-6.17e-04	0.4545	0.5064	0.0769	0.0847	1.05	1.06	128
3	1	-9.77e-03	7.56e-03	0.4977	0.4256	0.0895	0.0855	1.04	0.99	null
4	1	-8.08e-04	2.43e-03	0.5204	0.5626	0.0846	0.1035	0.94	0.97	null
5	1	-5.00e-03	-4.86e-03	0.4912	0.5125	0.0964	0.0944	1.01	0.97	null
6	1	3.43e-03	-6.95e-03	0.5860	0.7122	0.1037	0.1126	0.95	0.96	null
7	1	-2.44e-03	-1.04e-03	0.6523	0.5636	0.1162	0.1086	1.01	0.97	null
8	1	-5.72e-04	2.56e-03	0.7281	0.6938	0.1278	0.1289	1.00	0.99	null

Table 2.5.27 test case 10; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	-7.90e-06	5.84e-04	0.0068	0.0147	9.04e-04	1.33e-03	1.9	2.9	0	
1	2	4.94e-06	-5.18e-04	0.0064	0.0143	9.00e-04	1.37e-03	1.9	2.9	0	
2	1	4.18e-04	-1.31e-05	0.0150	0.0040	1.43e-03	8.66e-04	2.5	1.5	0	
2	2	3.15e-04	5.29e-04	0.0104	0.0133	1.27e-03	1.57e-03	2.2	2.8	0	
3	1	-1.35e-04	4.30e-05	0.0149	0.0120	1.66e-03	1.37e-03	2.5	2.1	0	
3	2	3.09e-04	-9.31e-05	0.0125	0.0088	1.45e-03	1.24e-03	2.2	1.9	0	
4	1	-7.86e-05	-1.65e-04	0.0133	0.0102	1.59e-03	1.36e-03	2.2	1.8	0	
4	2	9.55e-05	-1.04e-04	0.0213	0.0189	2.24e-03	2.02e-03	3.1	2.7	0	
5	1	-9.71e-05	-1.02e-04	0.0175	0.0087	1.75e-03	1.36e-03	2.2	1.7	0	
5	2	-1.72e-04	9.78e-05	0.0147	0.0191	1.94e-03	2.05e-03	2.4	2.5	0	
6	1	2.41e-04	-1.74e-04	0.0147	0.0093	2.03e-03	1.52e-03	2.3	1.7	0	
6	2	-4.38e-05	1.74e-04	0.0143	0.0162	1.99e-03	1.83e-03	2.3	2.1	0	
7	1	-1.96e-04	-3.42e-04	0.0146	0.0107	2.12e-03	1.72e-03	2.3	1.8	0	
7	2	2.80e-04	3.28e-04	0.0195	0.0085	1.89e-03	1.46e-03	2.0	1.6	0	
8	1	3.94e-05	1.54e-04	0.0121	0.0092	1.69e-03	1.62e-03	1.7	1.6	0	
8	2	-3.66e-05	-1.10e-04	0.0142	0.0136	1.93e-03	1.87e-03	2.0	1.9	0	

Table 2.5.28 test case 10; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: ! OVR*	*bias: OVR*	*applied dump - bias: OVR*
0	40000	39724	276	0	n-a
1	40000	39724	276	0	n-a
2	40000	39724	276	0	n-a
3	40000	39724	276	0	n-a
4	40000	39724	276	0	n-a
5	40000	39724	276	0	n-a
6	40000	39724	276	0	n-a
7	40000	39724	276	0	n-a

Table 2.5.29 test case 10: applied dump interval and bias.

A second set of tables, Table 2.5.30 to Table 2.5.32 inclusive, is generated for test case 10 using the behavioural model data of test case 11. This exercise serves to validate the tables as an indicator of model deviation from expected behaviour. The model of test case 11 uses 12000 samples for all chains.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*
1	1	-2.39e-03	2.15e-04	0.2454	0.2505	0.0455	0.0445	2.27	2.22	0
1	2	3.27e-03	7.12e-03	0.3241	0.3333	0.0655	0.0582	3.28	2.91	1
2	1	7.29e-04	6.60e-04	0.1334	0.1287	0.0204	0.0236	1.02	1.18	3
2	2	1.18e-04	-5.55e-04	0.2160	0.1904	0.0365	0.0357	1.82	1.79	1
3	1	1.88e-03	-5.54e-04	0.1787	0.1479	0.0294	0.0264	1.47	1.32	1
3	2	-3.87e-03	3.08e-03	0.2277	0.2712	0.0457	0.0492	2.28	2.46	0
4	1	5.36e-04	-4.35e-04	0.1981	0.2277	0.0365	0.0432	1.83	2.16	0
4	2	4.48e-03	-1.41e-03	0.4303	0.3824	0.0738	0.0726	3.69	3.63	1
5	1	3.64e-03	3.38e-03	0.2429	0.2512	0.0456	0.0455	2.28	2.28	0
5	2	-4.50e-03	-2.80e-03	0.4723	0.4700	0.0826	0.0800	4.13	4.00	1
6	1	-1.24e-03	4.09e-03	0.2867	0.3633	0.0541	0.0590	2.70	2.95	3
6	2	5.14e-03	2.76e-03	0.4598	0.5969	0.0890	0.0927	4.45	4.64	1
7	1	1.56e-03	-2.07e-04	0.3603	0.3519	0.0637	0.0614	3.18	3.07	1
7	2	-9.38e-04	9.25e-04	0.4904	0.4991	0.0916	0.0917	4.58	4.58	0
8	1	8.09e-04	9.07e-04	0.4228	0.4003	0.0731	0.0756	3.66	3.78	2
8	2	3.48e-04	-1.25e-04	0.5169	0.5209	0.0936	0.0952	4.68	4.76	1

Table 2.5.30 test case 10; test validation; statistical parameters of the difference between RTL and behavioural model lag coefficients. A 12000 sample test vector was applied as input to all chains of the behavioural model.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-1.06e-03	6.75e-03	0.4117	0.3592	0.0647	0.0601	0.78	0.66	null
2	1	-4.70e-03	-6.17e-04	0.4545	0.5064	0.0769	0.0847	1.03	0.98	null
3	1	-9.77e-03	7.56e-03	0.4977	0.4256	0.0895	0.0855	1.15	1.11	null
4	1	-8.08e-04	2.43e-03	0.5204	0.5626	0.0846	0.1035	1.12	1.30	null
5	1	-5.00e-03	-4.86e-03	0.4912	0.5125	0.0964	0.0944	1.12	1.33	null
6	1	3.43e-03	-6.95e-03	0.5860	0.7122	0.1037	0.1126	1.63	1.50	null
7	1	-2.44e-03	-1.04e-03	0.6523	0.5636	0.1162	0.1086	1.46	1.58	null
8	1	-5.72e-04	2.56e-03	0.7281	0.6938	0.1278	0.1289	1.81	1.85	null

Table 2.5.31 test case 10; test validation; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients. A 12000 sample test vector was applied as input to all chains of the behavioural model.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*
1	1	4.45e-04	1.02e-03	0.0299	0.0334	3.38e-03	3.53e-03	5.6	5.9	0
1	2	-9.59e-04	1.26e-04	0.0327	0.0205	5.21e-03	4.30e-03	8.7	7.2	0
2	1	3.11e-04	2.07e-05	0.0143	0.0109	1.90e-03	1.52e-03	3.2	2.5	0
2	2	4.94e-04	5.03e-04	0.0164	0.0302	2.66e-03	3.04e-03	4.4	5.1	0
3	1	-6.63e-05	-1.57e-05	0.0192	0.0121	2.38e-03	1.85e-03	4.0	3.1	0
3	2	-1.23e-04	5.98e-05	0.0343	0.0270	4.15e-03	3.32e-03	6.9	5.5	0
4	1	2.21e-04	3.32e-04	0.0350	0.0153	3.82e-03	2.56e-03	6.4	4.3	0
4	2	-4.68e-04	4.35e-05	0.0518	0.0324	6.34e-03	4.64e-03	10.6	7.7	0
5	1	-1.45e-04	-6.59e-04	0.0299	0.0218	3.79e-03	3.16e-03	6.3	5.3	0
5	2	2.73e-04	1.46e-04	0.0539	0.0316	6.90e-03	5.34e-03	11.5	8.9	0
6	1	-1.30e-04	2.44e-04	0.0418	0.0279	4.74e-03	3.69e-03	7.9	6.1	0
6	2	-4.75e-04	5.01e-04	0.0758	0.0419	7.94e-03	5.92e-03	13.2	9.9	0
7	1	5.27e-04	-8.92e-04	0.0433	0.0348	5.48e-03	4.29e-03	9.1	7.1	0
7	2	3.96e-04	5.94e-04	0.0613	0.0531	7.39e-03	6.52e-03	12.3	10.9	0
8	1	-8.07e-04	8.95e-04	0.0380	0.0354	5.96e-03	4.84e-03	9.9	8.1	0
8	2	-8.46e-04	-6.27e-04	0.0611	0.0467	7.69e-03	6.78e-03	12.8	11.3	0

Table 2.5.32 test case 10; test validation; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients. A 12000 sample test vector was applied as input to all chains of the behavioural model.

```
--------------------------------------------------------------------------------
# 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;
#
# --------------------------------------------------------------------------------
00000101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.19 content of file mcb_reg10.txt.

2.5.9 test case 11

Synopsis: No extraordinary table entries are noted although warnings regarding consistency of captured data were logged as a consequence of applied input synchronization errors. Data captured from the *X* input stream that differs between cells common to a chain and triggered these warnings are tabulated in Table 2.5.33.

Notes:

(1) **XSTATUS** and **YSTATUS** registers do not report errors on **X|Y-DVALID** in any of test cases 04, 06, 11 as these test cases did not invoke the test bench facility to specifically set an error condition on these lines. The template applied to *check_register_capture* did not reflect this knowledge a priori, hence, the log (Text 2.5.22) contains a warning regarding unexpected **X|YDVALID** states.

(2) Normalized standard deviations of the RTL-behavioural model difference in both lag and frequency domains is an estimated 30 to 50% higher than those of test case 12. Salient differences between the two cases are the **SCHID** frame size and application of input data synchronization errors.

(3) see the section on on line error reporting, 2.10.6, for additional notes.

```
***--------------------------------------------------------------------------***
tc11_postsim.m : V2.00 run time: Sat Sep 03 15:01:01 2005
Using postsim_lib: 1.40: 05-08-18

48 frames read for test case 11.
test case version: v2,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

WARNINGS encountered during register status and data capture checks;
                        details in log file.

ADVISORIES encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc11
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
```

Text 2.5.20 console log for scripts tc11_postsim.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

ADVISORY from check_register_capture: Synch. error status bit
           (X|YSyncer) settings in an output frame are not identical
            between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 1.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 1.

WARNING from check_register_capture: Captured X-Station baseband
      ID (BBID-X) differs between cells of chain 1.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 1.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      of chain 1 differs between dumps.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      of chain 1 differs between dumps.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) of chain 1 differs between dumps.

WARNING from check_register_capture: Captured Y-Station subband
      ID (SBID-Y) of chain 1 differs between dumps.

WARNING from check_register_capture: X-Station baseband
      ID (BBID-X) of chain 1 differs between dumps.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) of chain 1 differs between dumps.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      differs between cells of chain 5.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 5.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) differs between cells of chain 5.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 5.
```

Text 2.5.21 content of tc11_register_log.txt, part 1.

```
WARNING from check_register_capture: Captured X-Station baseband
      ID (BBID-X) differs between cells of chain 5.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 5.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      of chain 5 differs between dumps.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      of chain 5 differs between dumps.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) of chain 5 differs between dumps.

WARNING from check_register_capture: Captured Y-Station subband
      ID (SBID-Y) of chain 5 differs between dumps.

WARNING from check_register_capture: X-Station baseband
      ID (BBID-X) of chain 5 differs between dumps.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) of chain 5 differs between dumps.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

WARNING from check_register_capture: status X_DVALID
                             differs from template.

WARNING from check_register_capture: status Y_DVALID
                             differs from template.

WARNING from check_register_capture: status X_SE_CLK
                             differs from template.

WARNING from check_register_capture: status Y_SE_CLK
                             differs from template.
```

Text 2.5.22 content of tc11_register_log.txt, part 2

chain	***dump***	***XSyncerr***	***SID_X***	***SBID_X***	***BBID_X***
1	3	1 1	128 200	12 0	0 3
5	1	0 1	200 128	4 12	7 2

Table 2.5.33 test case 11: captured X input data obtained from the output frame for chains in which captured data differs between cells as reported in tc11_register_log.txt. Table contents generated from script tc11_inspect_status_captured.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	7.43e-04	2.72e-06	0.1891	0.2188	0.0292	0.0365	1.46	1.83	2	
1	2	1.47e-03	6.20e-03	0.2217	0.2132	0.0418	0.0348	2.09	1.74	2	*
1	3	-2.30e-03	-4.64e-03	0.1732	0.2620	0.0333	0.0398	1.66	1.99	1	
2	1	-1.81e-03	-6.24e-04	0.1414	0.1614	0.0275	0.0275	1.38	1.38	2	*
2	2	-7.83e-04	-1.24e-04	0.1623	0.1591	0.0279	0.0282	1.39	1.41	1	
2	3	1.39e-03	7.20e-04	0.1476	0.1756	0.0290	0.0329	1.45	1.65	0	
3	1	-1.11e-03	-7.63e-04	0.1441	0.1796	0.0286	0.0278	1.43	1.39	1	
3	2	4.27e-03	2.19e-03	0.1698	0.1974	0.0298	0.0301	1.49	1.50	3	
3	3	-1.83e-03	-2.26e-03	0.2088	0.2007	0.0332	0.0305	1.66	1.52	2	
4	1	-1.20e-03	6.34e-04	0.1547	0.1887	0.0315	0.0360	1.58	1.80	0	
4	2	-5.62e-04	-3.05e-03	0.1741	0.1670	0.0270	0.0328	1.35	1.64	1	
4	3	8.33e-04	1.68e-03	0.1877	0.1788	0.0316	0.0306	1.58	1.53	1	
5	1	4.81e-04	1.69e-03	0.1510	0.1304	0.0277	0.0237	1.38	1.19	0	
5	2	-2.88e-03	-1.94e-03	0.1887	0.1808	0.0301	0.0321	1.50	1.61	2	
5	3	2.62e-03	-2.19e-03	0.1613	0.1651	0.0293	0.0308	1.46	1.54	0	
6	1	9.47e-05	2.17e-04	0.1485	0.1475	0.0277	0.0302	1.39	1.51	0	
6	2	1.02e-03	2.85e-04	0.1628	0.1906	0.0310	0.0316	1.55	1.58	1	
6	3	-4.25e-04	2.58e-04	0.1479	0.1330	0.0295	0.0250	1.47	1.25	0	
7	1	1.96e-04	6.33e-04	0.1072	0.1307	0.0224	0.0251	1.12	1.25	0	
7	2	-4.60e-04	-5.59e-04	0.1243	0.1754	0.0255	0.0295	1.27	1.48	1	
7	3	2.76e-04	-1.29e-03	0.1514	0.1685	0.0280	0.0304	1.40	1.52	1	
8	1	-1.74e-03	1.76e-03	0.1658	0.1940	0.0299	0.0291	1.50	1.46	2	
8	2	4.22e-05	1.52e-04	0.1395	0.1685	0.0292	0.0313	1.46	1.57	0	
8	3	3.61e-04	-1.62e-04	0.1622	0.1488	0.0268	0.0263	1.34	1.32	1	

Table 2.5.34 test case 11; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[\,f-f\,]}$	$\overline{\Im[\,f-f\,]}$	*range* $\Re[\,f-f\,]$	*range* $\Im[\,f-f\,]$	$\sigma_{\Re[\,f-f\,]}$	$\sigma_{\Im[\,f-f\,]}$	$\frac{\sigma_{\Re[\,f-f\,]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[\,f-f\,]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-6.00e-03	6.05e-03	0.4759	0.4312	0.0967	0.0889	1.16	0.97	128
1	2	1.55e-03	-6.71e-05	0.4751	0.4411	0.0851	0.0825	1.05	1.07	null
2	1	-3.06e-03	1.10e-03	0.4871	0.4558	0.0835	0.0859	1.12	0.99	null
2	2	7.09e-03	1.77e-03	0.4577	0.4770	0.0813	0.0915	1.05	1.06	null
3	1	1.36e-03	6.88e-03	0.4619	0.4024	0.0815	0.0760	1.05	0.98	null
3	2	-8.42e-03	-1.40e-03	0.4721	0.4855	0.0834	0.0876	0.96	0.96	null
4	1	-4.12e-03	-2.77e-04	0.4117	0.4553	0.0744	0.0758	0.99	0.96	null
4	2	4.45e-03	2.67e-03	0.5309	0.3747	0.0874	0.0785	1.05	0.91	null
5	1	-2.16e-04	-2.30e-03	0.4539	0.4097	0.0784	0.0742	0.91	1.05	null
5	2	4.38e-04	-3.59e-03	0.4829	0.3539	0.0777	0.0713	0.88	0.93	null
6	1	-2.02e-03	-5.55e-03	0.3466	0.4503	0.0668	0.0787	1.05	1.05	null
6	2	2.50e-03	2.86e-03	0.4108	0.4442	0.0790	0.0772	1.06	1.10	null
7	1	-6.01e-04	-3.36e-03	0.4466	0.3888	0.0806	0.0815	1.01	1.19	null
7	2	-1.68e-03	3.41e-03	0.5202	0.5320	0.0970	0.0879	1.10	1.02	null
8	1	1.68e-03	1.99e-03	0.3426	0.4226	0.0708	0.0688	1.00	0.99	null
8	2	-6.87e-04	-8.28e-05	0.4198	0.4727	0.0772	0.0805	1.05	1.08	null

Table 2.5.35 test case 11; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	7.11e-04	1.12e-03	0.0118	0.0256	2.18e-03	2.77e-03	3.6	4.6	0	
1	2	-9.44e-04	4.51e-04	0.0359	0.0222	3.45e-03	2.38e-03	5.8	4.0	0	
1	3	-1.40e-04	-1.43e-03	0.0145	0.0309	2.35e-03	2.87e-03	3.9	4.8	0	
2	1	-1.13e-04	-1.13e-04	0.0166	0.0102	2.25e-03	1.76e-03	3.8	2.9	0	
2	2	1.79e-04	2.03e-04	0.0159	0.0158	2.42e-03	2.07e-03	4.0	3.4	0	
2	3	4.34e-04	-3.80e-06	0.0269	0.0133	2.68e-03	2.02e-03	4.5	3.4	0	
3	1	5.39e-05	-8.74e-05	0.0192	0.0146	2.31e-03	2.14e-03	3.9	3.6	0	
3	2	-1.89e-04	4.65e-04	0.0267	0.0161	2.61e-03	2.20e-03	4.3	3.7	0	
3	3	3.63e-04	1.92e-04	0.0295	0.0134	2.73e-03	2.16e-03	4.6	3.6	0	
4	1	-1.18e-04	2.00e-04	0.0324	0.0216	3.02e-03	2.38e-03	5.0	4.0	0	
4	2	2.07e-04	2.14e-04	0.0257	0.0148	2.58e-03	1.81e-03	4.3	3.0	0	
4	3	-1.36e-04	-4.26e-04	0.0202	0.0237	2.29e-03	2.53e-03	3.8	4.2	0	
5	1	-3.15e-04	-1.47e-04	0.0167	0.0136	2.21e-03	1.78e-03	3.7	3.0	0	
5	2	7.61e-05	-3.17e-05	0.0211	0.0146	2.35e-03	1.90e-03	3.9	3.2	0	
5	3	1.17e-04	-2.44e-04	0.0225	0.0132	2.51e-03	2.05e-03	4.2	3.4	0	
6	1	5.35e-04	-4.11e-04	0.0195	0.0181	2.54e-03	2.17e-03	4.2	3.6	0	
6	2	-1.08e-04	3.81e-04	0.0302	0.0119	2.94e-03	1.70e-03	4.9	2.8	0	
6	3	1.08e-04	-3.60e-04	0.0187	0.0156	2.25e-03	1.90e-03	3.7	3.2	0	
7	1	-2.27e-04	3.42e-05	0.0120	0.0159	1.86e-03	1.84e-03	3.1	3.1	0	
7	2	1.99e-04	7.11e-05	0.0227	0.0122	2.30e-03	1.79e-03	3.8	3.0	0	
7	3	-4.67e-05	2.99e-04	0.0206	0.0126	2.54e-03	1.94e-03	4.2	3.2	0	
8	1	3.51e-04	-2.56e-04	0.0235	0.0147	2.49e-03	2.00e-03	4.2	3.3	0	
8	2	-1.13e-04	9.28e-06	0.0208	0.0156	2.55e-03	2.14e-03	4.2	3.6	0	
8	3	1.62e-04	-1.25e-04	0.0182	0.0130	2.18e-03	1.90e-03	3.6	3.2	0	

Table 2.5.36 test case 11; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: ! OVR*	*bias: OVR*	*applied dump - bias: OVR*
0	12000	11724	276	0	n-a
1	12000	11724	276	0	n-a
2	12000	11724	276	0	n-a
3	12000	11724	276	0	n-a
4	12000	11724	276	0	n-a
5	12000	11724	276	0	n-a
6	12000	11724	276	0	n-a
7	12000	11724	276	0	n-a

Table 2.5.37 test case 11; applied dump and bias.

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

11111111  XSTATUS: X-SDATA1: 1111, X-SDATA0: 1111
11111111  YSTATUS: Y-SDATA1: 1111, Y-SDATA0: 1111
11111111  XSTATUS: X-SDATA3: 1111, X-SDATA2: 1111
11111111  YSTATUS: Y-SDATA3: 1111, Y-SDATA2: 1111
11111111  XSTATUS: X-SDATA5: 1111, X-SDATA4: 1111
11111111  YSTATUS: Y-SDATA5: 1111, Y-SDATA4: 1111
11111111  XSTATUS: X-SDATA7: 1111, X-SDATA6: 1111
11111111  YSTATUS: Y-SDATA7: 1111, Y-SDATA6: 1111
11111111  XSTATUS: X-PHASE1: 1111, X-PHASE0: 1111
11111111  YSTATUS: Y-PHASE1: 1111, Y-PHASE0: 1111
11111111  XSTATUS: X-PHASE3: 1111, X-PHASE2: 1111
11111111  YSTATUS: Y-PHASE3: 1111, Y-PHASE2: 1111
11111111  XSTATUS: X-PHASE5: 1111, X-PHASE4: 1111
11111111  YSTATUS: Y-PHASE5: 1111, Y-PHASE4: 1111
11111111  XSTATUS: X-PHASE7: 1111, X-PHASE6: 1111
11111111  YSTATUS: Y-PHASE7: 1111, Y-PHASE6: 1111
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
10111111  XSTATUS: X-SE_CLK: 10111111
10111111  YSTATUS: Y-SE_CLK: 10111111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000100  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=1
00000100  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=1

11111111  DESSR_0_7: 11111111
11111111  DESSR_8_15: 11111111
```

Text 2.5.23 content of file mcb_reg11.txt

2.5.10 test case 12

Synopsis: No extraordinary table entries are noted and and no advisories or warnings regarding captured data or register content were logged.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc12_postsim.m : V2.00 run time: Sat Sep 03 17:59:21 2005
Using postsim_lib: 1.40: 05-08-18

48 frames read for test case 12.
test case version: v2,05may14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc12
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
```

Text 2.5.24 console log for scripts tc12_postsim

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	1.06e-03	-6.78e-04	0.1792	0.1216	0.0259	0.0215	1.29	1.08	2	
1	2	-4.63e-04	8.93e-04	0.1443	0.1398	0.0257	0.0245	1.28	1.23	3	
1	3	8.80e-05	1.17e-03	0.1311	0.1484	0.0224	0.0265	1.12	1.33	1	
2	1	-2.12e-03	-9.67e-05	0.1300	0.1178	0.0252	0.0239	1.26	1.19	0	
2	2	-1.18e-03	1.04e-03	0.1727	0.1472	0.0263	0.0238	1.31	1.19	3	
2	3	1.26e-03	-5.45e-04	0.1526	0.1670	0.0274	0.0302	1.37	1.51	1	
3	1	1.40e-04	-7.99e-05	0.1110	0.1205	0.0229	0.0237	1.15	1.18	0	
3	2	7.99e-04	1.86e-03	0.1278	0.1439	0.0236	0.0226	1.18	1.13	3	
3	3	-9.84e-04	-1.17e-03	0.1425	0.1383	0.0242	0.0265	1.21	1.33	0	
4	1	-1.39e-03	1.30e-04	0.1217	0.1413	0.0241	0.0279	1.20	1.39	0	
4	2	3.93e-04	-1.32e-03	0.1209	0.1356	0.0208	0.0257	1.04	1.28	1	
4	3	4.26e-04	3.53e-04	0.1402	0.1318	0.0239	0.0239	1.19	1.19	2	
5	1	1.30e-03	1.54e-03	0.1422	0.1084	0.0252	0.0208	1.26	1.04	1	
5	2	-2.45e-03	-1.33e-03	0.1364	0.1285	0.0227	0.0216	1.13	1.08	2	
5	3	1.82e-03	-8.02e-04	0.1370	0.1670	0.0257	0.0260	1.29	1.30	1	
6	1	6.47e-04	1.22e-04	0.1339	0.1099	0.0230	0.0227	1.15	1.13	1	
6	2	2.02e-04	6.24e-04	0.1279	0.1258	0.0243	0.0243	1.22	1.21	0	
6	3	-5.85e-04	5.53e-04	0.0977	0.1173	0.0204	0.0212	1.02	1.06	1	
7	1	1.37e-03	-3.70e-04	0.1060	0.1245	0.0191	0.0203	0.95	1.01	1	
7	2	-1.72e-03	-1.38e-03	0.1235	0.1469	0.0225	0.0234	1.13	1.17	2	
7	3	7.42e-04	-8.14e-04	0.1115	0.1177	0.0215	0.0248	1.07	1.24	0	
8	1	-9.57e-04	1.89e-03	0.1070	0.1288	0.0200	0.0230	1.00	1.15	0	
8	2	6.49e-04	4.17e-04	0.1201	0.1192	0.0212	0.0218	1.06	1.09	1	
8	3	7.38e-04	1.36e-03	0.1277	0.1388	0.0211	0.0214	1.06	1.07	3	

Table 2.5.38 test case 12; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-8.25e-03	1.42e-03	0.4654	0.5042	0.0936	0.0975	1.13	1.07	128
1	2	5.87e-03	1.11e-02	0.4326	0.4388	0.0813	0.0771	1.00	1.00	null
2	1	-3.15e-03	1.74e-03	0.5308	0.4945	0.0807	0.0832	1.08	0.96	null
2	2	7.36e-03	-6.60e-04	0.4963	0.5375	0.0888	0.0924	1.14	1.07	null
3	1	-3.36e-03	5.87e-03	0.4846	0.4732	0.0778	0.0793	1.00	1.02	null
3	2	-4.11e-03	2.39e-05	0.4397	0.4798	0.0864	0.0896	0.99	0.99	null
4	1	-2.97e-03	1.96e-03	0.3926	0.4793	0.0731	0.0771	0.97	0.97	null
4	2	3.09e-03	-3.83e-04	0.4608	0.4249	0.0861	0.0847	1.03	0.98	null
5	1	-6.04e-04	-1.54e-03	0.4142	0.4009	0.0776	0.0715	0.90	1.01	null
5	2	-7.97e-04	-2.82e-03	0.4797	0.3664	0.0801	0.0715	0.90	0.94	null
6	1	-3.39e-03	-5.12e-03	0.3905	0.4265	0.0674	0.0776	1.06	1.03	null
6	2	3.16e-03	2.82e-03	0.3823	0.5146	0.0764	0.0786	1.02	1.12	null
7	1	-3.04e-03	-3.18e-03	0.4080	0.4107	0.0766	0.0749	0.96	1.09	null
7	2	4.72e-05	4.71e-03	0.4401	0.5290	0.0900	0.0869	1.02	1.01	null
8	1	1.49e-03	2.12e-03	0.3408	0.3731	0.0709	0.0684	1.00	0.98	null
8	2	-9.17e-04	1.18e-03	0.4252	0.4204	0.0754	0.0754	1.03	1.01	null

Table 2.5.39 test case 12; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	8.26e-04	-4.68e-07	0.0156	0.0076	2.14e-03	1.53e-03	3.6	2.6	0	
1	2	-1.26e-04	5.77e-04	0.0211	0.0121	2.19e-03	1.61e-03	3.6	2.7	0	
1	3	-2.54e-04	-4.62e-04	0.0143	0.0126	1.80e-03	1.75e-03	3.0	2.9	0	
2	1	-1.93e-04	-3.18e-04	0.0146	0.0160	1.93e-03	1.70e-03	3.2	2.8	0	
2	2	-7.25e-05	-8.59e-05	0.0148	0.0101	2.07e-03	1.70e-03	3.5	2.8	0	
2	3	4.30e-04	3.60e-04	0.0236	0.0142	2.51e-03	1.88e-03	4.2	3.1	0	
3	1	1.20e-04	-2.08e-04	0.0154	0.0120	1.92e-03	1.71e-03	3.2	2.8	0	
3	2	-9.43e-05	2.60e-04	0.0150	0.0122	1.95e-03	1.74e-03	3.2	2.9	0	
3	3	1.31e-04	-9.19e-05	0.0217	0.0136	2.24e-03	1.87e-03	3.7	3.1	0	
4	1	3.12e-05	6.34e-05	0.0228	0.0137	2.28e-03	1.81e-03	3.8	3.0	0	
4	2	3.91e-05	5.33e-05	0.0179	0.0125	1.97e-03	1.56e-03	3.3	2.6	0	
4	3	-9.37e-05	-3.08e-04	0.0136	0.0142	1.76e-03	1.87e-03	2.9	3.1	0	
5	1	-1.19e-04	-2.79e-05	0.0217	0.0111	2.15e-03	1.35e-03	3.6	2.3	0	
5	2	3.61e-05	1.70e-04	0.0121	0.0144	1.70e-03	1.55e-03	2.8	2.6	0	
5	3	-1.46e-04	-1.81e-04	0.0177	0.0117	2.00e-03	1.72e-03	3.3	2.9	0	
6	1	5.17e-04	-2.28e-04	0.0166	0.0103	2.11e-03	1.53e-03	3.5	2.5	0	
6	2	5.47e-05	2.89e-04	0.0169	0.0118	2.13e-03	1.50e-03	3.5	2.5	0	
6	3	1.19e-04	-2.07e-04	0.0102	0.0144	1.55e-03	1.62e-03	2.6	2.7	0	
7	1	-1.21e-04	-1.48e-04	0.0105	0.0140	1.61e-03	1.55e-03	2.7	2.6	0	
7	2	-6.99e-05	1.19e-04	0.0191	0.0096	2.01e-03	1.39e-03	3.3	2.3	0	
7	3	1.18e-04	1.64e-04	0.0176	0.0111	1.91e-03	1.56e-03	3.2	2.6	0	
8	1	5.91e-05	1.50e-04	0.0163	0.0102	1.81e-03	1.58e-03	3.0	2.6	0	
8	2	4.14e-05	-1.08e-04	0.0148	0.0100	1.73e-03	1.41e-03	2.9	2.4	0	
8	3	2.57e-05	-1.92e-04	0.0138	0.0098	1.86e-03	1.41e-03	3.1	2.3	0	

Table 2.5.40 test case 12; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: ! OVR*	*bias: OVR*	*applied dump - bias: OVR*
0	12000	11724	276	0	n-a
1	12000	11724	276	0	n-a
2	12000	11724	276	0	n-a
3	12000	11724	276	0	n-a
4	12000	11724	276	0	n-a
5	12000	11724	276	0	n-a
6	12000	11724	276	0	n-a
7	12000	11724	276	0	n-a

Table 2.5.41 test case 12; applied dump interval and bias.

```
#-------------------------------------------------------------------------------
# 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:
#
# -------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
01010101  XSTATUS: X-DUMP_EN: 01010101
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.25 content of file mcb_reg12.txt

2.5.11 test case 14

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged. Plots are included to exhibit operation of a four polarization configuration and appear as expected. Notably, the frequency domain coefficient plots exhibit the expected absence of continuum in cross polarized products and the expected number of dominant spectral peaks.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc14_postsim.m : V2.00 run time: Sun Sep 04 12:16:08 2005
Using postsim_lib: 1.40: 05-08-18

48 frames read for test case 14.
test case version: v3,05apr22
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
0(0:3)0  0(4:7)1  1(8:11)0  1(12:15)1


Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc14
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
```

Text 2.5.26 console log for scripts tc14_postsim.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	2.11e-03	-4.21e-04	0.1966	0.1939	0.0248	0.0255	1.13	1.16	3	*
1	2	-2.13e-03	4.93e-04	0.1537	0.1103	0.0220	0.0220	1.01	1.00	3	
1	3	5.91e-04	-3.55e-06	0.1606	0.1834	0.0253	0.0276	1.15	1.26	2	
2	1	9.78e-04	2.96e-05	0.1623	0.1238	0.0219	0.0200	1.00	0.91	3	
2	2	-7.22e-04	-1.53e-03	0.1754	0.1263	0.0270	0.0228	1.23	1.04	1	*
2	3	8.10e-05	-5.32e-04	0.1307	0.1185	0.0217	0.0187	0.99	0.86	2	
3	1	3.23e-04	-2.79e-04	0.1311	0.1182	0.0203	0.0197	0.93	0.90	3	
3	2	2.25e-04	7.78e-04	0.1262	0.1634	0.0220	0.0246	1.01	1.12	3	*
3	3	-2.96e-04	1.67e-04	0.1513	0.1521	0.0213	0.0237	0.97	1.08	2	
4	1	-1.01e-04	-5.14e-04	0.1061	0.1183	0.0176	0.0189	0.80	0.86	3	
4	2	-1.00e-03	1.28e-05	0.1476	0.1364	0.0237	0.0247	1.08	1.13	1	*
4	3	1.48e-03	-2.90e-04	0.1243	0.1093	0.0216	0.0201	0.98	0.92	0	

Table 2.5.42 test case 14; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-2.16e-03	4.09e-03	0.5771	0.4837	0.0914	0.0880	0.99	0.95	255 256
1	2	2.09e-03	3.73e-03	0.5297	0.5595	0.0876	0.0859	1.05	0.98	256
2	1	-8.70e-04	-1.15e-03	0.5933	0.5021	0.0972	0.0942	1.06	1.02	null
2	2	1.09e-03	1.85e-03	0.5558	0.5253	0.0971	0.0937	1.07	1.02	null
3	1	1.17e-03	2.97e-03	0.4455	0.5275	0.0844	0.0878	1.03	1.05	null
3	2	3.07e-05	-2.61e-04	0.4771	0.6460	0.0864	0.0940	1.00	1.00	null
4	1	-4.37e-03	2.61e-03	0.5456	0.5056	0.0852	0.0853	0.95	0.95	null
4	2	1.87e-03	-2.09e-03	0.6518	0.5360	0.0927	0.0927	0.95	0.94	null

Table 2.5.43 test case 14; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	1.03e-04	4.61e-04	0.0113	0.0246	1.20e-03	1.70e-03	1.8	2.6	2	*
1	2	-3.26e-04	7.89e-05	0.0167	0.0063	1.31e-03	1.02e-03	2.0	1.5	1	
1	3	-1.77e-04	-3.97e-04	0.0157	0.0243	1.18e-03	1.66e-03	1.8	2.5	2	
2	1	1.71e-04	-7.84e-05	0.0091	0.0132	1.00e-03	1.17e-03	1.5	1.8	0	
2	2	-1.42e-04	8.88e-05	0.0134	0.0192	1.41e-03	1.38e-03	2.1	2.1	1	*
2	3	5.94e-05	1.84e-05	0.0070	0.0087	1.11e-03	1.10e-03	1.7	1.7	0	
3	1	1.01e-04	-1.16e-04	0.0149	0.0108	1.15e-03	1.12e-03	1.7	1.7	0	
3	2	-4.27e-05	6.41e-05	0.0156	0.0177	1.23e-03	1.39e-03	1.9	2.1	2	*
3	3	6.73e-06	6.23e-05	0.0108	0.0121	1.16e-03	1.15e-03	1.8	1.7	0	
4	1	-2.74e-05	-1.96e-05	0.0086	0.0067	1.02e-03	9.38e-04	1.6	1.4	0	
4	2	3.55e-04	2.66e-05	0.0190	0.0088	1.50e-03	1.08e-03	2.3	1.6	0	*
4	3	-1.89e-04	-5.42e-06	0.0124	0.0083	1.22e-03	9.97e-04	1.9	1.5	0	

Table 2.5.44 test case 14; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: ! OVR*	*bias: OVR*	*applied dump - bias: OVR*
0	20000	19724	276	0	n-a
1	20000	19724	276	0	n-a

Table 2.5.45 test case 14; applied dump and bias.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111100  XSTATUS: X-SE_CLK: 11111100
11111100  YSTATUS: Y-SE_CLK: 11111100
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.27 content of file mcb_reg14.txt

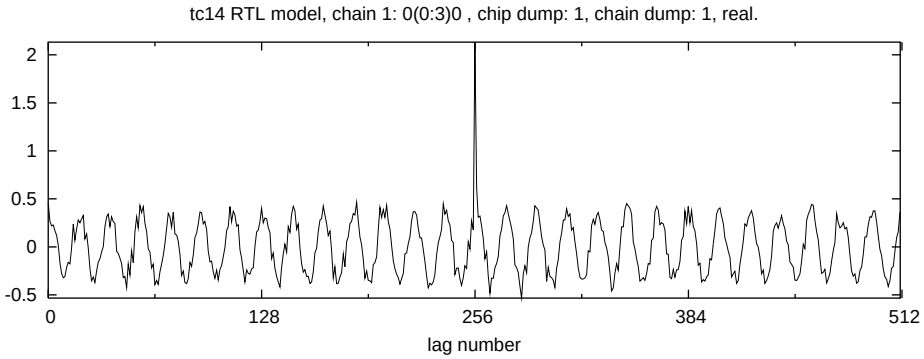

Illustration 2.5.217

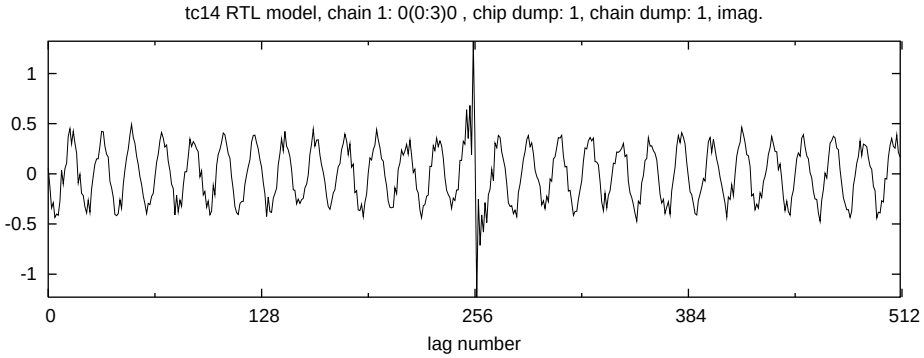

Illustration 2.5.218

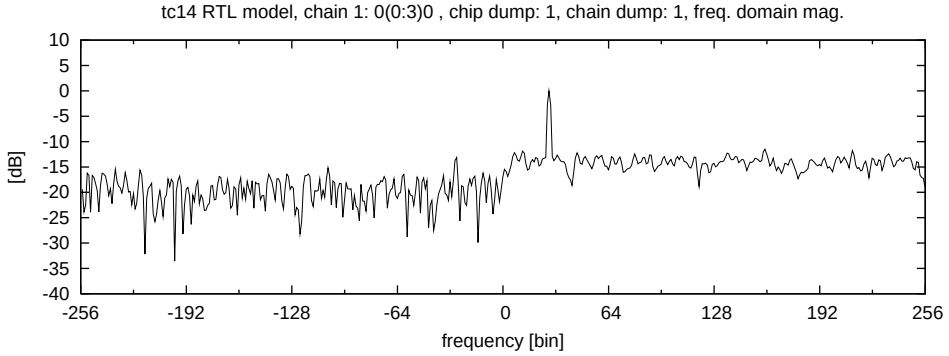

Illustration 2.5.219

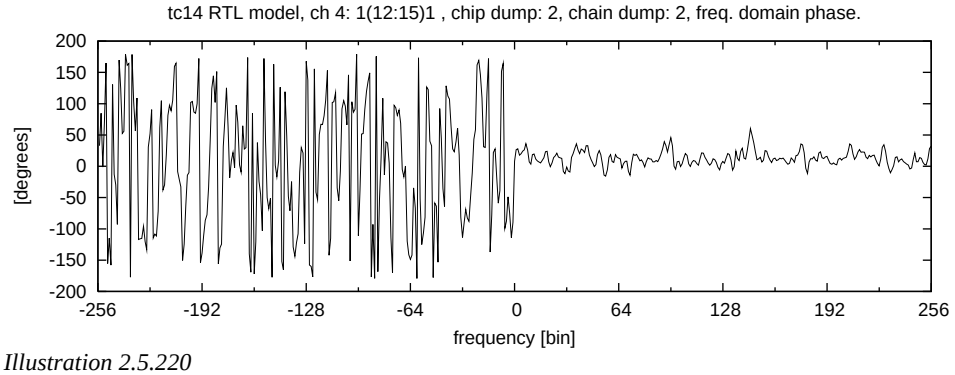

Illustration 2.5.220

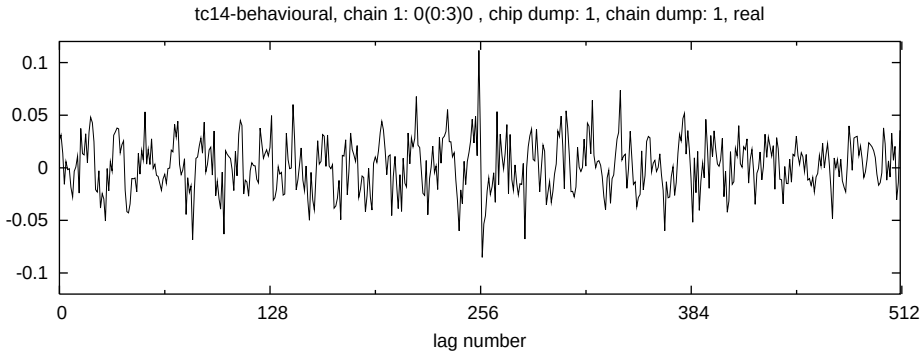

Illustration 2.5.221

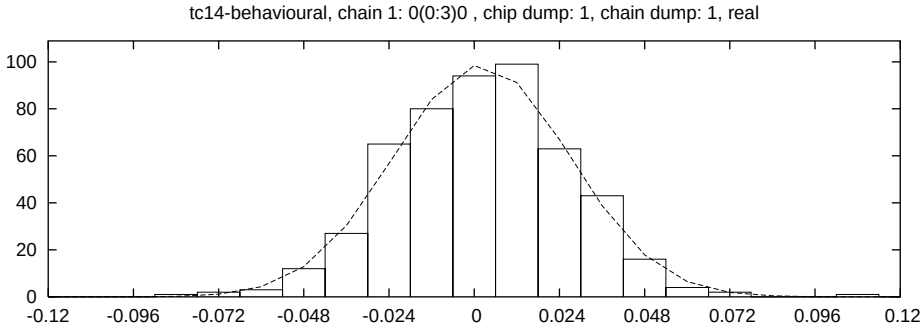

Illustration 2.5.222

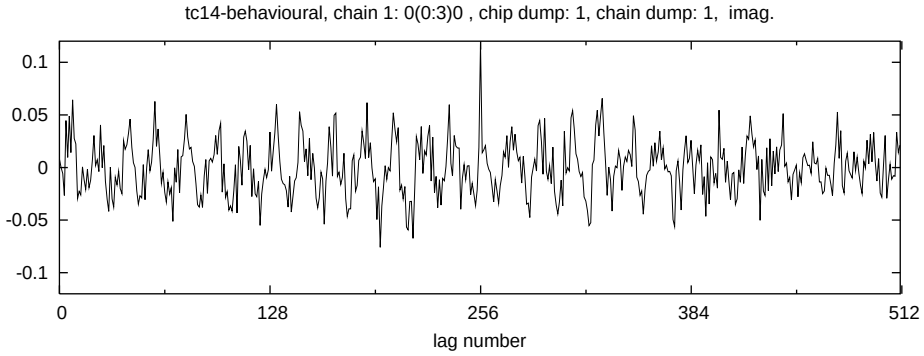

Illustration 2.5.223

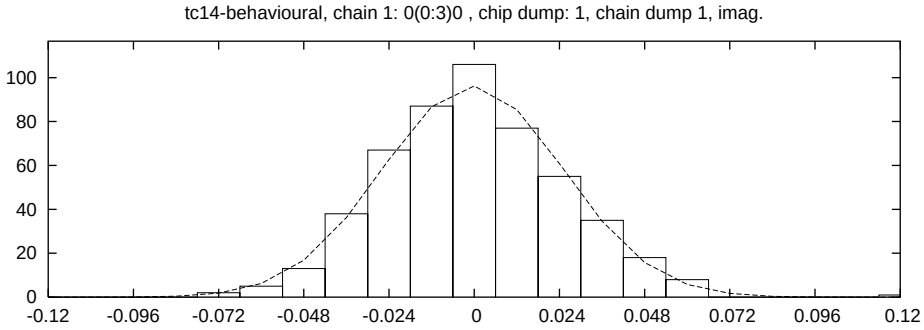

Illustration 2.5.224

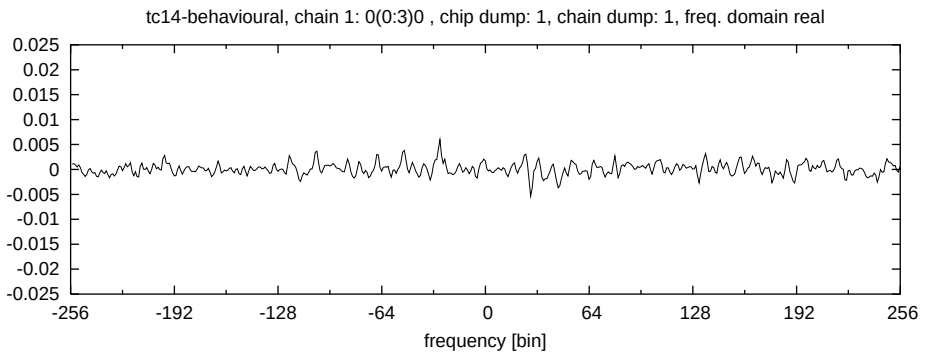

Illustration 2.5.225

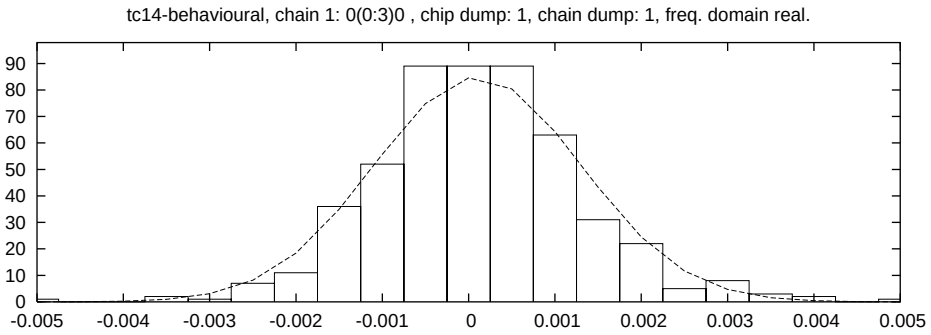

Illustration 2.5.226

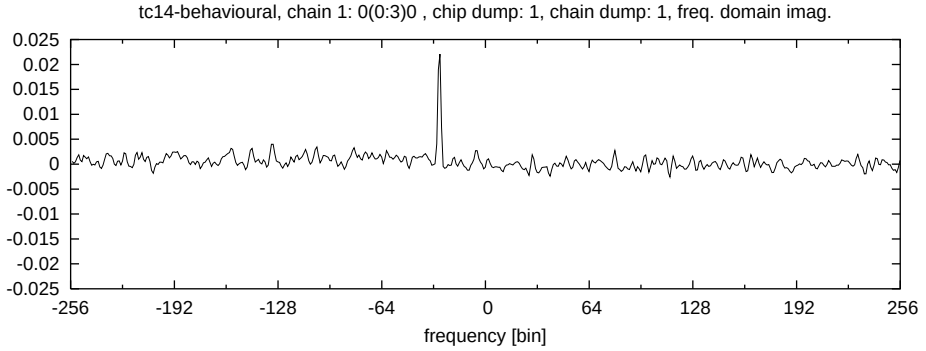

Illustration 2.5.227

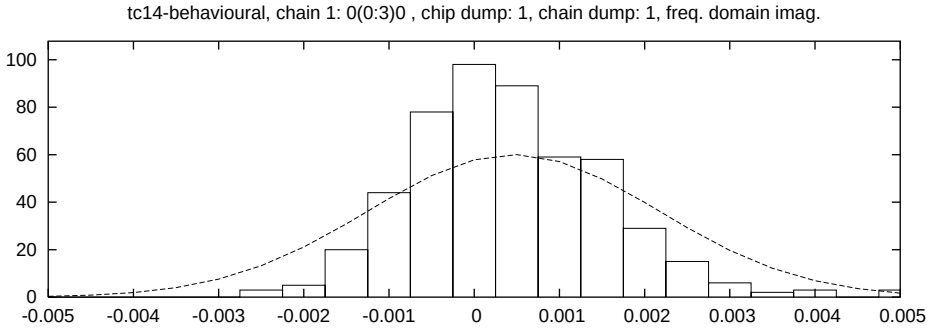

Illustration 2.5.228

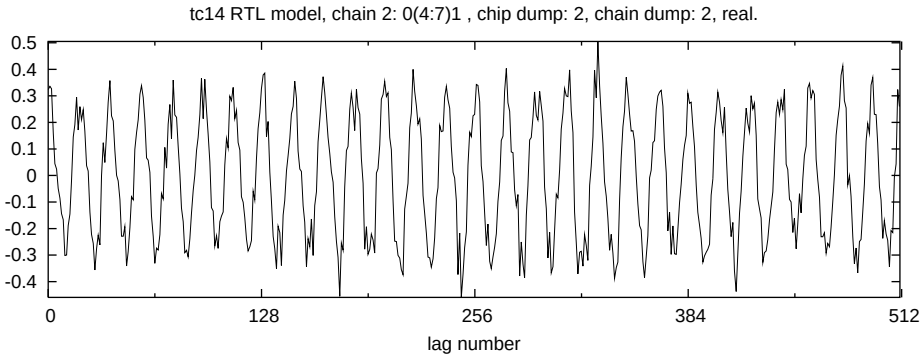

Illustration 2.5.229

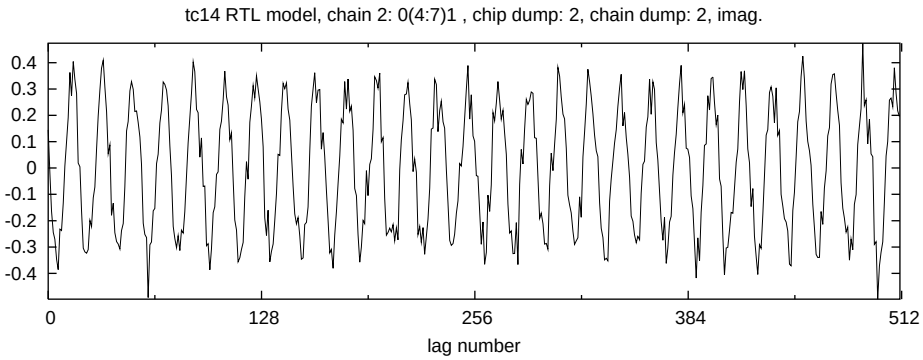

Illustration 2.5.230

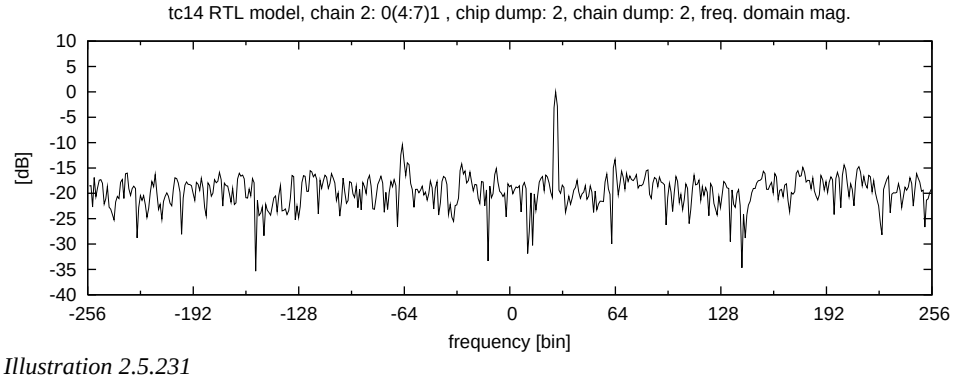

Illustration 2.5.231

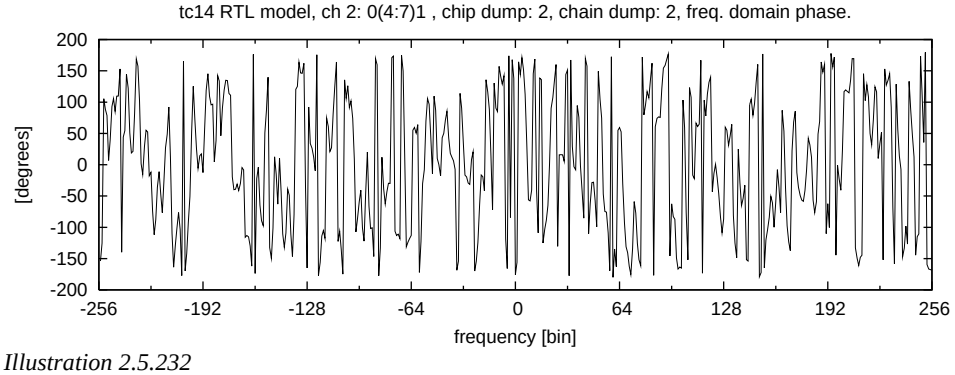

Illustration 2.5.232

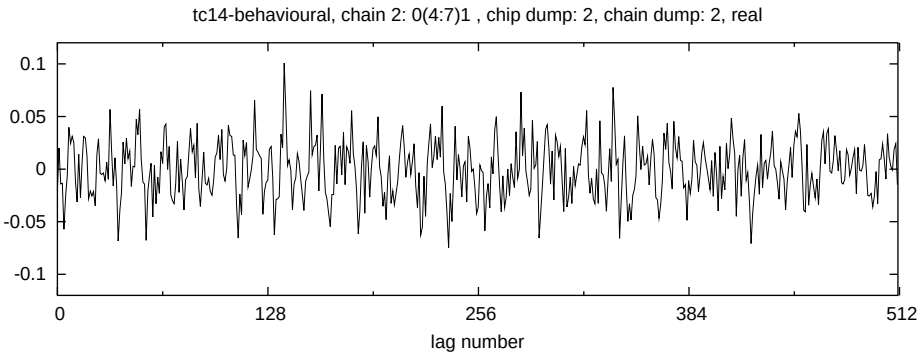

Illustration 2.5.233

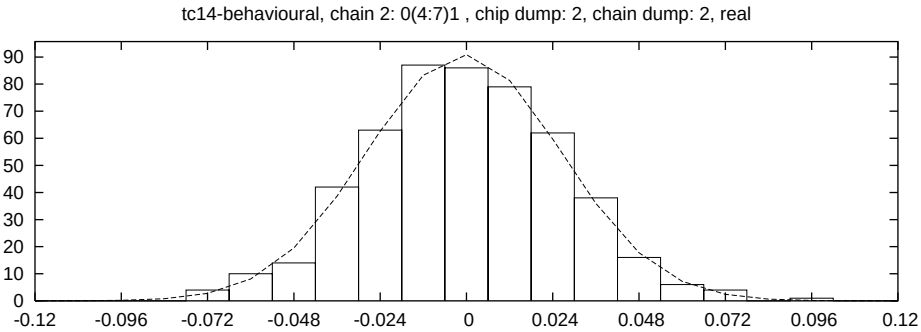

Illustration 2.5.234

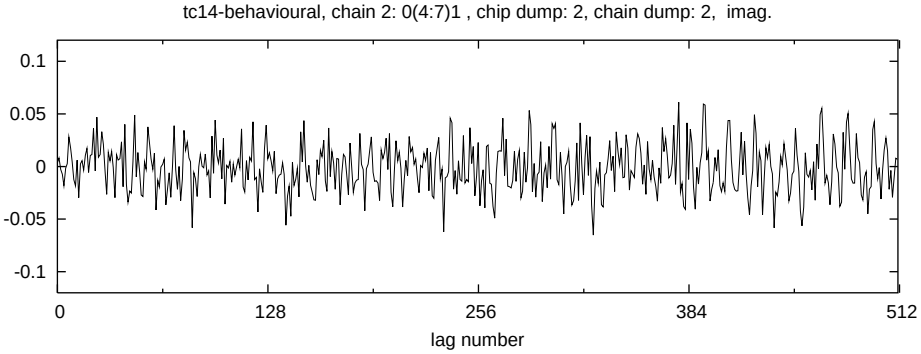

Illustration 2.5.235

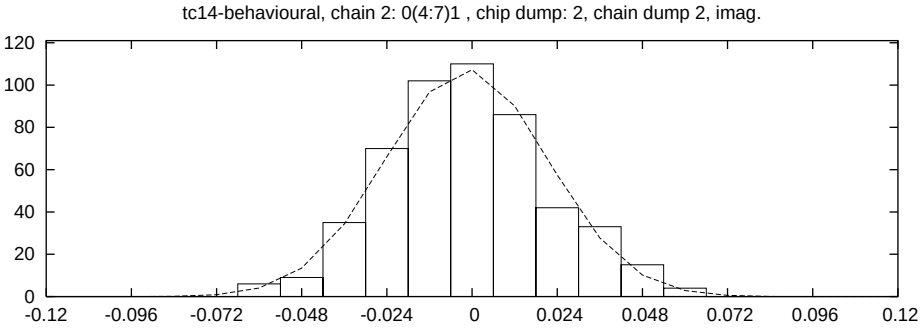

Illustration 2.5.236

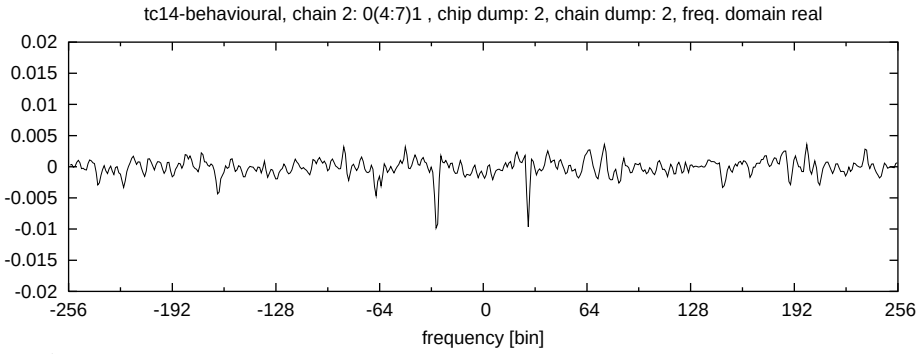

Illustration 2.5.237

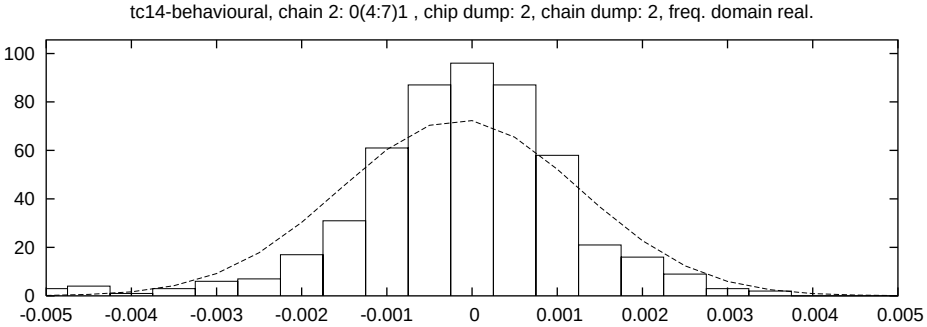

Illustration 2.5.238

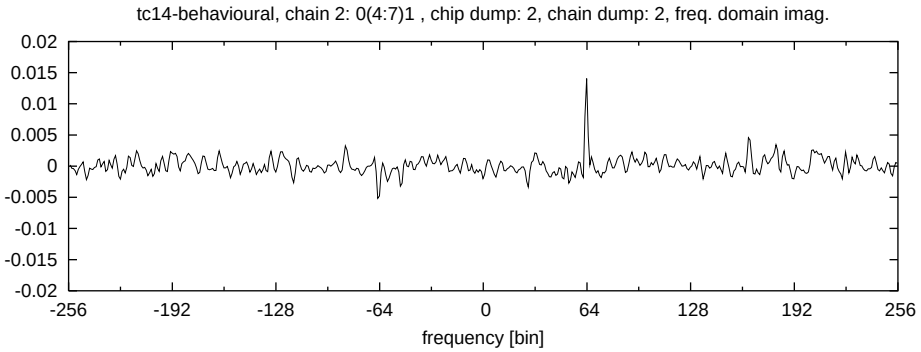

Illustration 2.5.239

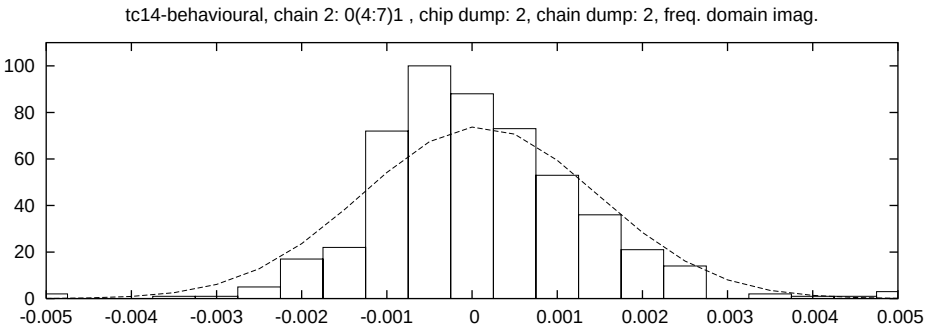

Illustration 2.5.240

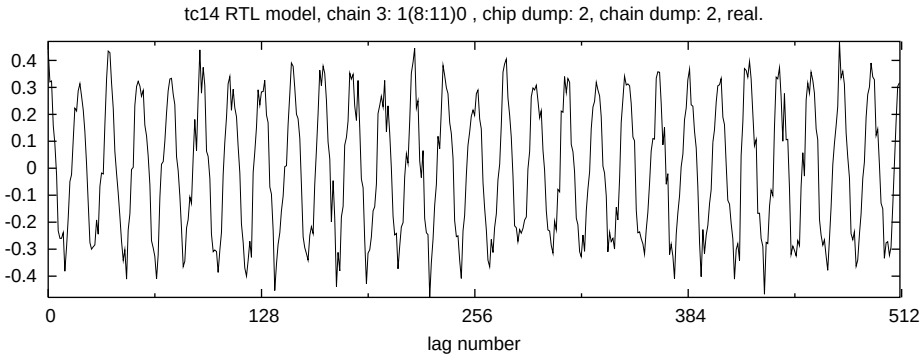

Illustration 2.5.241

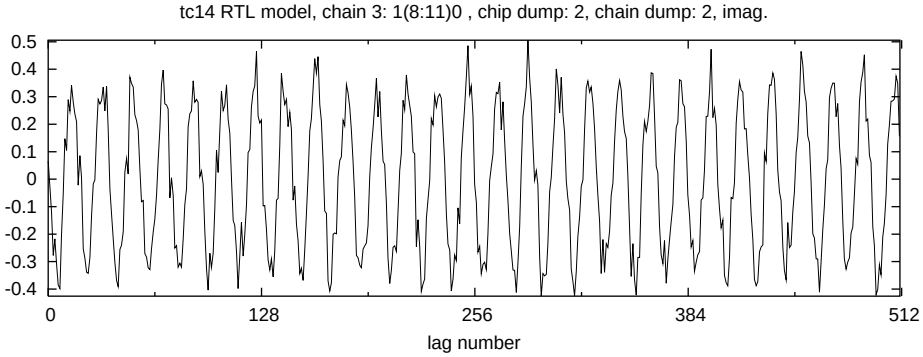

Illustration 2.5.242

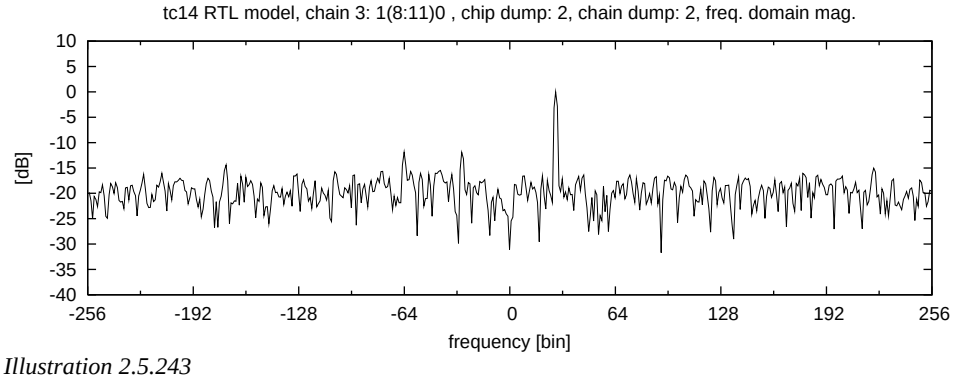

Illustration 2.5.243

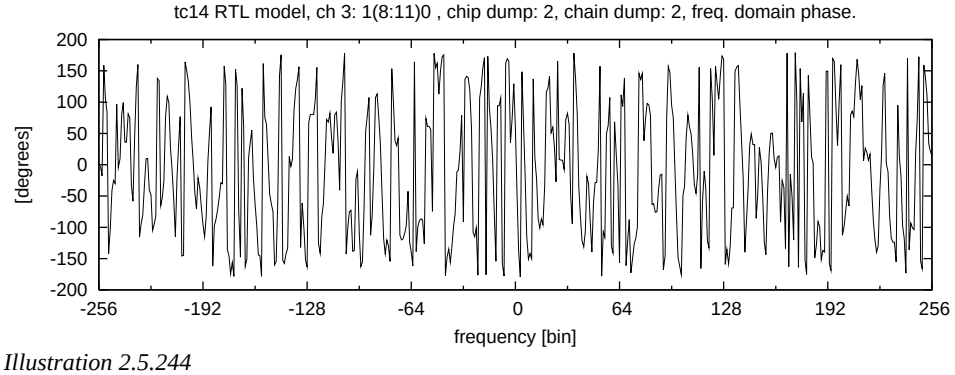

Illustration 2.5.244

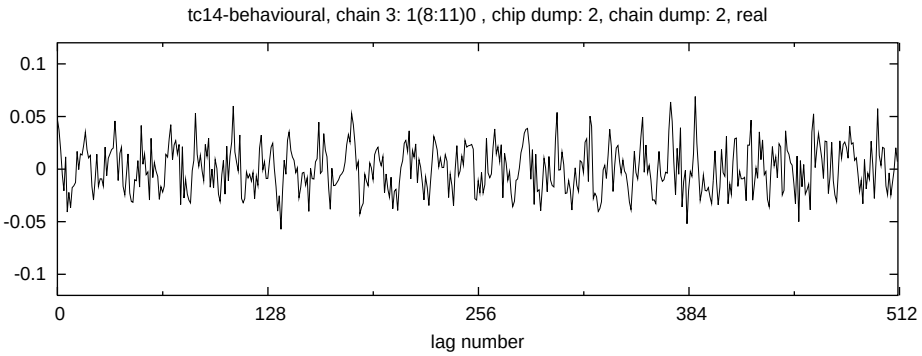

Illustration 2.5.245

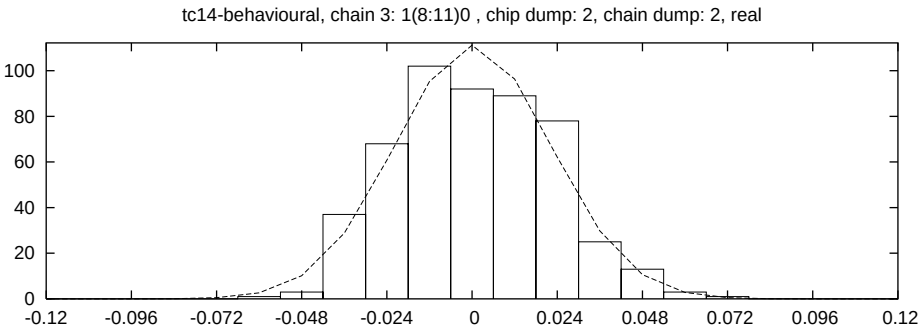

Illustration 2.5.246

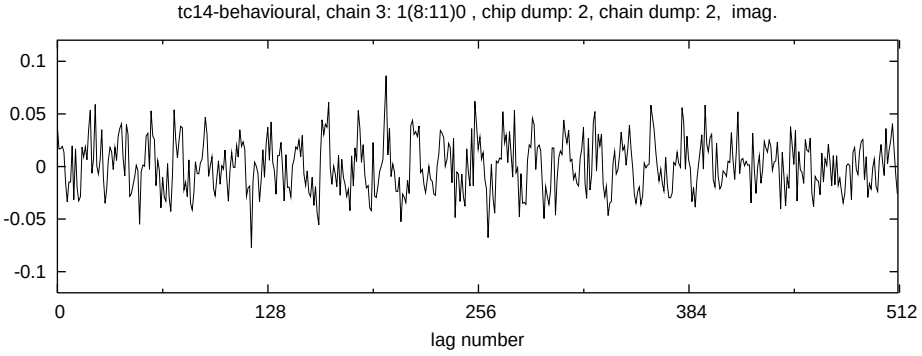

Illustration 2.5.247

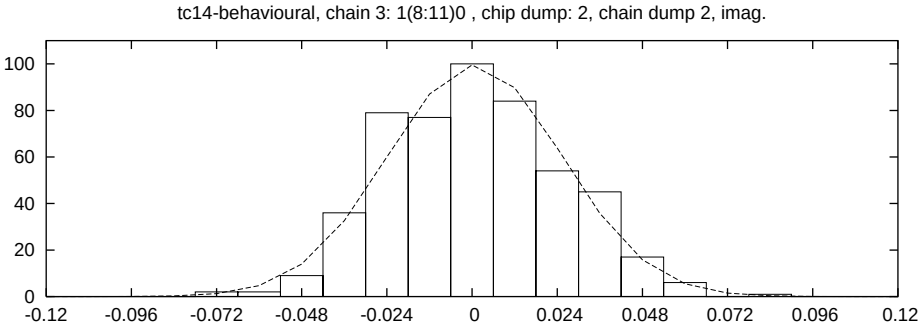

Illustration 2.5.248

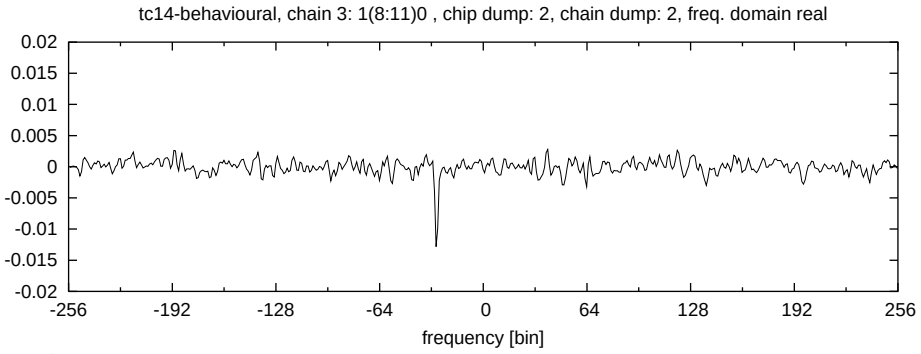

Illustration 2.5.249

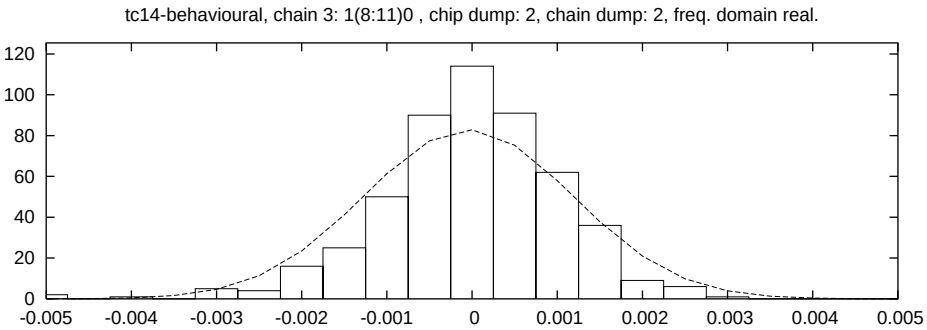

Illustration 2.5.250

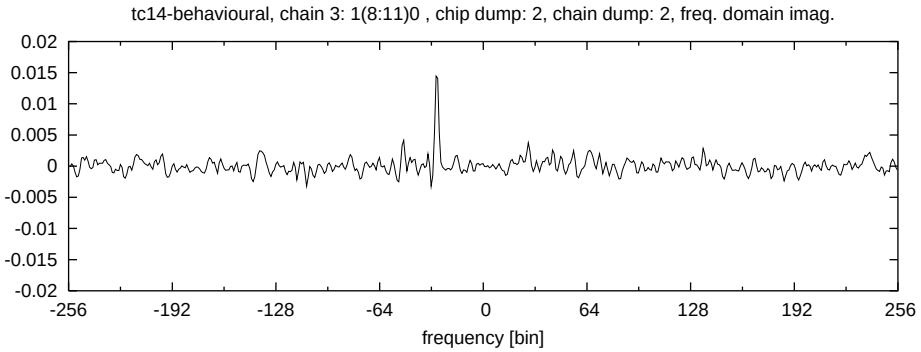

Illustration 2.5.251

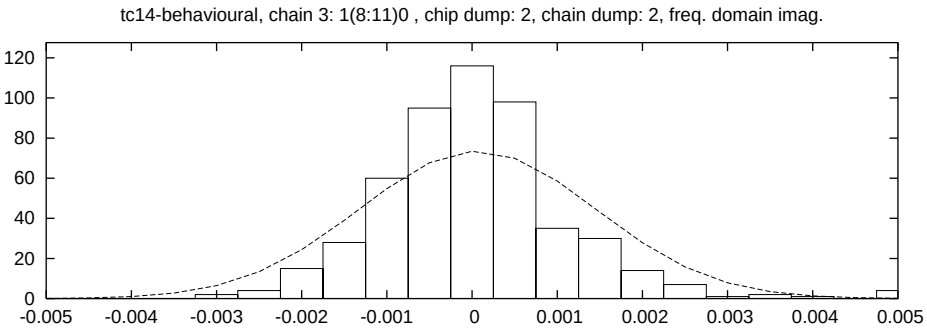

Illustration 2.5.252

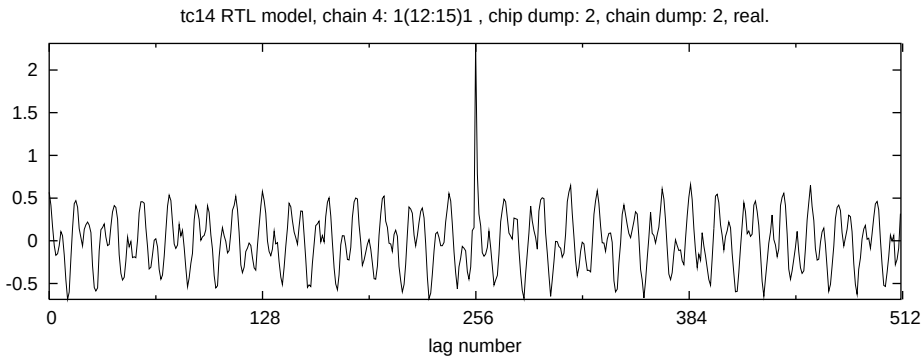

Illustration 2.5.253

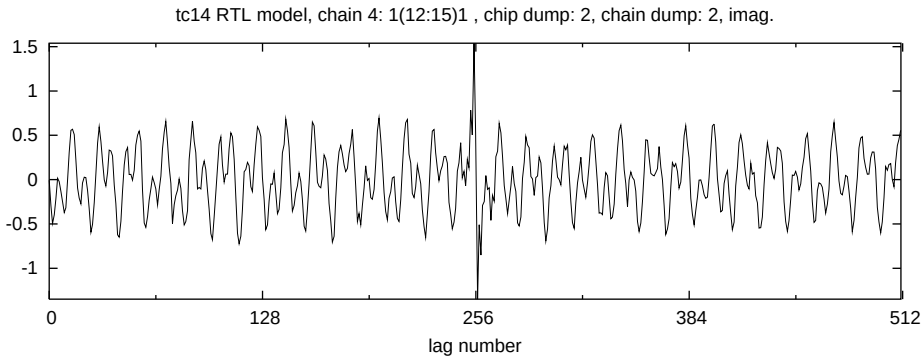

Illustration 2.5.254

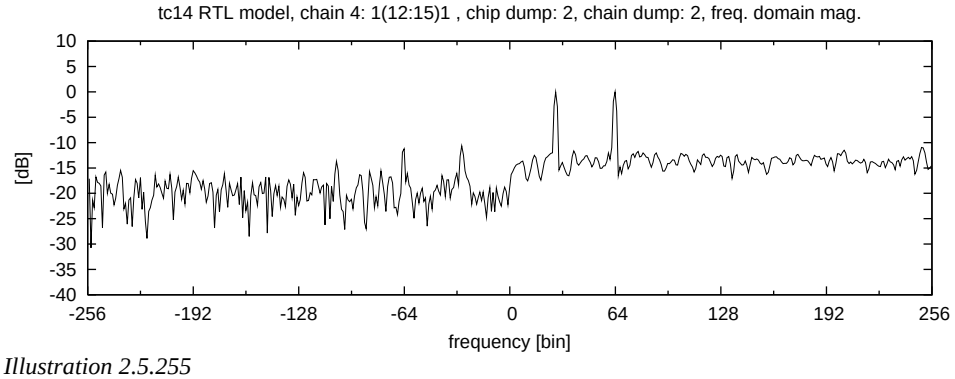

Illustration 2.5.255

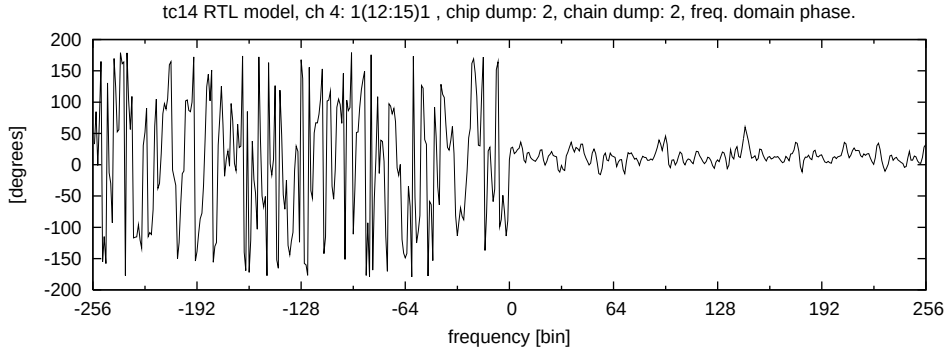

Illustration 2.5.256

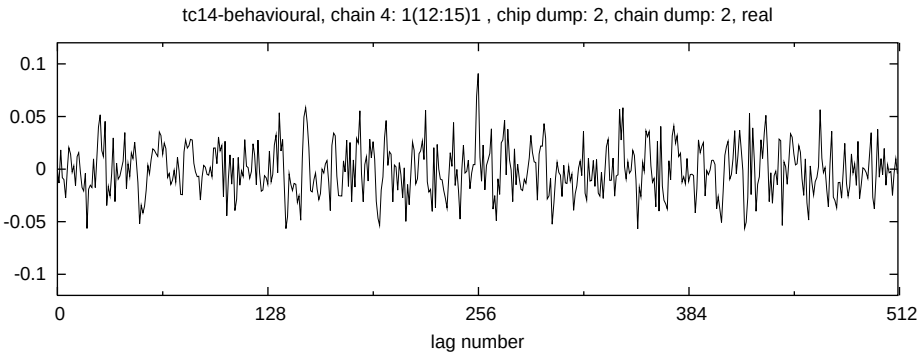

Illustration 2.5.257

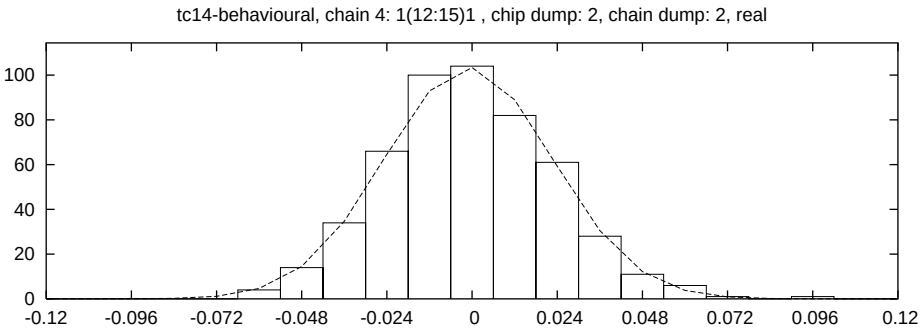

Illustration 2.5.258

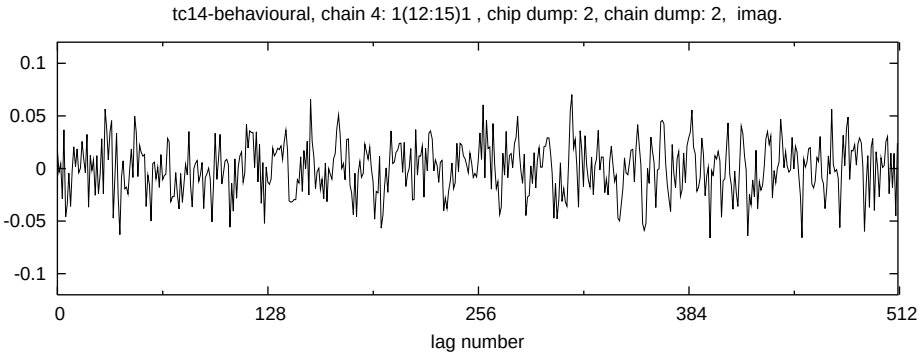

Illustration 2.5.259

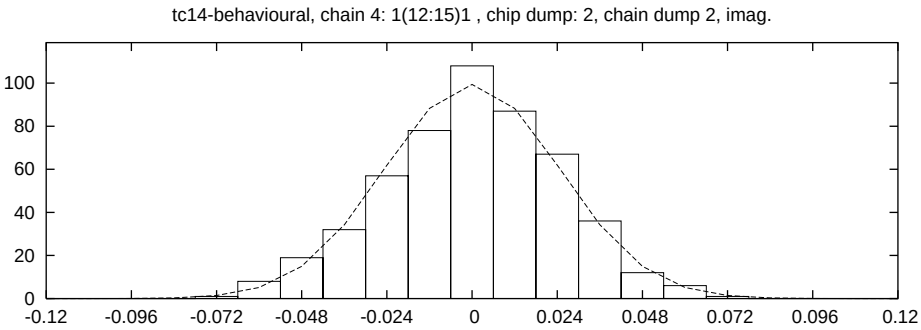

Illustration 2.5.260

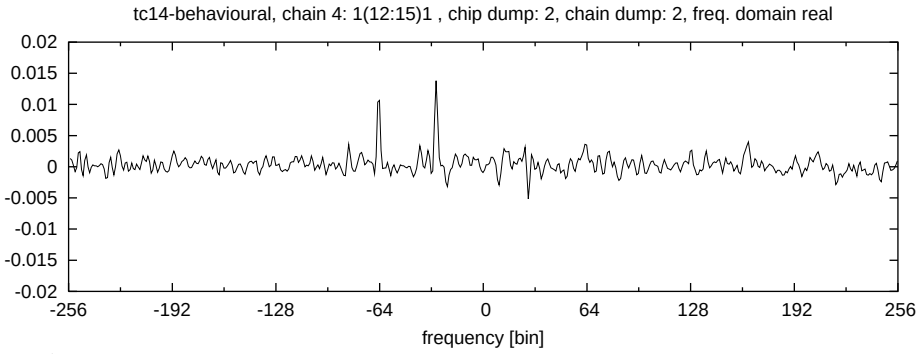

Illustration 2.5.261

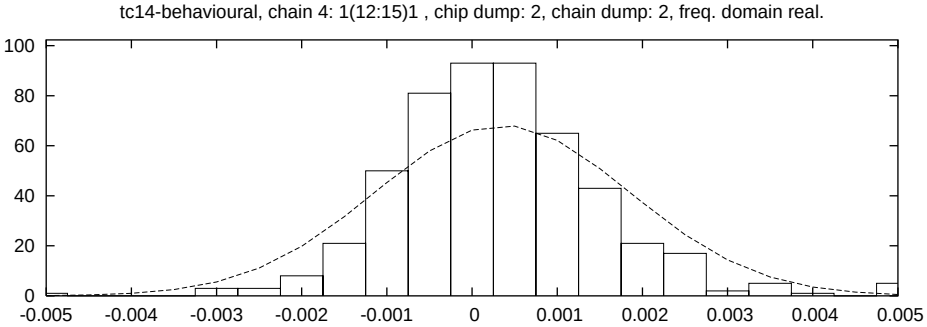

Illustration 2.5.262

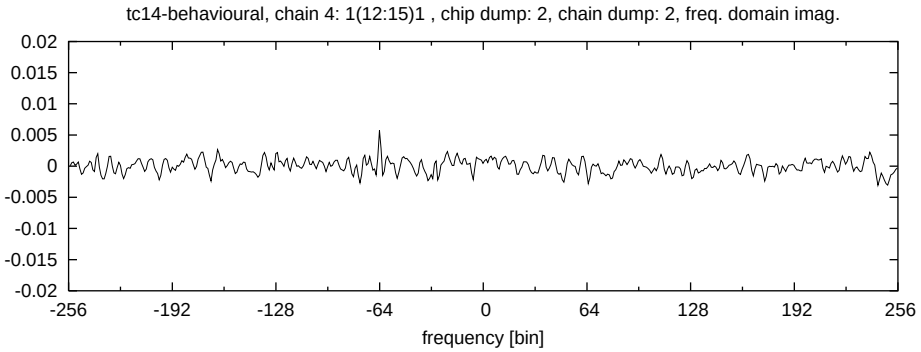

Illustration 2.5.263

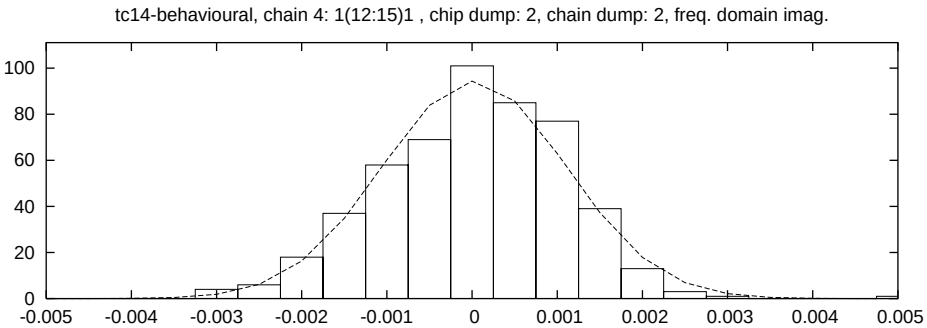

Illustration 2.5.264

2.5.12 test case 25

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged. However, a slightly elevated normalized standard deviation and peak count in the lag and frequency domains respectively (see note below) prompt the inclusion of plots in Illustration 2.5.265 to Illustration 2.5.312. Chains with common input data streams produced identical lag data which compares on a bit exact basis (Text 2.5.28).

Notes: Peaks in the frequency domain model difference plots are due to significant power at the negative frequency image of the main spectral line within the behavioural model that is absent in the RTL model. These peaks affect the high standard deviation calculated for some chains. This power in the negative frequency domain is not a concern; see the discussion in sections 2.7.7 and 2.7.8.

```
***--------------------------------------------------------------------------***
tc25_postsim.m : V2.00 run time: Sun Sep 04 13:30:32 2005
Using postsim_lib: 1.40: 05-08-18

160 frames read for test case 25.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc25
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
octave:3> tc25_chain_similarity_check

No differences exceeding the floating point precision limit of this
machine were found between lag data of similar chains for any of the 10 dumps.
```

Text 2.5.28 console log for scripts tc25_postsim, tc25_chain_similarity_check.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	-2.16e-04	-6.90e-04	0.2278	0.2097	0.0332	0.0377	1.17	1.33	1	
1	2	-1.10e-03	-9.73e-05	0.1794	0.1881	0.0363	0.0351	1.28	1.24	0	*
1	3	-1.30e-03	-5.47e-04	0.1979	0.1708	0.0348	0.0315	1.23	1.11	0	
1	4	1.51e-04	-2.59e-06	0.1921	0.1771	0.0369	0.0315	1.31	1.11	0	
1	5	2.84e-03	-8.11e-04	0.1428	0.1928	0.0275	0.0301	0.97	1.07	2	
1	6	-2.89e-05	-1.63e-04	0.1614	0.1747	0.0329	0.0344	1.16	1.22	0	
1	7	3.45e-05	2.87e-03	0.2095	0.1901	0.0340	0.0318	1.20	1.12	2	
1	8	-2.50e-03	2.24e-04	0.2393	0.1790	0.0340	0.0302	1.20	1.07	3	
1	9	2.82e-03	-2.04e-04	0.1969	0.1838	0.0364	0.0326	1.29	1.15	0	
1	10	-1.22e-03	-9.92e-04	0.2495	0.1752	0.0370	0.0334	1.31	1.18	1	
2	1	3.15e-03	9.15e-05	0.1979	0.2075	0.0402	0.0354	1.42	1.25	0	
2	2	-3.33e-03	-9.36e-04	0.2538	0.3234	0.0442	0.0502	1.56	1.78	3	
2	3	-2.18e-04	2.96e-03	0.2606	0.2411	0.0480	0.0396	1.70	1.40	1	
2	4	-1.25e-03	-7.91e-04	0.1728	0.2169	0.0282	0.0390	1.00	1.38	2	
2	5	2.24e-03	3.05e-04	0.2718	0.2973	0.0446	0.0471	1.58	1.66	2	*
2	6	-6.89e-04	1.18e-03	0.1852	0.3283	0.0339	0.0529	1.20	1.87	2	
2	7	4.09e-04	-1.27e-03	0.2550	0.2390	0.0326	0.0363	1.15	1.28	4	
2	8	1.58e-03	1.87e-03	0.2422	0.2439	0.0434	0.0374	1.53	1.32	3	
2	9	-2.47e-03	1.76e-03	0.2485	0.2488	0.0452	0.0322	1.60	1.14	1	
2	10	4.77e-03	3.29e-03	0.2415	0.2327	0.0348	0.0416	1.23	1.47	4	
3	1	5.41e-04	-2.87e-03	0.1863	0.1980	0.0297	0.0355	1.05	1.26	3	
3	2	3.19e-04	1.56e-03	0.2139	0.2393	0.0342	0.0470	1.21	1.66	2	*
3	3	-1.57e-03	-1.29e-03	0.1759	0.2000	0.0338	0.0368	1.20	1.30	0	
3	4	1.37e-03	-7.30e-04	0.1796	0.2106	0.0333	0.0345	1.18	1.22	1	
3	5	2.77e-04	-1.22e-03	0.1721	0.1752	0.0323	0.0357	1.14	1.26	0	
3	6	2.07e-03	-1.36e-03	0.2036	0.1847	0.0333	0.0345	1.18	1.22	1	
3	7	-8.35e-04	1.12e-03	0.1976	0.1964	0.0340	0.0318	1.20	1.13	2	
3	8	-1.50e-03	-8.84e-05	0.1787	0.2142	0.0333	0.0366	1.18	1.29	3	
3	9	2.28e-03	4.85e-04	0.1897	0.2144	0.0355	0.0358	1.25	1.27	1	
3	10	-9.83e-04	-8.55e-04	0.2415	0.2099	0.0425	0.0378	1.50	1.34	0	
4	1	2.29e-03	-3.43e-04	0.2145	0.1772	0.0404	0.0319	1.43	1.13	1	
4	2	1.92e-04	-4.64e-04	0.2030	0.2243	0.0375	0.0386	1.32	1.37	2	
4	3	-5.63e-04	-3.71e-04	0.2880	0.2273	0.0559	0.0424	1.98	1.50	0	*
4	4	3.53e-04	1.03e-03	0.2010	0.2025	0.0369	0.0369	1.30	1.31	3	
4	5	1.37e-03	3.10e-04	0.2351	0.2195	0.0391	0.0396	1.38	1.40	1	
4	6	1.61e-03	-1.02e-03	0.2151	0.2566	0.0448	0.0428	1.58	1.51	1	
4	7	-8.61e-04	9.26e-04	0.2683	0.1766	0.0452	0.0307	1.60	1.08	2	
4	8	-1.59e-03	-2.20e-03	0.1929	0.2121	0.0365	0.0413	1.29	1.46	0	
4	9	1.16e-03	1.78e-03	0.2496	0.2474	0.0377	0.0442	1.33	1.56	3	
4	10	1.41e-03	-8.92e-04	0.2217	0.1738	0.0413	0.0298	1.46	1.05	1	

Table 2.5.46 test case 25; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-3.57e-03	-1.11e-02	0.6247	0.6740	0.1207	0.1187	1.00	1.01	null
1	2	-2.29e-03	8.09e-03	0.6374	0.6126	0.1140	0.1206	0.93	0.99	null
1	3	1.37e-04	-3.70e-03	0.6394	0.6379	0.1274	0.1081	0.92	0.96	null
1	4	2.00e-03	2.17e-04	0.7001	0.5887	0.1185	0.1027	0.94	0.95	null
1	5	9.66e-03	4.70e-03	0.5988	0.6127	0.1185	0.1151	1.00	1.02	null
1	6	-7.07e-03	3.49e-03	0.7206	0.8505	0.1262	0.1305	1.07	1.11	null
1	7	-4.13e-03	-1.99e-03	0.6861	0.6920	0.1177	0.1202	1.06	1.09	null
1	8	-1.80e-03	-3.90e-03	0.7222	0.6786	0.1174	0.1208	1.07	1.03	null
1	9	7.45e-03	-7.52e-04	0.6919	0.6703	0.1271	0.1119	0.99	0.94	null
2	1	4.69e-03	3.56e-03	0.6804	0.6052	0.1212	0.1163	1.07	1.10	null
2	2	-7.78e-03	4.54e-03	0.7090	0.6755	0.1350	0.1297	1.20	1.13	128 129
2	3	-3.82e-03	-9.90e-03	0.6955	0.6574	0.1120	0.1247	1.06	1.11	null
2	4	1.02e-02	2.05e-02	0.7414	0.6266	0.1256	0.1053	1.16	1.00	128
2	5	-2.90e-03	-1.03e-02	0.5988	0.6860	0.1253	0.1218	1.20	1.05	127 128
2	6	-4.21e-03	-4.03e-03	0.6274	0.5046	0.1164	0.0963	1.04	0.92	128 129
2	7	5.11e-03	-8.31e-04	0.6050	0.6220	0.1144	0.1107	0.98	1.01	128
2	8	1.44e-02	4.92e-03	0.6919	0.6667	0.1127	0.1225	0.96	1.02	128
2	9	-6.69e-03	-3.41e-03	0.6453	0.6531	0.1207	0.1186	1.05	1.06	128
3	1	9.34e-04	5.58e-03	0.5433	0.7268	0.1134	0.1185	1.09	0.99	null
3	2	-9.35e-03	-3.28e-04	0.6490	0.6506	0.1301	0.1293	1.04	1.01	null
3	3	6.80e-03	-5.66e-04	0.6935	0.6751	0.1315	0.1297	1.09	1.00	null
3	4	2.12e-03	1.55e-03	0.6253	0.5925	0.1187	0.1158	1.04	1.02	null
3	5	-3.65e-03	-2.71e-03	0.6414	0.7907	0.1253	0.1312	1.06	1.04	null
3	6	-1.16e-03	5.28e-03	0.7138	0.6949	0.1234	0.1188	1.00	0.96	null
3	7	-4.44e-03	-8.59e-03	0.6147	0.6231	0.1146	0.1117	1.03	1.04	null
3	8	1.41e-02	1.04e-02	0.6655	0.6293	0.1085	0.1212	1.01	1.04	null
3	9	-1.41e-02	-3.62e-03	0.5975	0.6167	0.1109	0.1255	1.03	1.03	null
4	1	-7.95e-03	2.77e-03	0.6427	0.6022	0.1279	0.1122	1.11	0.96	null
4	2	4.20e-04	-4.15e-04	0.6500	0.7437	0.1283	0.1083	1.06	0.90	208
4	3	4.52e-03	6.00e-03	0.7293	0.7408	0.1179	0.1193	1.06	1.10	null
4	4	-3.41e-03	-1.39e-02	0.6058	0.6303	0.1099	0.1140	0.97	1.01	149
4	5	7.00e-03	2.06e-03	0.6489	0.5794	0.1232	0.1102	0.94	0.96	null
4	6	-8.24e-03	7.78e-03	0.6117	0.6839	0.1103	0.1190	0.97	1.14	null
4	7	-3.61e-03	1.36e-03	0.6333	0.6628	0.1153	0.1131	0.93	0.89	null
4	8	7.69e-03	1.29e-03	0.6304	0.6788	0.1115	0.1241	0.91	1.11	null
4	9	-4.14e-04	-1.20e-02	0.5890	0.6840	0.1169	0.1229	0.93	1.04	null

Table 2.5.47 test case 25; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	1.18e-03	-4.97e-04	0.0230	0.0142	2.96e-03	2.21e-03	3.5	2.6	0	
1	2	-7.03e-04	5.58e-04	0.0244	0.0166	2.97e-03	2.31e-03	3.5	2.7	0	*
1	3	3.71e-04	-3.04e-04	0.0136	0.0137	2.43e-03	2.38e-03	2.9	2.8	0	
1	4	3.32e-04	3.81e-04	0.0215	0.0202	2.62e-03	2.43e-03	3.1	2.9	0	
1	5	-5.13e-04	-8.57e-05	0.0138	0.0129	2.29e-03	2.22e-03	2.7	2.6	0	
1	6	5.48e-04	-3.74e-04	0.0282	0.0137	3.05e-03	2.04e-03	3.6	2.4	0	
1	7	-3.68e-04	3.06e-04	0.0215	0.0129	2.88e-03	2.08e-03	3.4	2.4	0	
1	8	-9.69e-05	-3.07e-04	0.0147	0.0203	2.53e-03	2.35e-03	3.0	2.8	0	
1	9	6.33e-04	2.52e-04	0.0121	0.0177	2.07e-03	2.83e-03	2.4	3.3	0	
1	10	-3.22e-04	2.45e-04	0.0190	0.0202	2.80e-03	2.58e-03	3.3	3.0	0	
2	1	5.74e-04	7.30e-04	0.0259	0.0344	2.84e-03	3.17e-03	3.3	3.7	0	
2	2	4.45e-04	1.74e-03	0.0210	0.0474	2.93e-03	4.28e-03	3.5	5.0	0	
2	3	-1.11e-03	1.86e-04	0.0337	0.0197	3.86e-03	2.72e-03	4.6	3.2	0	
2	4	-7.45e-04	-2.07e-04	0.0259	0.0166	2.92e-03	2.32e-03	3.4	2.7	0	
2	5	-5.68e-04	-1.52e-03	0.0330	0.0500	3.18e-03	4.08e-03	3.7	4.8	1	*
2	6	7.17e-04	-1.42e-03	0.0393	0.0254	3.53e-03	3.40e-03	4.2	4.0	0	
2	7	1.03e-03	2.76e-04	0.0245	0.0150	2.92e-03	2.49e-03	3.4	2.9	0	
2	8	5.69e-04	1.11e-03	0.0392	0.0271	3.50e-03	2.77e-03	4.1	3.3	0	
2	9	-2.22e-04	1.30e-03	0.0227	0.0273	2.83e-03	3.21e-03	3.3	3.8	0	
2	10	-8.25e-04	9.41e-04	0.0359	0.0256	3.39e-03	2.76e-03	4.0	3.2	0	
3	1	4.07e-04	1.75e-04	0.0144	0.0153	2.62e-03	2.51e-03	3.1	3.0	0	
3	2	-3.77e-04	1.19e-04	0.0272	0.0287	2.95e-03	3.54e-03	3.5	4.2	0	*
3	3	1.77e-04	1.59e-04	0.0204	0.0284	2.71e-03	2.82e-03	3.2	3.3	0	
3	4	2.28e-04	-4.60e-04	0.0140	0.0209	2.37e-03	2.65e-03	2.8	3.1	0	
3	5	-8.98e-05	1.31e-04	0.0208	0.0175	2.56e-03	2.51e-03	3.0	3.0	0	
3	6	-2.01e-05	-1.95e-04	0.0156	0.0244	2.19e-03	2.67e-03	2.6	3.1	0	
3	7	5.39e-05	7.82e-05	0.0160	0.0167	2.60e-03	2.46e-03	3.1	2.9	0	
3	8	-4.13e-04	-2.24e-05	0.0230	0.0214	2.58e-03	2.74e-03	3.0	3.2	0	
3	9	1.12e-04	-9.67e-05	0.0193	0.0230	2.62e-03	2.62e-03	3.1	3.1	0	
3	10	5.52e-04	-2.05e-06	0.0270	0.0317	3.40e-03	3.01e-03	4.0	3.5	0	
4	1	2.89e-04	-3.18e-04	0.0269	0.0284	2.83e-03	2.75e-03	3.3	3.2	0	
4	2	1.08e-04	3.67e-04	0.0273	0.0319	2.94e-03	2.93e-03	3.5	3.5	0	
4	3	-5.61e-05	-3.18e-04	0.0186	0.0617	2.51e-03	4.73e-03	3.0	5.6	1	*
4	4	1.86e-05	-1.70e-04	0.0272	0.0180	2.99e-03	2.64e-03	3.5	3.1	0	
4	5	-1.03e-04	5.50e-04	0.0172	0.0382	2.54e-03	3.40e-03	3.0	4.0	0	
4	6	2.90e-04	-3.69e-04	0.0287	0.0403	3.18e-03	3.63e-03	3.7	4.3	0	
4	7	-2.54e-04	1.41e-04	0.0316	0.0163	3.26e-03	2.27e-03	3.8	2.7	0	
4	8	4.62e-04	1.49e-04	0.0240	0.0308	2.74e-03	3.04e-03	3.2	3.6	0	
4	9	-2.45e-04	-5.69e-04	0.0219	0.0360	2.76e-03	3.47e-03	3.3	4.1	0	
4	10	-3.27e-04	3.16e-04	0.0345	0.0273	3.22e-03	2.70e-03	3.8	3.2	0	

Table 2.5.48 test case 25; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: ! OVR*	*bias: OVR*	*applied dump - bias: OVR*
0	12000	11724	276	0	n-a
1	12000	11724	276	0	n-a

Table 2.5.49 test case 25: applied dump and bias.

```
#-------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

01000010  CCCSCR_0_1:   CCC1: Y=pri X=adj   CCC0: Y=adj X=sec
10000001  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=pri
01000001  CCCSCR_4_5:   CCC5: Y=pri X=adj   CCC4: Y=adj X=pri
10000010  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=sec
01000010  CCCSCR_8_9:   CCC9: Y=pri X=adj   CCC8: Y=adj X=sec
10000001  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=pri
01000001  CCCSCR_12_13: CCC13: Y=pri X=adj CCC12: Y=adj X=pri
10000010  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=sec

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111100  XSTATUS: X-SE_CLK: 11111100
11111100  YSTATUS: Y-SE_CLK: 11111100
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.29 content of file mcb_reg25.txt

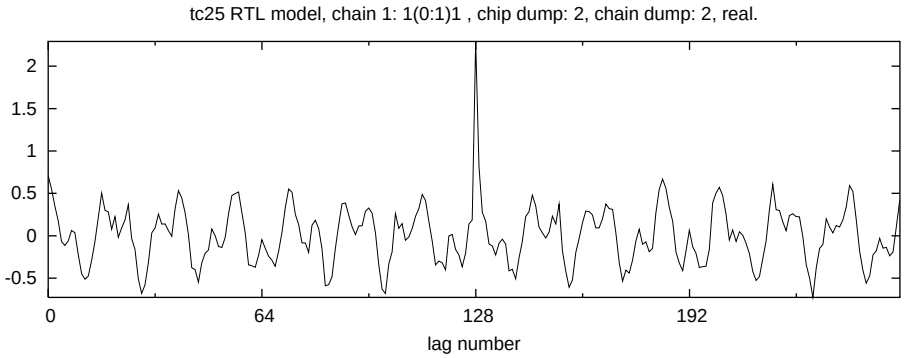

Illustration 2.5.265

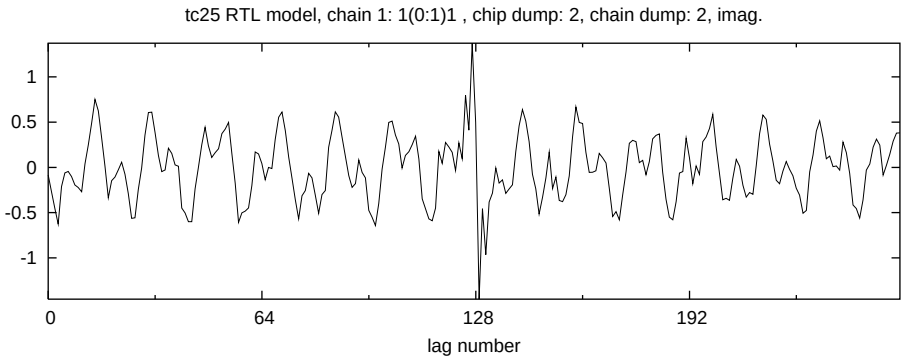

Illustration 2.5.266

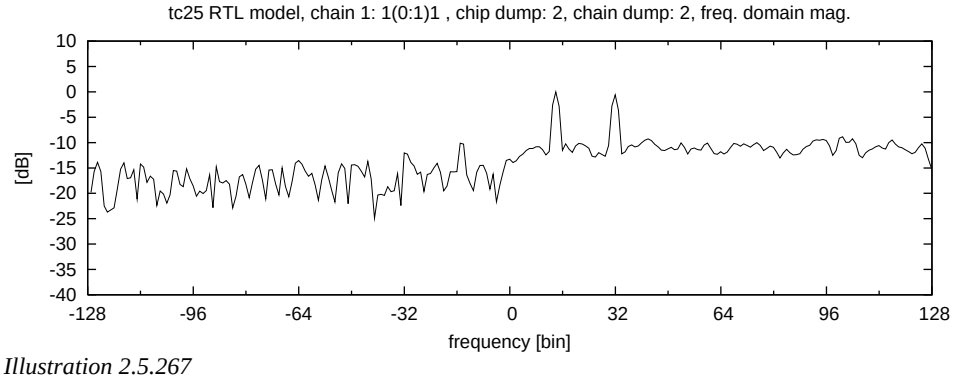

Illustration 2.5.267

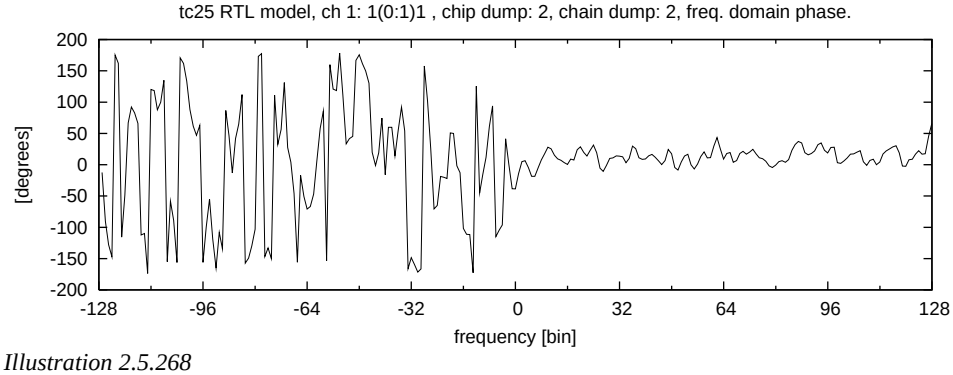

Illustration 2.5.268

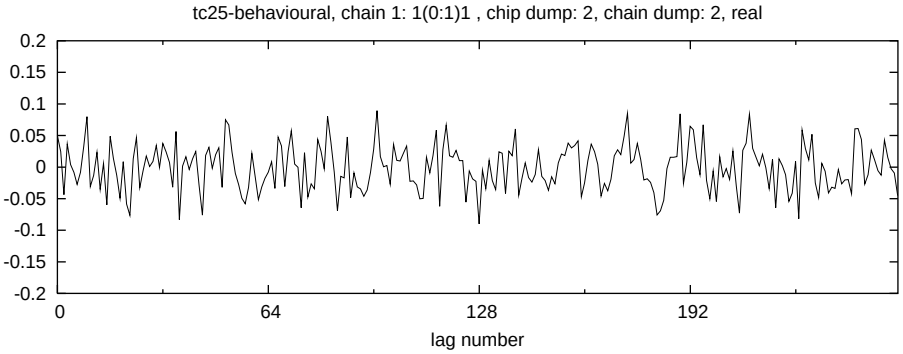

Illustration 2.5.269

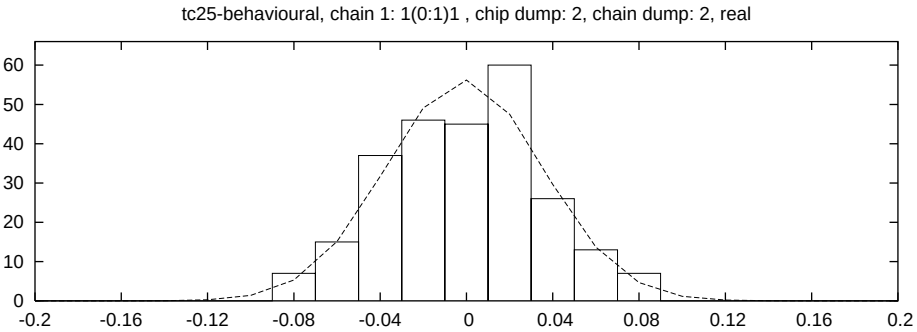

Illustration 2.5.270

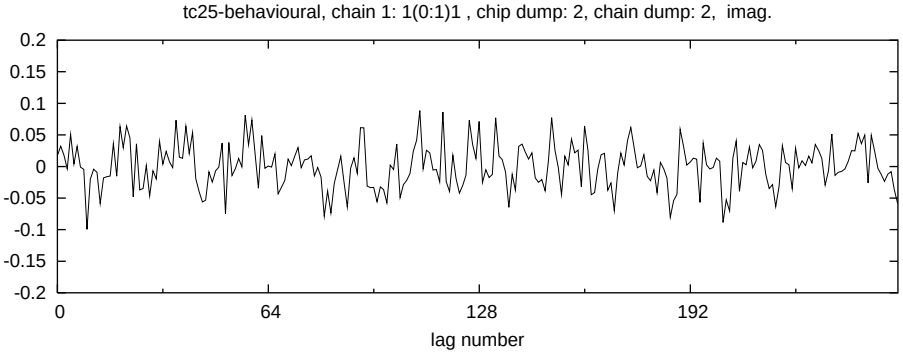

Illustration 2.5.271

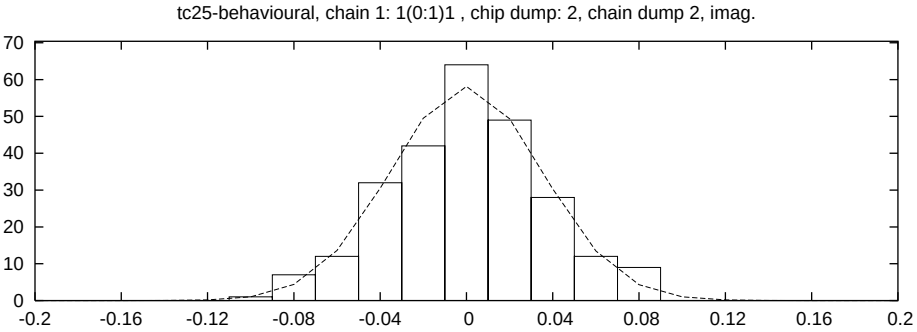

Illustration 2.5.272

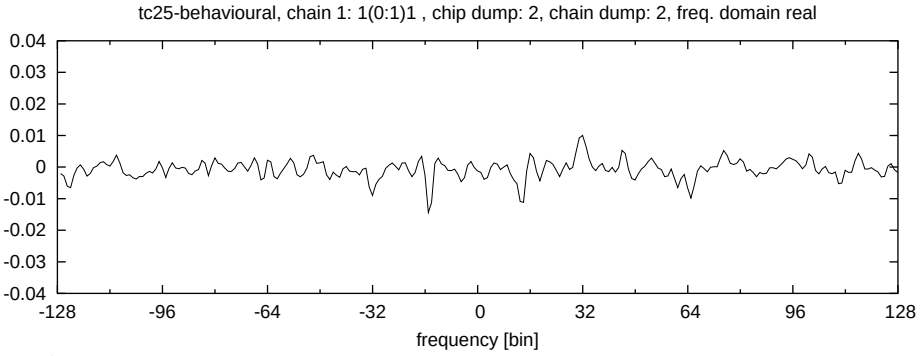

Illustration 2.5.273

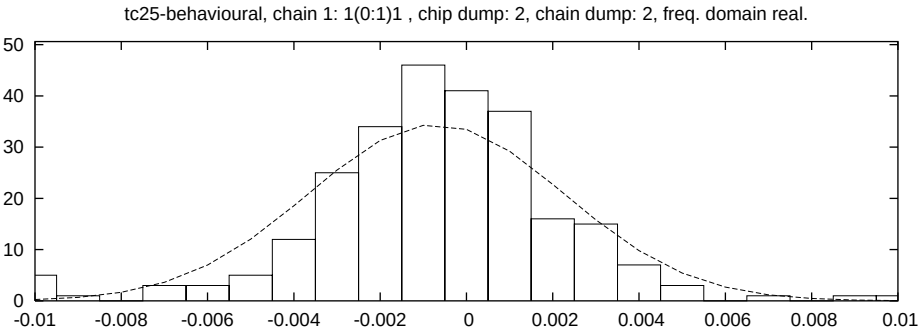

Illustration 2.5.274

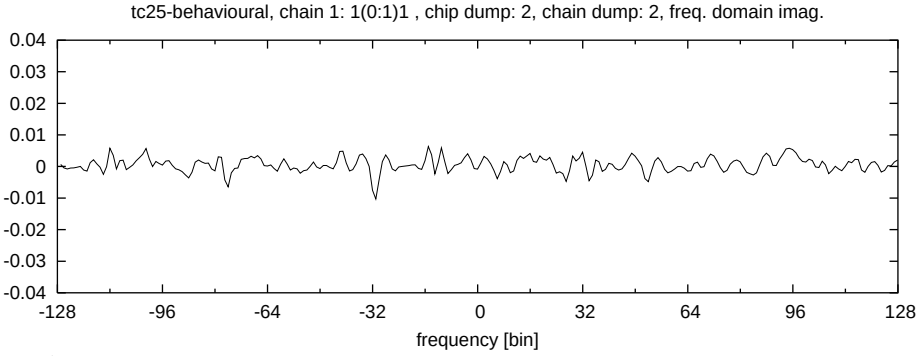

Illustration 2.5.275

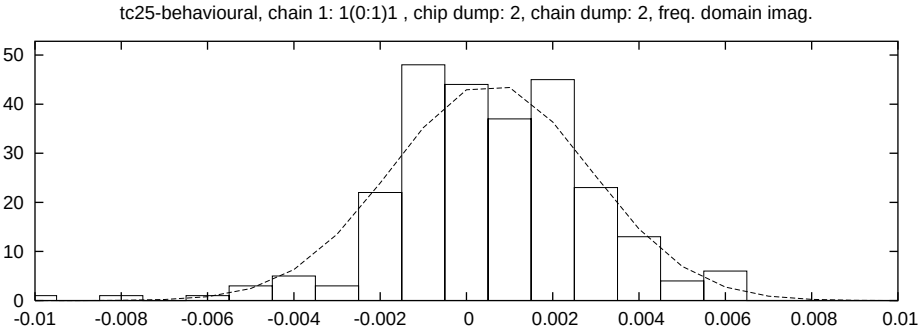

Illustration 2.5.276

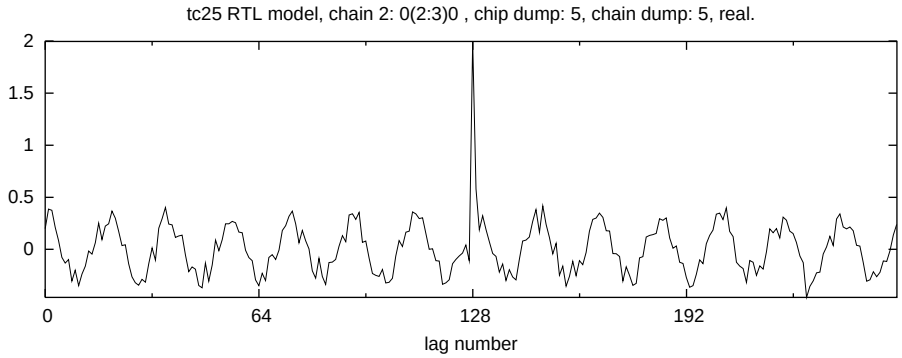

Illustration 2.5.277

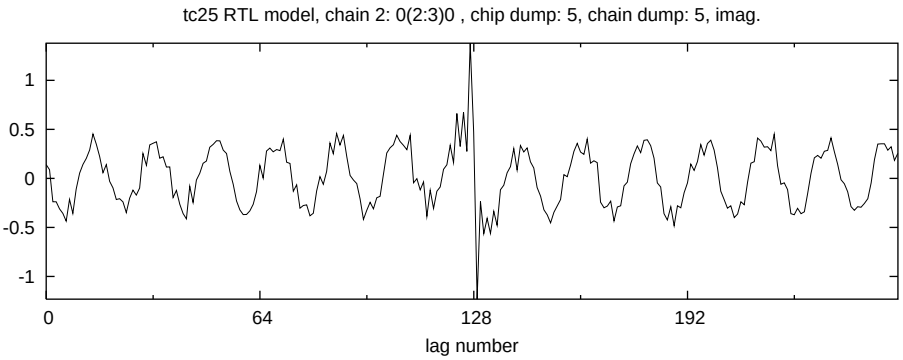

Illustration 2.5.278

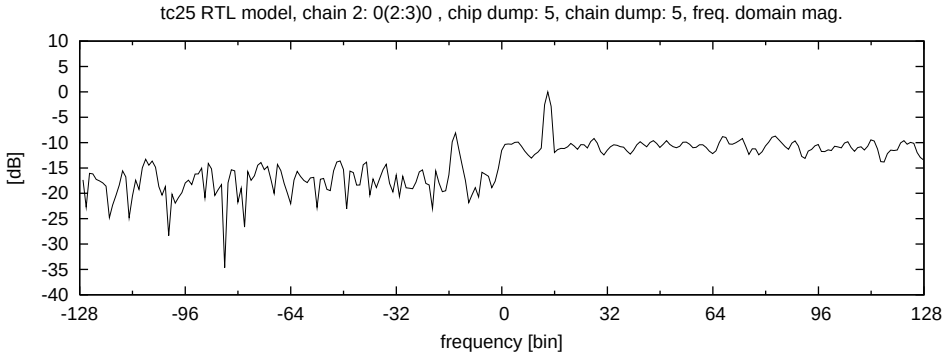

Illustration 2.5.279

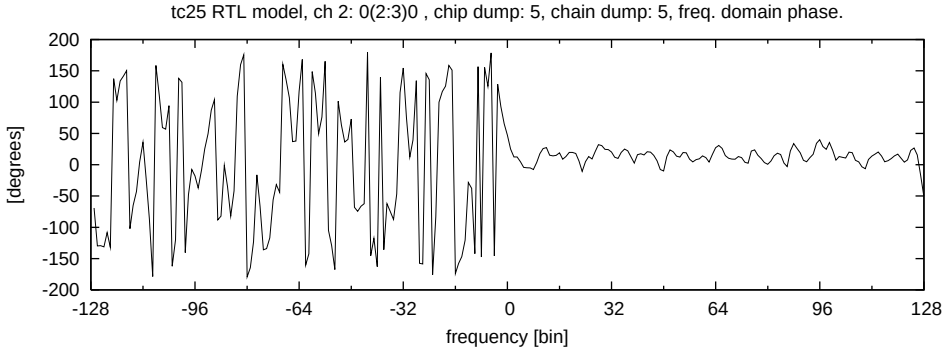

Illustration 2.5.280

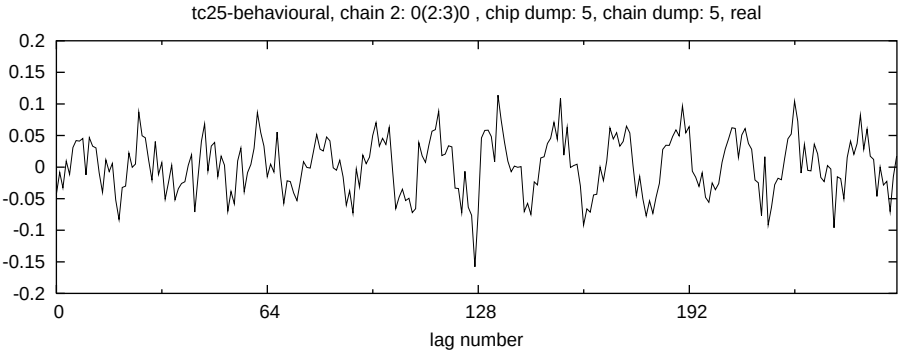

Illustration 2.5.281

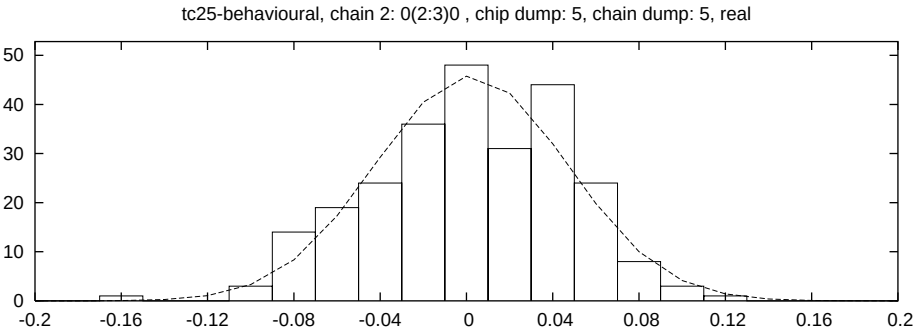

Illustration 2.5.282

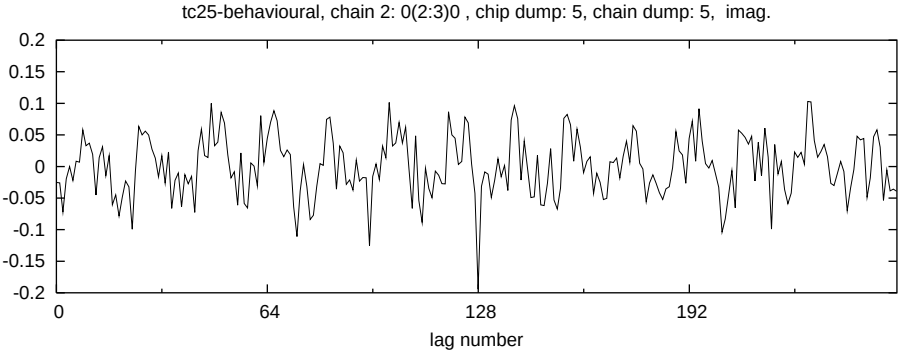

Illustration 2.5.283

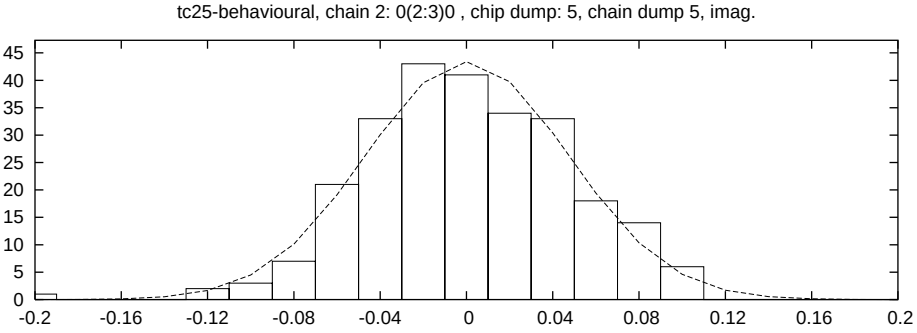

Illustration 2.5.284

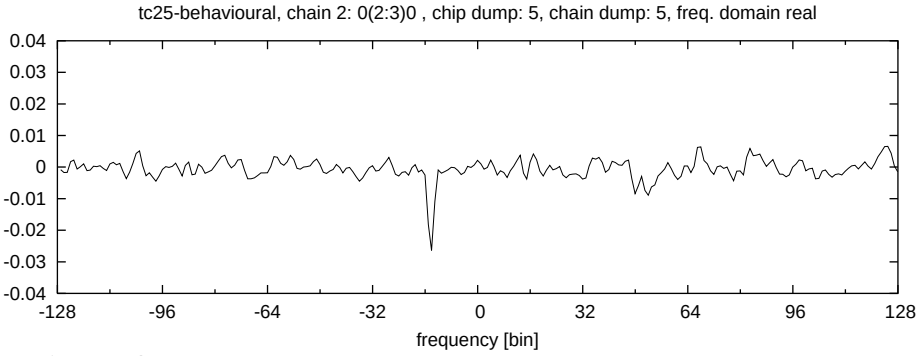

Illustration 2.5.285

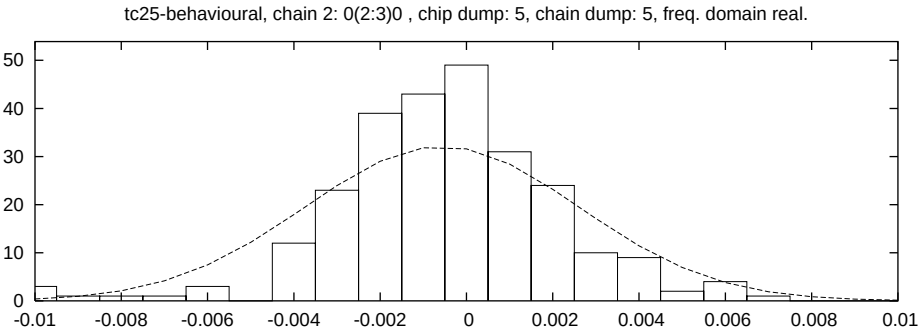

Illustration 2.5.286

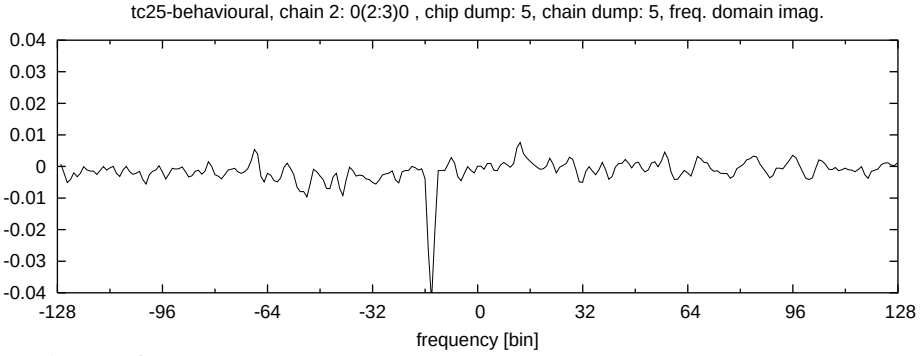

Illustration 2.5.287

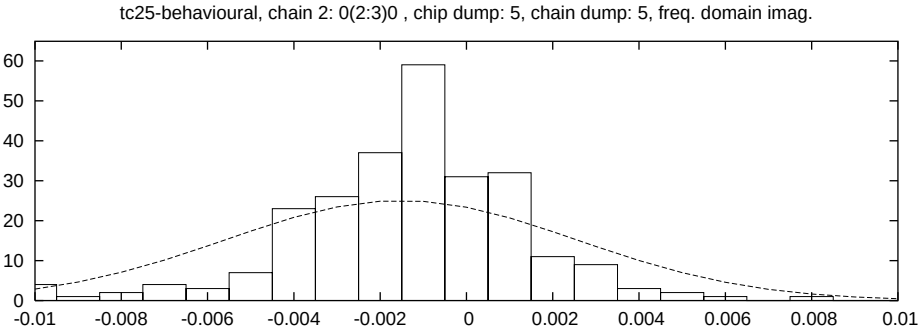

Illustration 2.5.288

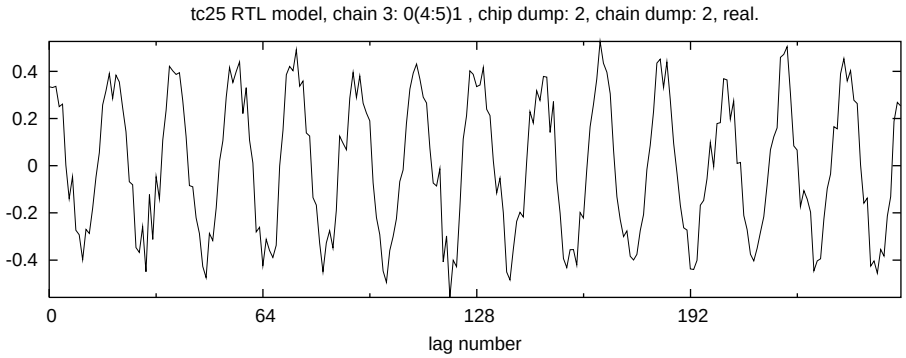

Illustration 2.5.289

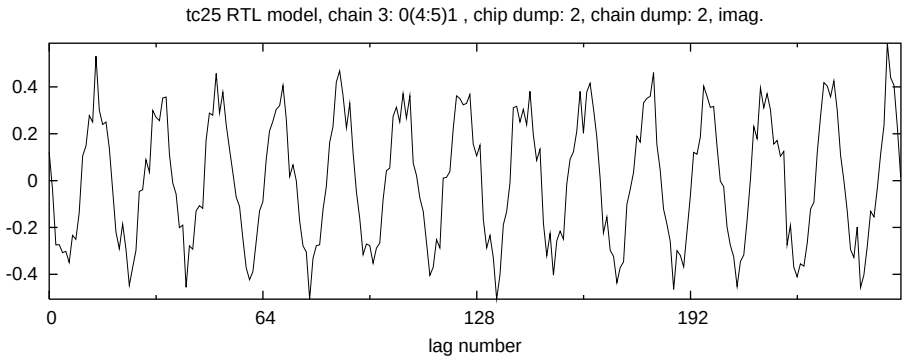

Illustration 2.5.290

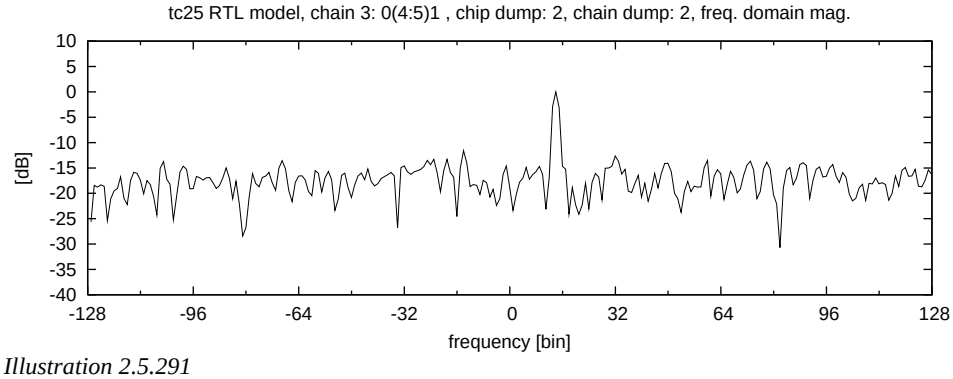

Illustration 2.5.291

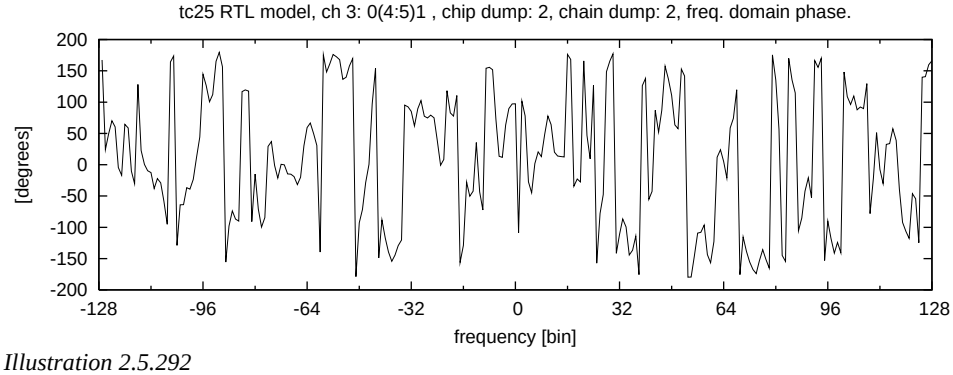

Illustration 2.5.292

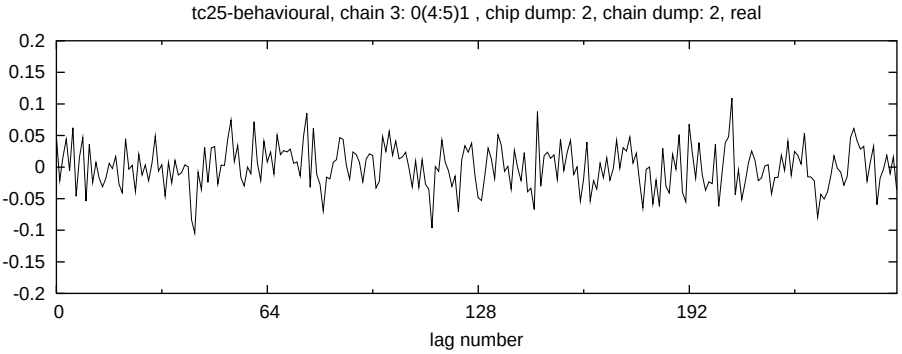

Illustration 2.5.293

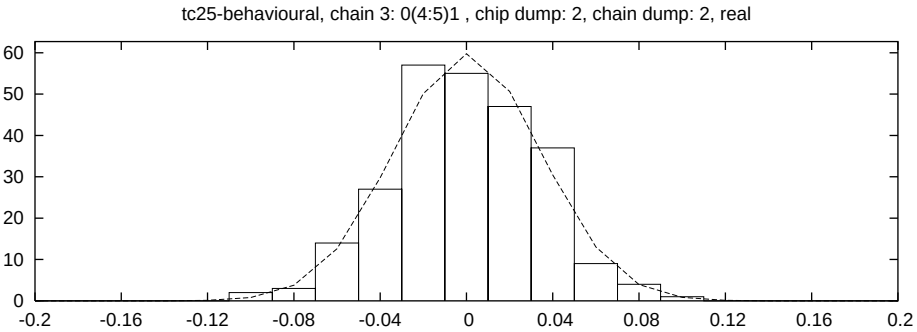

Illustration 2.5.294

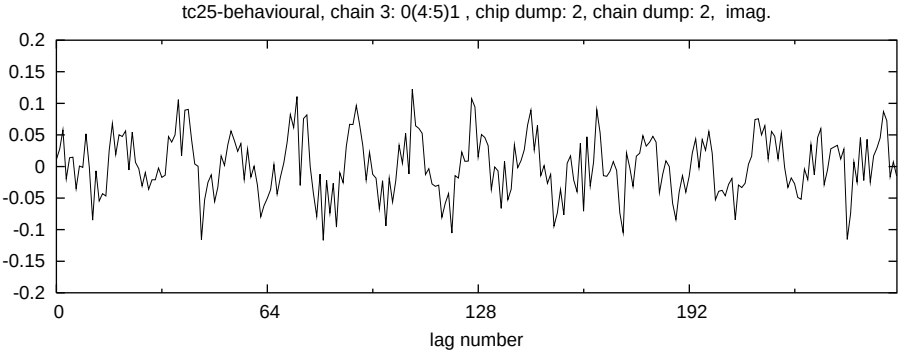

Illustration 2.5.295

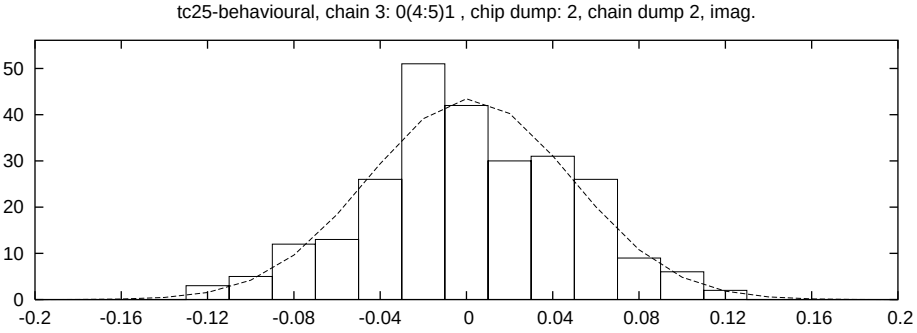

Illustration 2.5.296

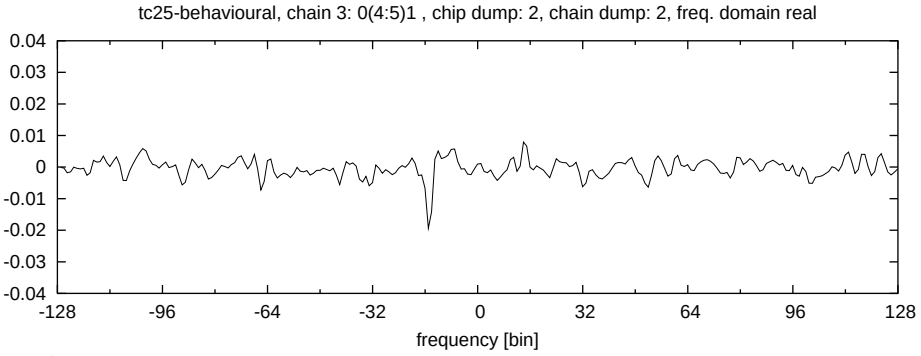

Illustration 2.5.297

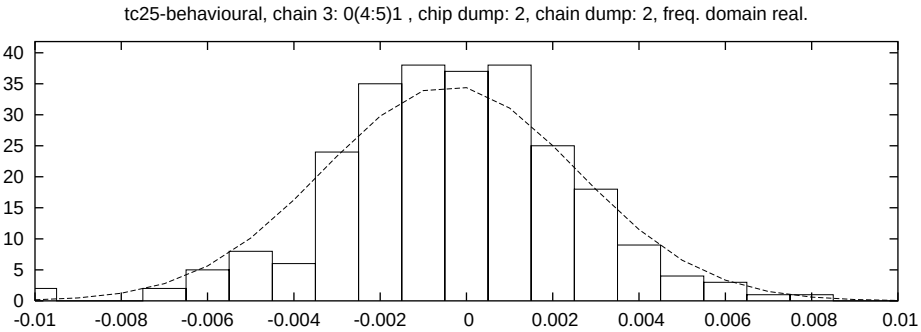

Illustration 2.5.298

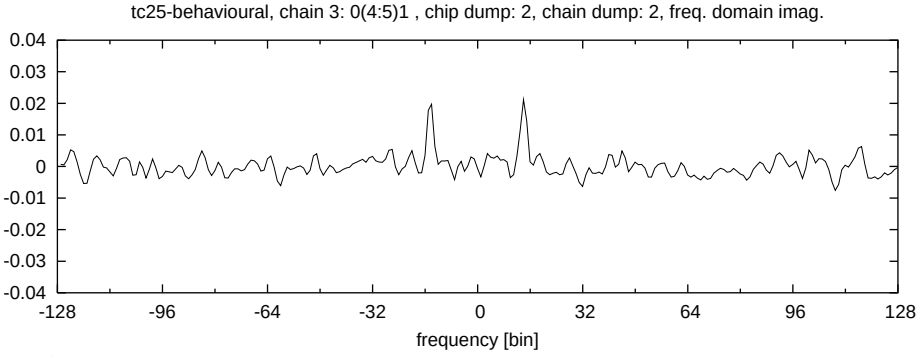

Illustration 2.5.299

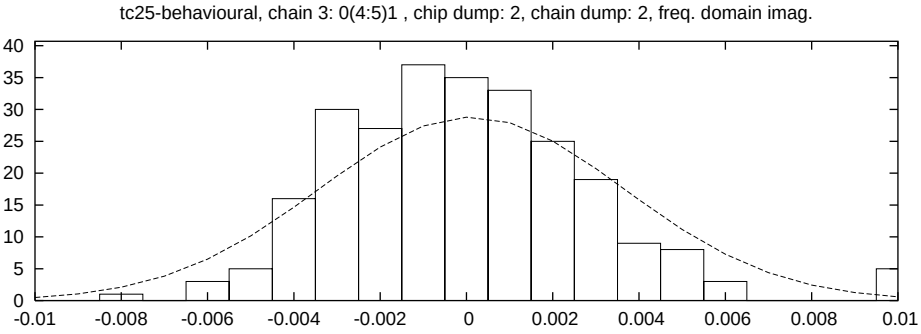

Illustration 2.5.300

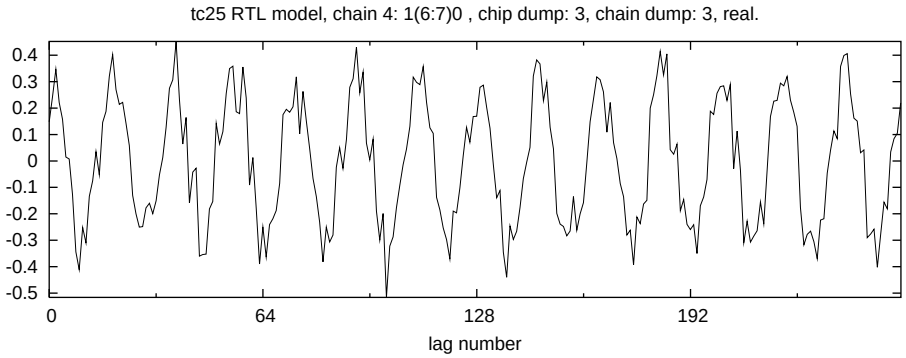

Illustration 2.5.301

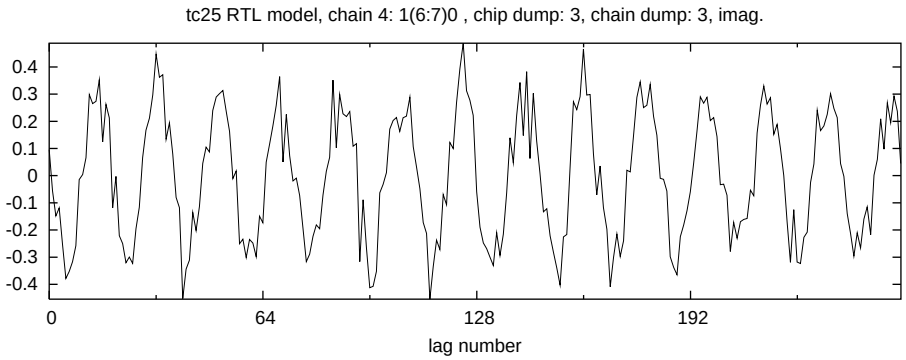

Illustration 2.5.302

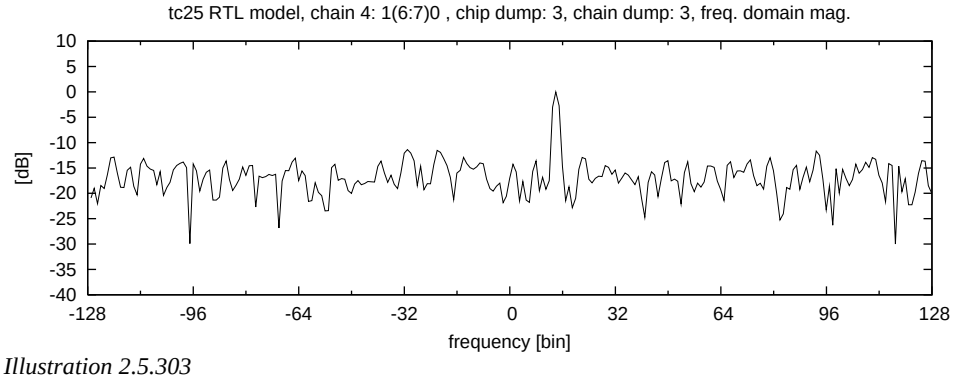

Illustration 2.5.303

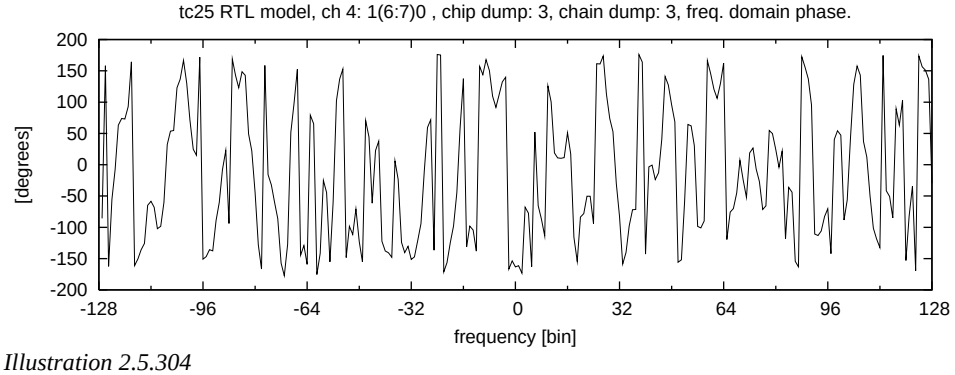

Illustration 2.5.304

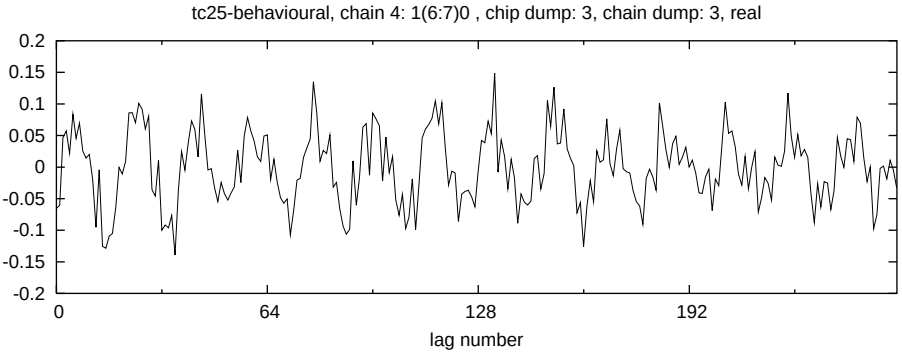

Illustration 2.5.305

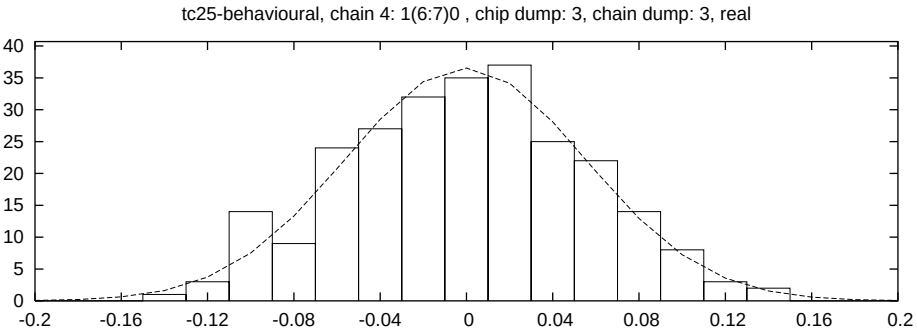

Illustration 2.5.306

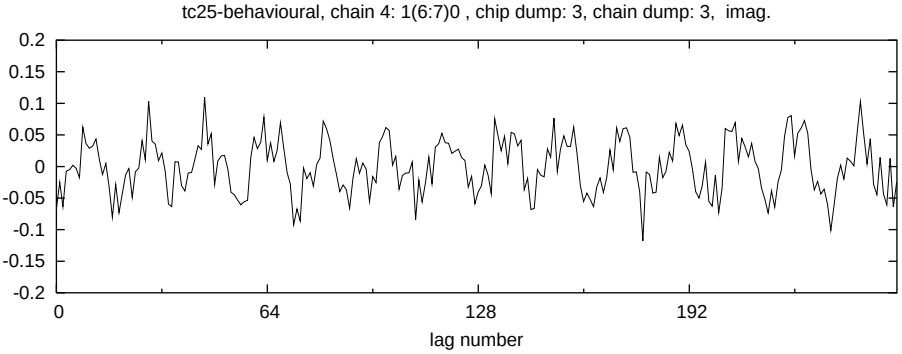

Illustration 2.5.307

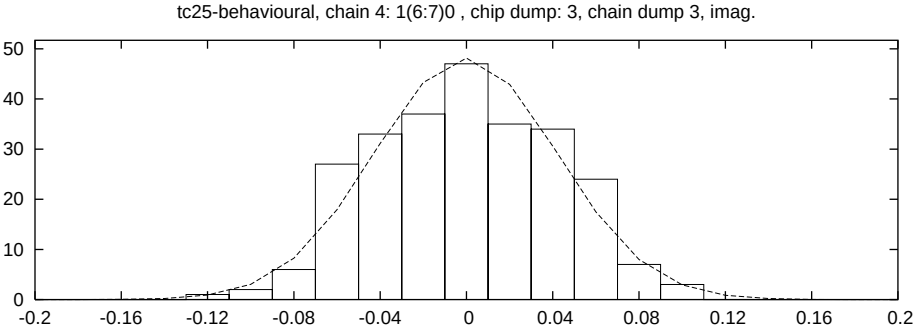

Illustration 2.5.308

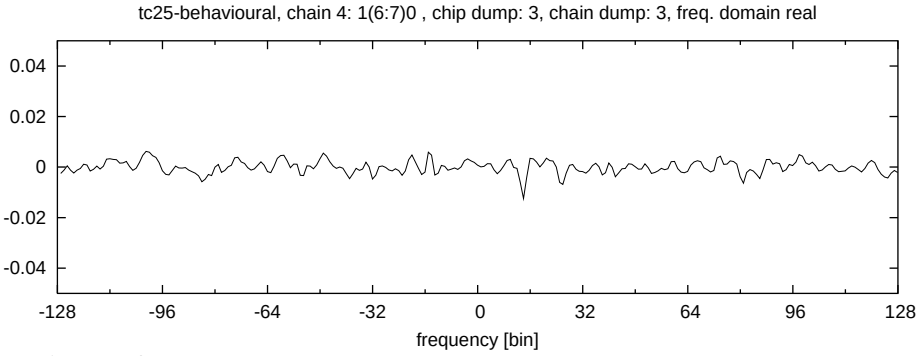

Illustration 2.5.309

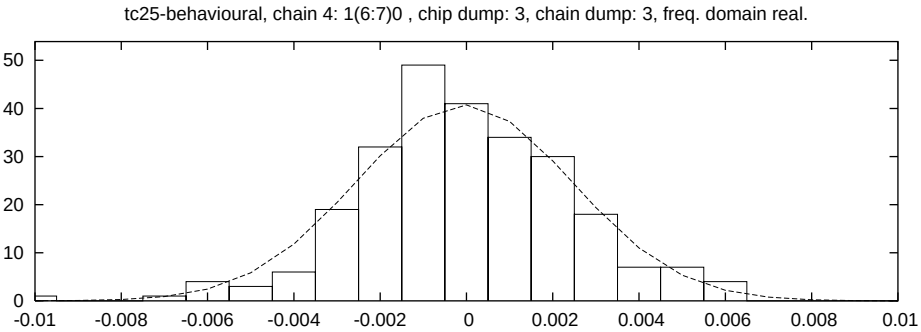

Illustration 2.5.310

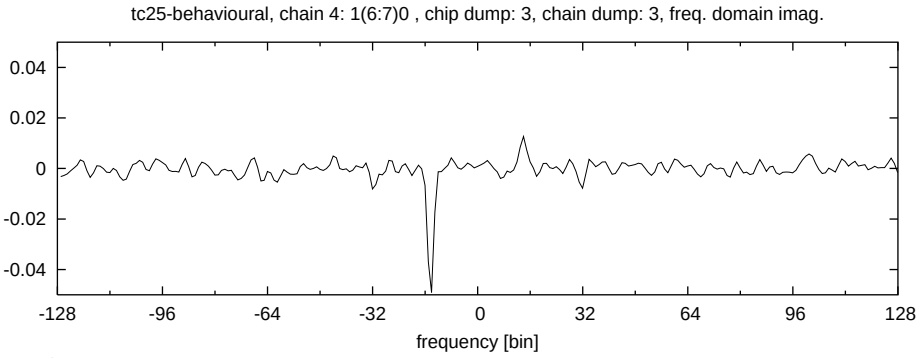

Illustration 2.5.311

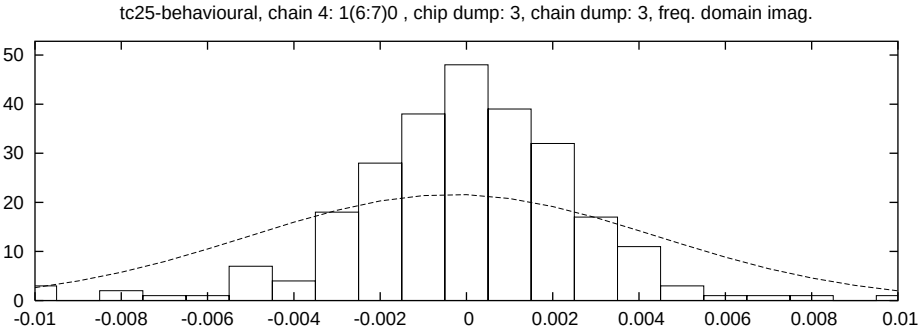

Illustration 2.5.312

2.5.13 test case 28

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc28_postsim.m : V2.00 run time: Sun Oct 02 14:47:12 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 28.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 3.
0(0:2)1  1(3:7)3  5(8:15)6

Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.10: 05-10-02
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc28
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
```

Text 2.5.30: console log for script tc28_postsim.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	-1.65e-04	-8.77e-04	0.1222	0.1456	0.0223	0.0252	1.12	1.26	1	
1	2	5.43e-04	3.91e-04	0.1323	0.1426	0.0244	0.0261	1.22	1.30	1	*
2	1	-2.09e-05	-5.02e-05	0.1421	0.1309	0.0233	0.0236	1.16	1.18	3	
2	2	1.37e-04	-3.98e-04	0.1512	0.1579	0.0242	0.0235	1.21	1.17	5	*
3	1	-3.47e-04	-4.03e-05	0.1210	0.1407	0.0208	0.0205	1.04	1.03	3	
3	2	7.96e-04	2.50e-04	0.1528	0.1498	0.0233	0.0234	1.17	1.17	7	*

Table 2.5.50: test case 28; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-2.05e-03	-3.16e-04	0.4628	0.4765	0.0895	0.0851	1.04	1.02	null
2	1	-2.47e-03	1.78e-03	0.4905	0.5712	0.0873	0.0914	0.95	0.94	null
3	1	-1.27e-03	-6.72e-04	0.4936	0.5172	0.0738	0.0754	0.95	0.96	null

Table 2.5.51: test case 28; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	1.05e-04	1.13e-04	0.0095	0.0150	1.45e-03	1.55e-03	2.4	2.6	0	
1	2	1.21e-04	-6.32e-05	0.0135	0.0105	1.55e-03	1.59e-03	2.6	2.7	0	*
2	1	5.88e-05	-3.69e-05	0.0121	0.0123	1.14e-03	1.08e-03	1.9	1.8	0	
2	2	-1.31e-05	1.18e-04	0.0109	0.0138	1.13e-03	1.13e-03	1.9	1.9	0	*
3	1	-2.81e-05	-1.52e-05	0.0086	0.0133	6.99e-04	8.60e-04	1.2	1.4	1	
3	2	6.49e-05	1.89e-05	0.0136	0.0123	9.34e-04	8.56e-04	1.6	1.4	0	*

Table 2.5.52: test case 28; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00100100  CCCSCR_2_3:   CCC3: Y=adj X=sec   CCC2: Y=pri X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000001  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11011100  XSTATUS: X-SE_CLK: 11011100
10110101  YSTATUS: Y-SE_CLK: 10110101
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.31: content of file mcb_reg28.txt.

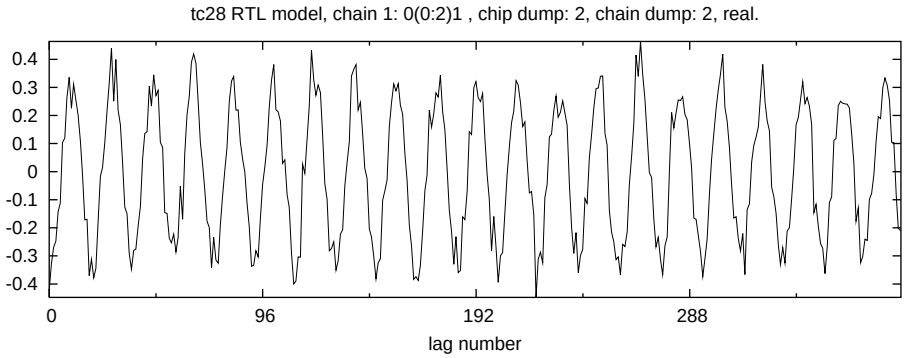

Illustration 2.5.313

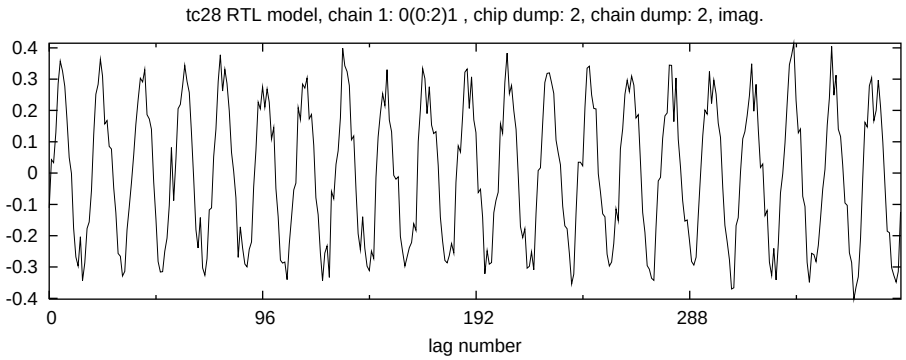

Illustration 2.5.314

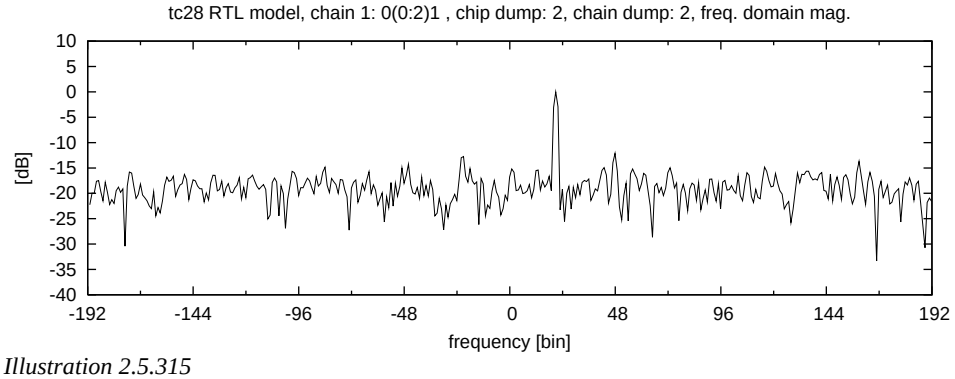

Illustration 2.5.315

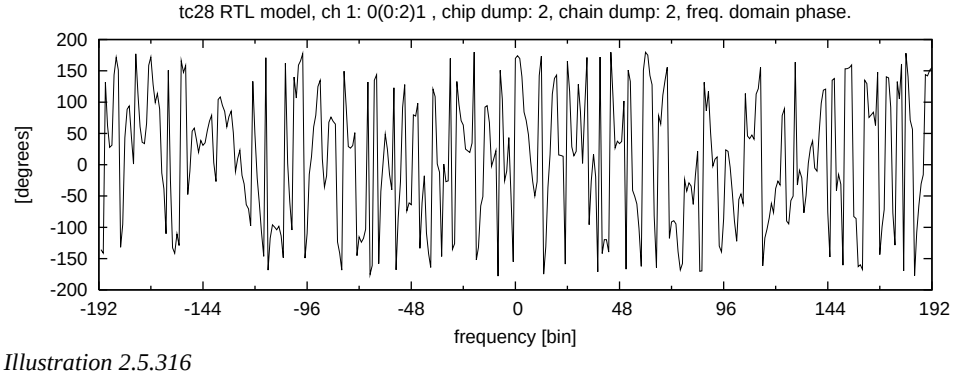

Illustration 2.5.316

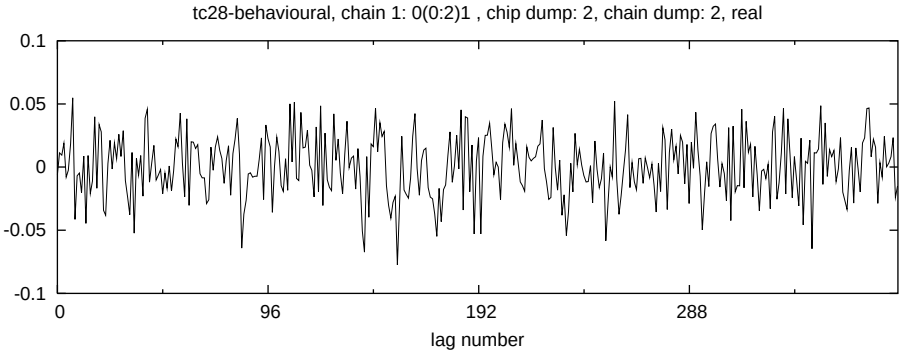

Illustration 2.5.317

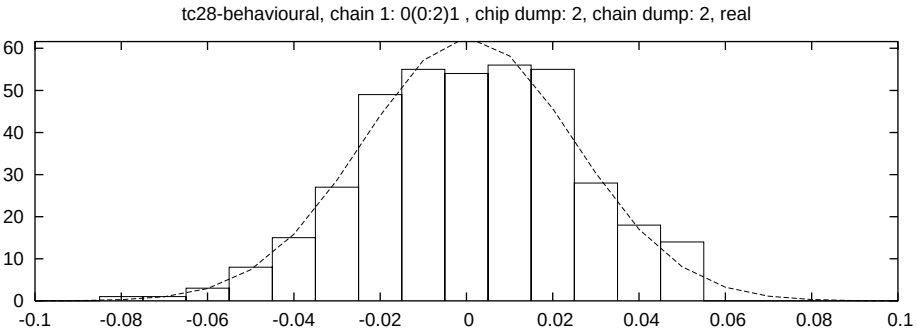

Illustration 2.5.318

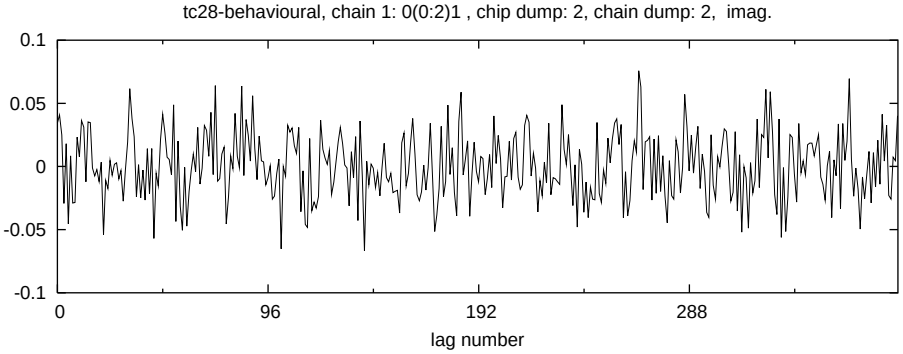

Illustration 2.5.319

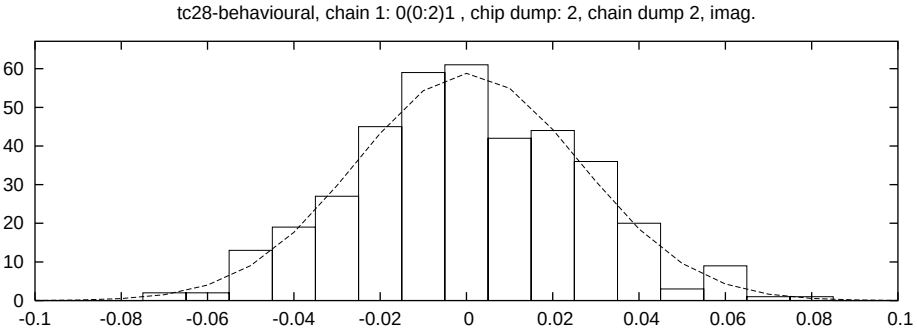

Illustration 2.5.320

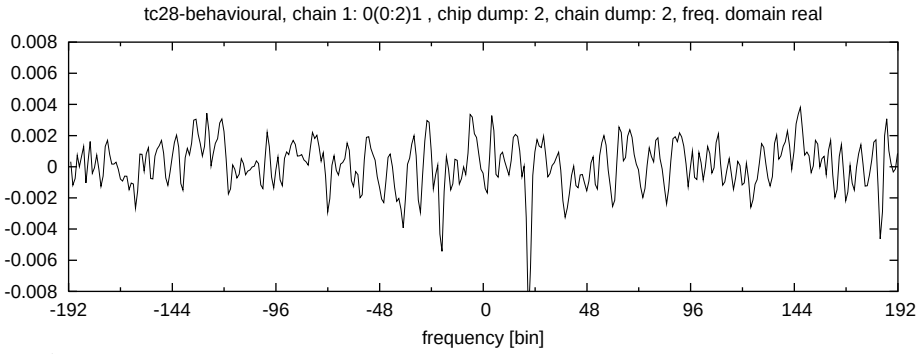

Illustration 2.5.321

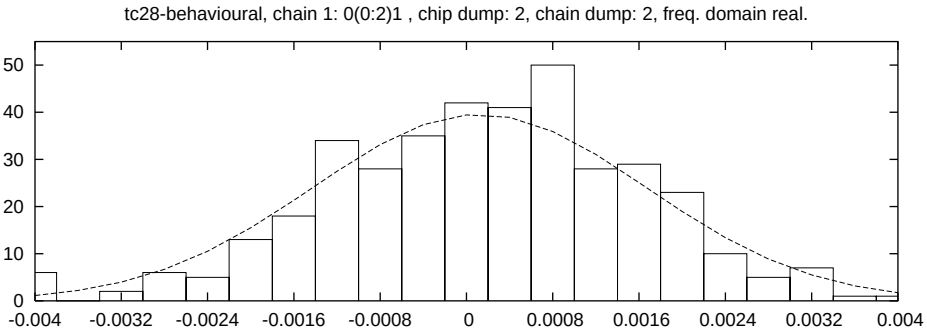

Illustration 2.5.322

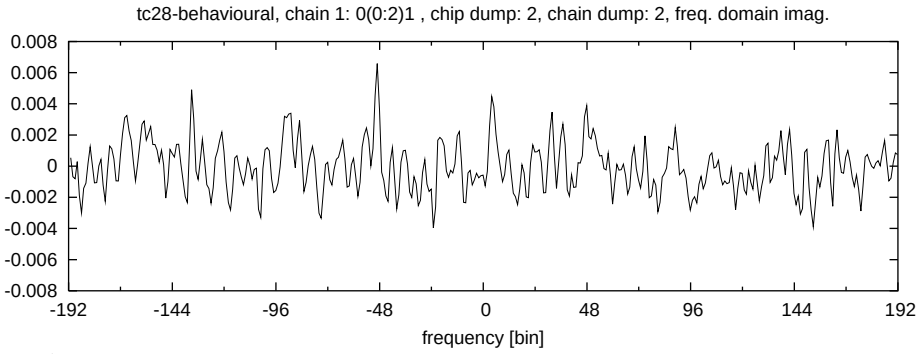

Illustration 2.5.323

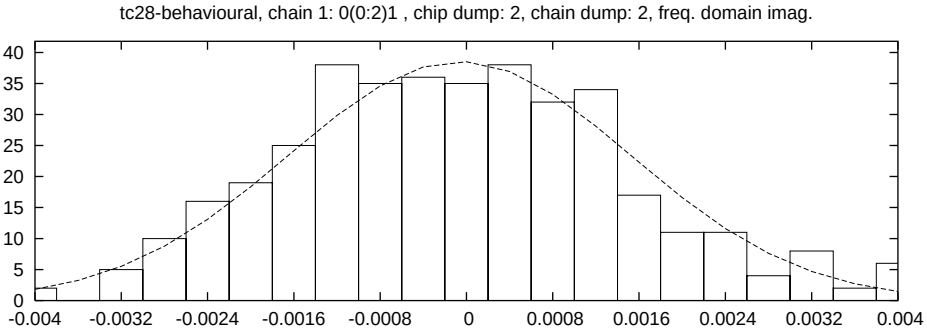

Illustration 2.5.324

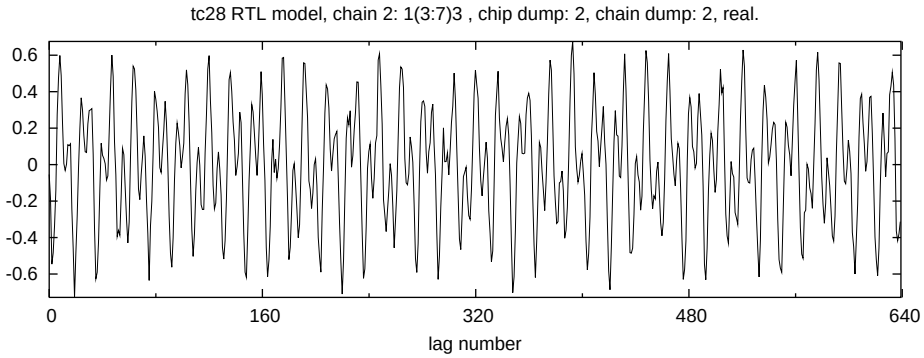

Illustration 2.5.325

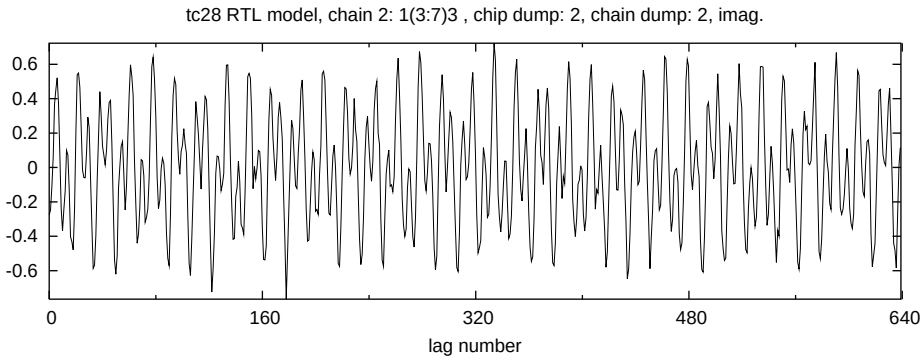

Illustration 2.5.326

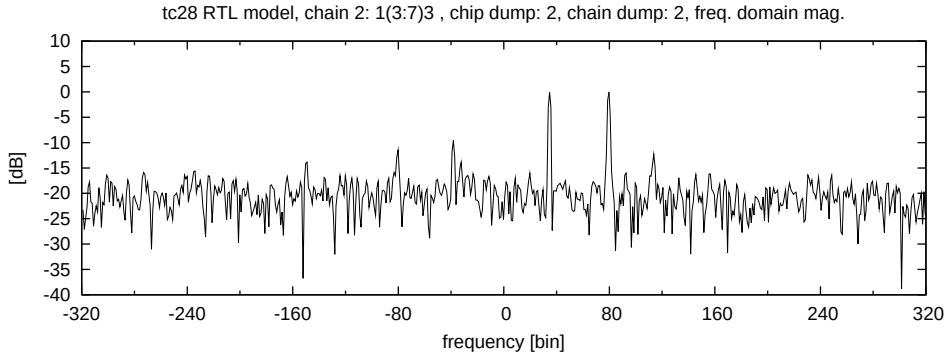

Illustration 2.5.327

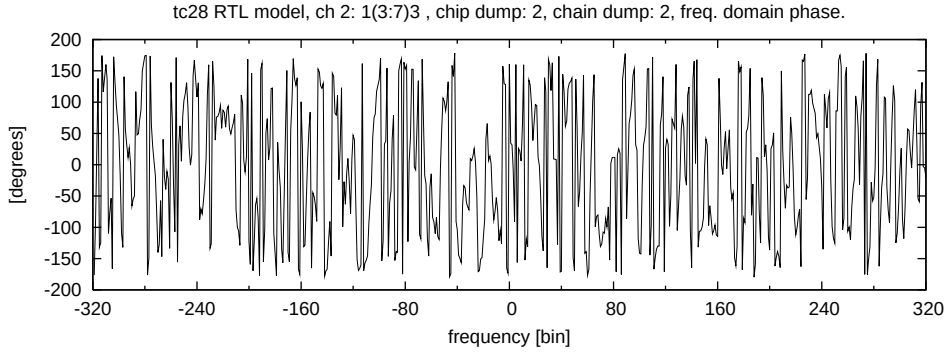

Illustration 2.5.328

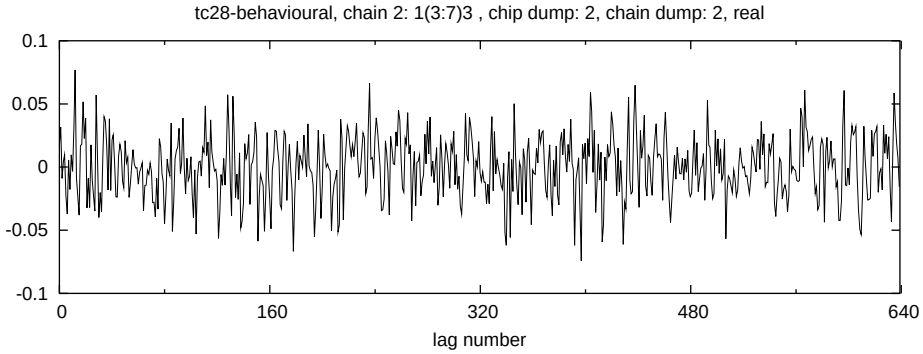

Illustration 2.5.329

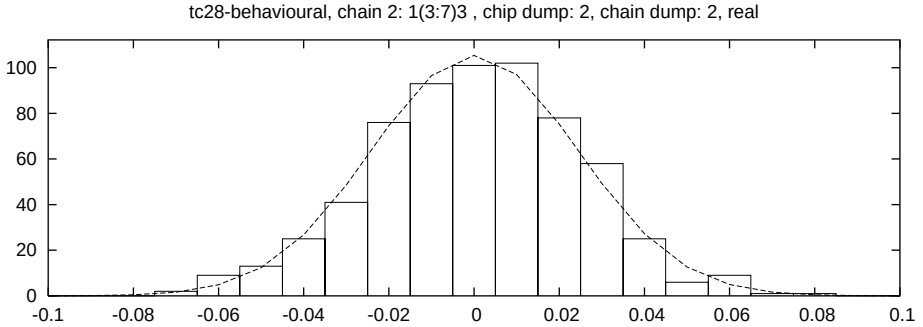

Illustration 2.5.330

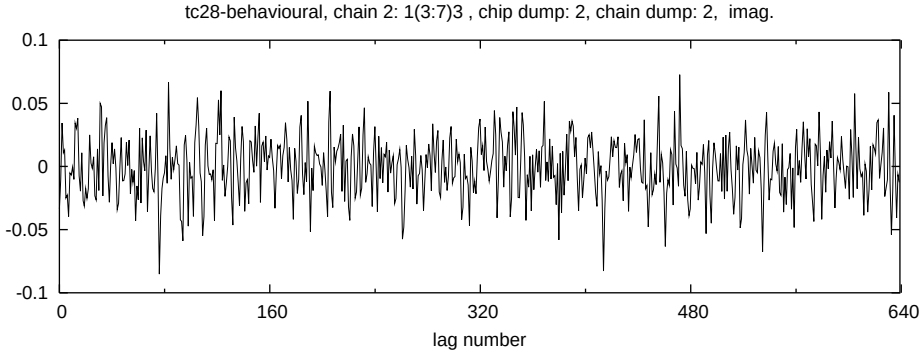

Illustration 2.5.331

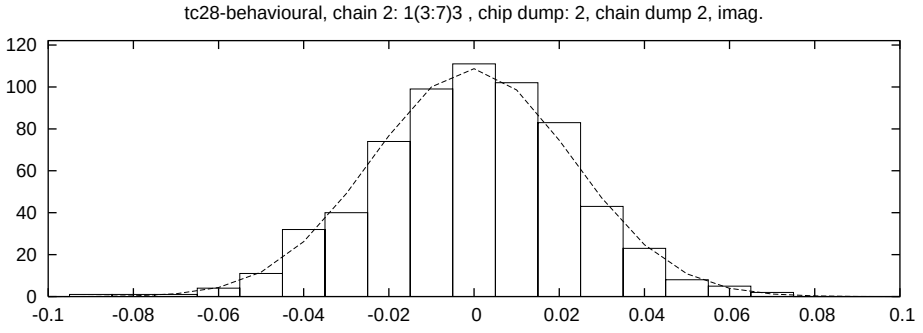

Illustration 2.5.332

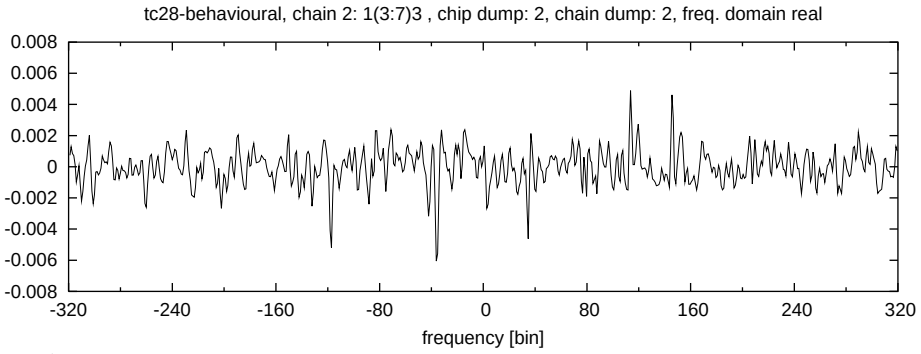

Illustration 2.5.333

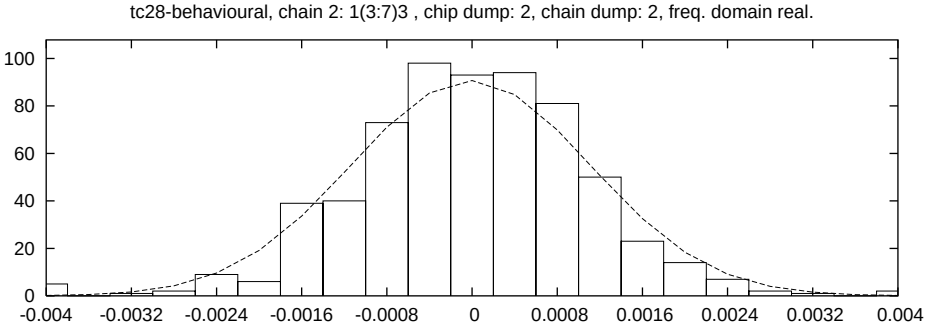

Illustration 2.5.334

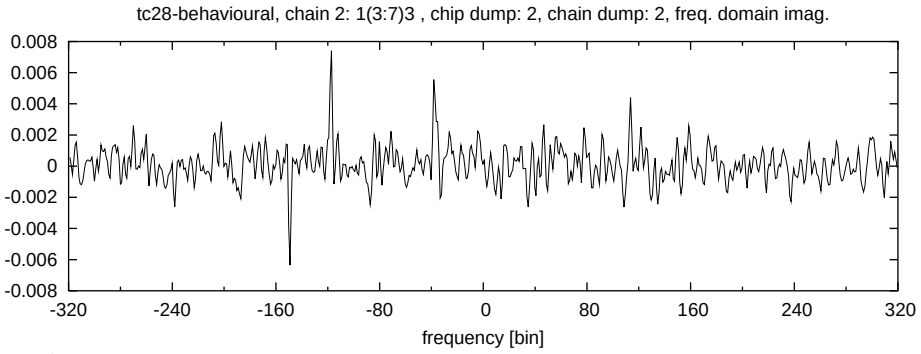

Illustration 2.5.335

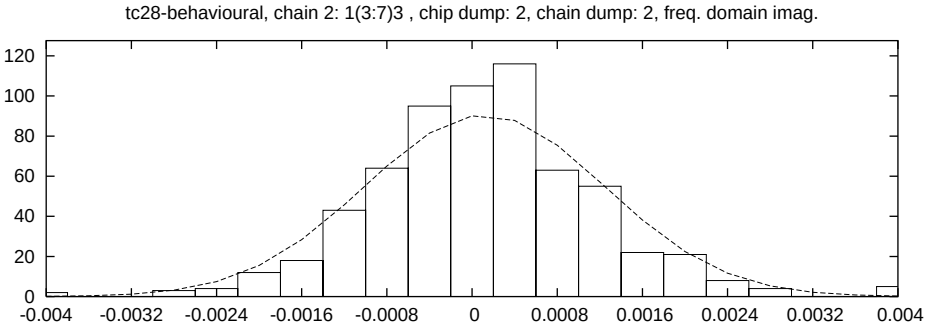

Illustration 2.5.336

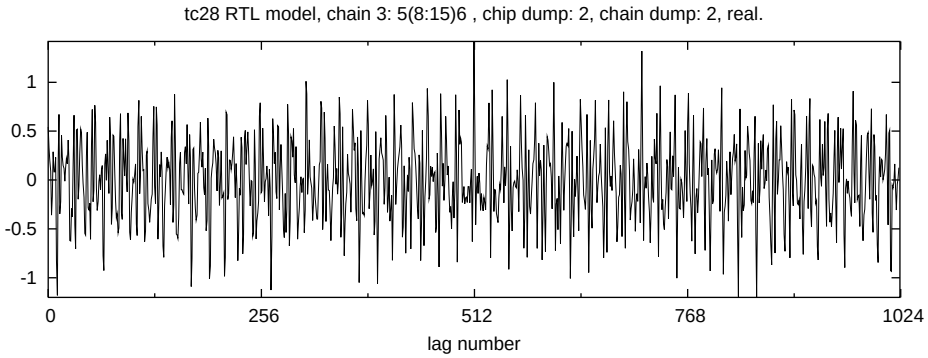

Illustration 2.5.337

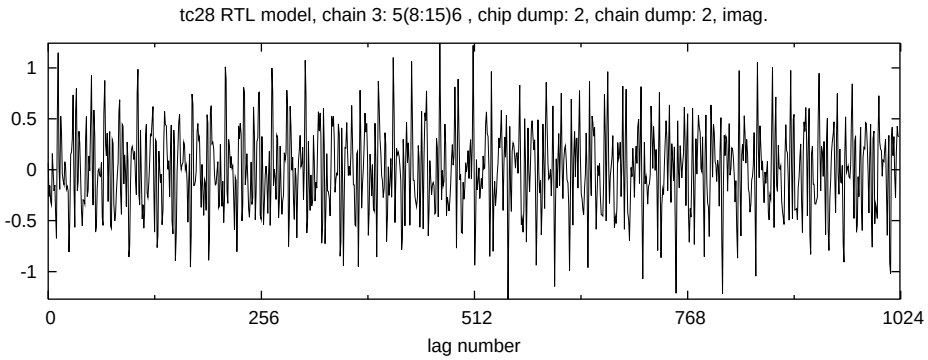

Illustration 2.5.338

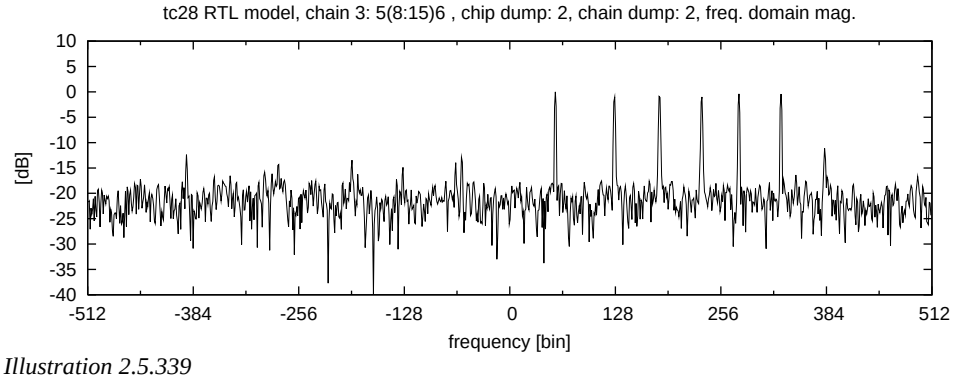

Illustration 2.5.339

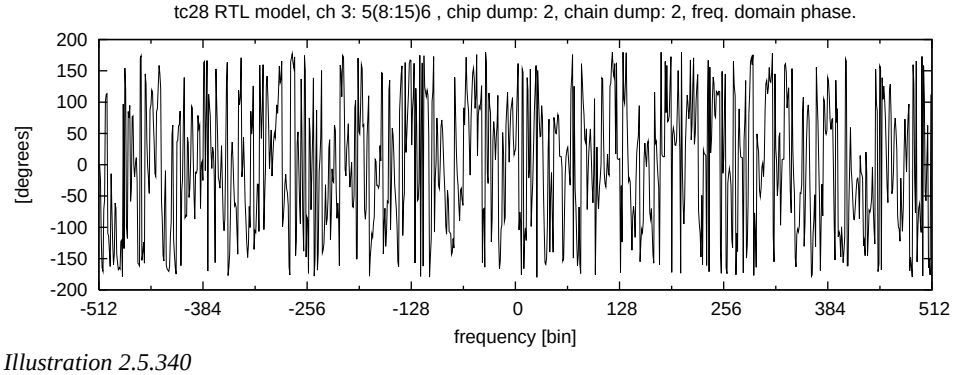

Illustration 2.5.340

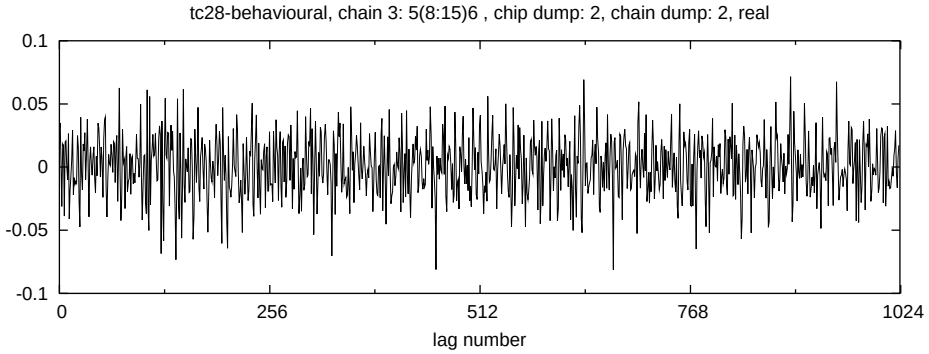

Illustration 2.5.341

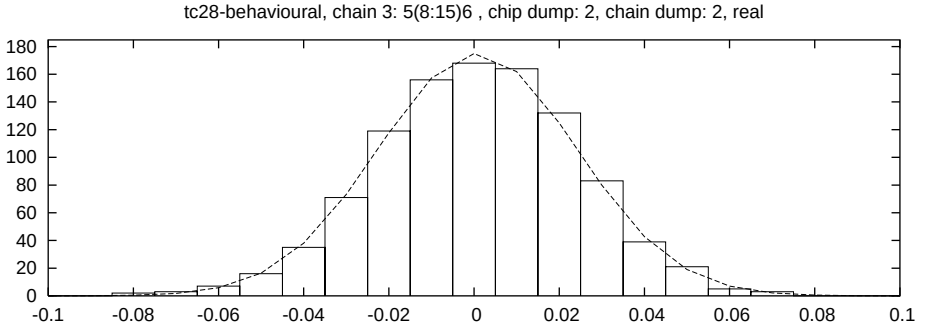

Illustration 2.5.342

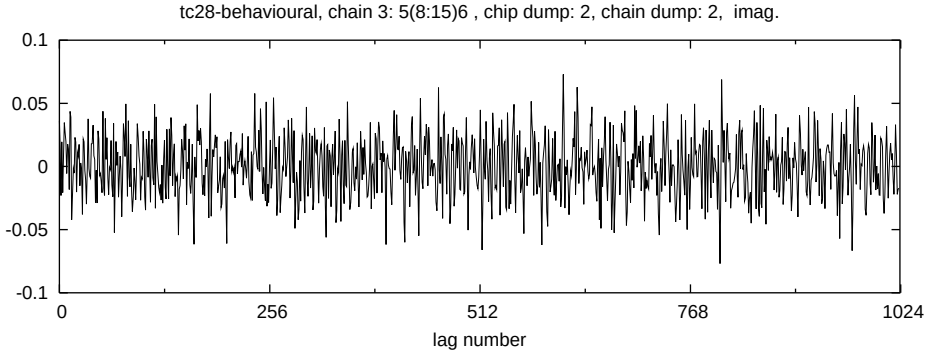

Illustration 2.5.343

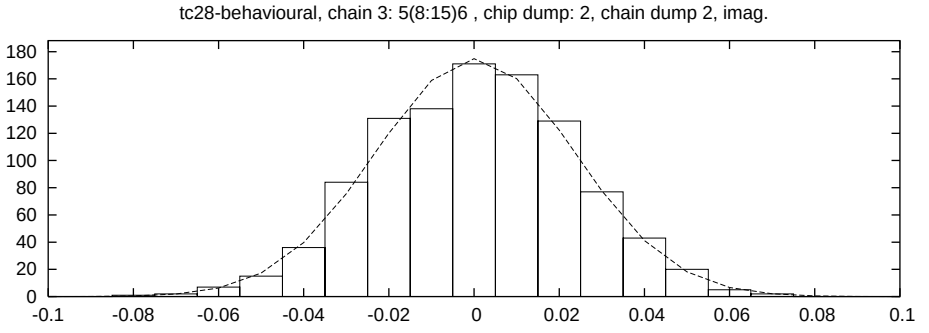

Illustration 2.5.344

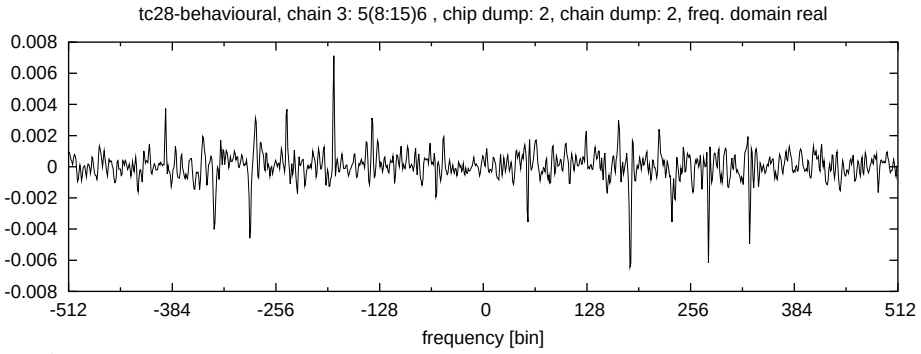

Illustration 2.5.345

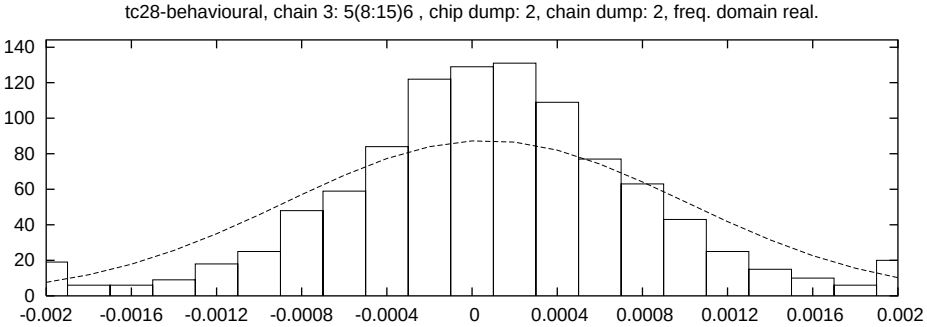

Illustration 2.5.346

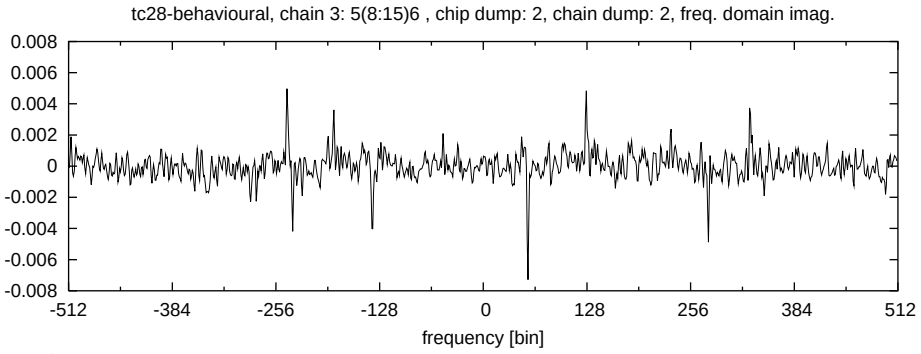

Illustration 2.5.347

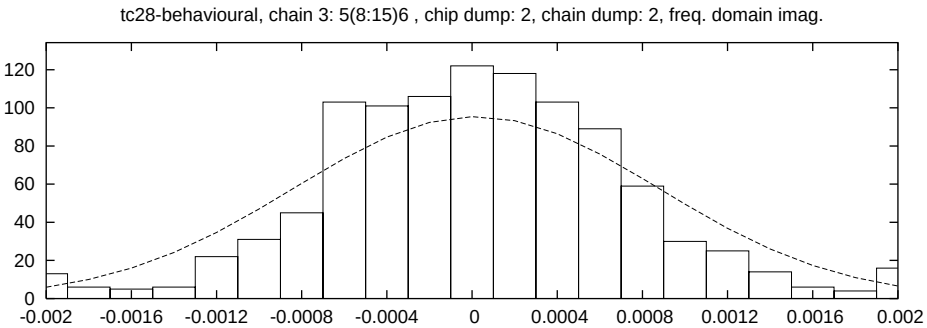

Illustration 2.5.348

2.5.14 test case 30

Synopsis: No advisories or warnings regarding captured data or register content were logged. With exception of the sample represented in the plots of Illustration 2.5.361 to Illustration 2.5.272 , tabulated values of standard deviation are comparable in magnitude to those of test case 25 with no frequency domain peaks exceeding 10 standard deviation reported. Hence, a full family of test case plots is not included. The lag coefficient difference plots of Illustration 2.5.365 to Illustration 2.5.368 show significant structure which manifests itself as a spectral line in the negative frequency domain (Illustration 2.5.369 to Illustration 2.5.372). Although a demonstration in section 2.7.8 suggests this behaviour is explainable by known differences between the RTL and behavioural models, confidence in a statistical model comparison is increased by a successful bit exact test involving all four polarizations of quad 0 (Text 2.5.33).

Notes: the dump immediately before that referred to above exhibits a lower spectral peak level and as seen in Illustration 2.5.349 to Illustration 2.5.360 .

```
**-------------------------------------------------------------------------***
tc30_postsim.m : V2.00 run time: Sun Sep 04 17:09:03 2005
Using postsim_lib: 1.40: 05-08-18

120 frames read for test case 30.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
1(0:0)1  1(1:1)0  0(2:2)1  0(3:3)0
3(4:4)3  3(5:5)2  2(6:6)3  2(7:7)2
5(8:8)5  5(9:9)4  4(10:10)5  4(11:11)4
7(12:12)7  7(13:13)6  6(14:14)7  6(15:15)6


Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc30
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***-------------------------------------------------------------------------***
```

Text 2.5.32 console output for script tc30_postsim.

```
***--------------------------------------------------------------------------***
tc30_bit_exact.m : V1.00 run time: Sun Dec 11 22:14:02 2005
Using postsim_lib: 1.40: 05-08-18

4 frames read for test case 30.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
1(0:0)1  1(1:1)0  0(2:2)1  0(3:3)0
3(4:4)3  3(5:5)2  2(6:6)3  2(7:7)2
5(8:8)5  5(9:9)4  4(10:10)5  4(11:11)4
7(12:12)7  7(13:13)6  6(14:14)7  6(15:15)6


Initial (single dump) timestamp (word 0): 222333444
Cannot open file CC3x3_test30-S_128lagsD0.dat.
Cannot open file CC3x2_test30-S_128lagsD0.dat.
Cannot open file CC2x3_test30-S_128lagsD0.dat.
Cannot open file CC2x2_test30-S_128lagsD0.dat.
Cannot open file CC5x5_test30-S_128lagsD0.dat.
Cannot open file CC5x4_test30-S_128lagsD0.dat.
Cannot open file CC4x5_test30-S_128lagsD0.dat.
Cannot open file CC4x4_test30-S_128lagsD0.dat.
Cannot open file CC7x7_test30-S_128lagsD0.dat.
Cannot open file CC7x6_test30-S_128lagsD0.dat.
Cannot open file CC6x7_test30-S_128lagsD0.dat.
Cannot open file CC6x6_test30-S_128lagsD0.dat.
***--------------------------------------------------------------------------***
octave:56> tc30_chain_similarity_check

Behavioural and functional models agree on a bit exact basis
        for all four polarizations.
```

Text 2.5.33: console output for scripts tc30_bit_exact, tc30_chain_similarity_check. The simulation was halted after the first dump and a bit exact comparison to the behavioural model was performed as described in the CMAM_correlation section of [TVP-02]. In this case, test vector set DS16 was applied to the RTL model. All four polarizations using cells of quad 0 are represented by the behavioural model output, data for the remaining quads are absent.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	-2.16e-03	-4.03e-04	0.1175	0.1033	0.0263	0.0222	1.31	1.11	0	
1	2	-5.33e-04	-1.24e-04	0.1363	0.1157	0.0263	0.0229	1.32	1.14	0	
1	3	8.12e-04	-9.62e-05	0.1493	0.1264	0.0279	0.0284	1.39	1.42	0	
1	4	-9.69e-04	2.87e-03	0.0929	0.1218	0.0199	0.0232	1.00	1.16	0	
1	5	4.96e-04	2.69e-03	0.1549	0.1368	0.0249	0.0247	1.24	1.24	1	
1	6	-2.29e-03	1.47e-03	0.1459	0.1352	0.0324	0.0271	1.62	1.36	0	
1	7	9.33e-04	-2.66e-03	0.1080	0.1351	0.0217	0.0248	1.08	1.24	0	
1	8	-1.02e-03	9.49e-04	0.1335	0.1325	0.0217	0.0266	1.08	1.33	1	
1	9	3.07e-03	-2.08e-03	0.1157	0.1236	0.0216	0.0243	1.08	1.22	0	
1	10	2.41e-04	-1.29e-03	0.1265	0.1279	0.0272	0.0245	1.36	1.23	0	
1	11	1.71e-03	8.18e-04	0.1100	0.0967	0.0225	0.0215	1.13	1.08	0	
1	12	-1.34e-03	2.66e-03	0.1337	0.1365	0.0274	0.0296	1.37	1.48	0	
1	13	1.12e-03	-3.31e-04	0.1258	0.1153	0.0223	0.0216	1.12	1.08	1	
1	14	-1.85e-03	3.13e-04	0.1354	0.1126	0.0238	0.0248	1.19	1.24	1	
1	15	3.40e-03	8.89e-04	0.1388	0.1133	0.0295	0.0220	1.47	1.10	0	
1	16	-7.19e-03	-1.27e-03	0.1286	0.1084	0.0245	0.0197	1.22	0.98	2	
2	1	1.27e-03	-1.92e-03	0.1789	0.1255	0.0357	0.0289	1.79	1.45	0	
2	2	7.21e-04	3.11e-04	0.1591	0.1100	0.0288	0.0233	1.44	1.16	0	
2	3	-9.75e-04	2.01e-03	0.1307	0.1592	0.0290	0.0315	1.45	1.58	0	
2	4	2.71e-03	-1.83e-03	0.1473	0.1681	0.0327	0.0338	1.63	1.69	0	
2	5	1.84e-03	-1.73e-03	0.1462	0.1805	0.0342	0.0289	1.71	1.44	2	
2	6	-2.31e-03	3.00e-03	0.1138	0.1286	0.0232	0.0267	1.16	1.33	0	*
2	7	-1.16e-03	-5.16e-05	0.1635	0.1564	0.0351	0.0296	1.76	1.48	1	*
2	8	-2.24e-03	1.35e-03	0.1581	0.1322	0.0310	0.0304	1.55	1.52	0	
2	9	1.35e-03	-1.42e-03	0.1528	0.1374	0.0266	0.0252	1.33	1.26	2	
2	10	-1.16e-03	1.93e-03	0.1383	0.1145	0.0277	0.0236	1.39	1.18	0	
2	11	2.49e-03	1.02e-03	0.1614	0.1972	0.0343	0.0370	1.71	1.85	0	
2	12	2.48e-04	2.03e-04	0.1643	0.1594	0.0321	0.0321	1.61	1.61	0	
2	13	6.83e-04	1.15e-03	0.1333	0.1226	0.0250	0.0217	1.25	1.09	1	
2	14	-2.45e-03	-4.36e-03	0.1044	0.1512	0.0218	0.0295	1.09	1.47	0	
2	15	2.46e-03	2.11e-03	0.1223	0.1573	0.0241	0.0343	1.21	1.71	0	
2	16	-2.94e-03	-8.99e-04	0.1320	0.1285	0.0262	0.0317	1.31	1.59	0	
3	1	8.30e-04	-1.14e-03	0.1175	0.1376	0.0237	0.0283	1.18	1.42	0	
3	2	-1.98e-03	-1.89e-03	0.1317	0.1564	0.0248	0.0248	1.24	1.24	2	
3	3	1.68e-03	2.99e-03	0.1062	0.1306	0.0239	0.0294	1.20	1.47	0	
3	4	-7.98e-04	-3.57e-04	0.1292	0.1287	0.0267	0.0239	1.33	1.19	0	
3	5	-4.43e-04	-6.55e-04	0.1483	0.1343	0.0269	0.0239	1.35	1.19	0	
3	6	1.93e-04	-8.82e-04	0.1323	0.1401	0.0295	0.0255	1.48	1.28	0	
3	7	-9.32e-04	1.20e-03	0.1420	0.1319	0.0233	0.0268	1.16	1.34	1	
3	8	2.51e-03	2.80e-03	0.1409	0.1217	0.0264	0.0235	1.32	1.17	0	
3	9	4.44e-05	-5.55e-04	0.1279	0.1218	0.0223	0.0222	1.12	1.11	2	
3	10	6.22e-04	8.79e-05	0.1235	0.1407	0.0260	0.0303	1.30	1.51	1	
3	11	-8.52e-04	-2.27e-04	0.0900	0.1610	0.0197	0.0285	0.98	1.42	1	
3	12	9.09e-04	3.72e-05	0.1027	0.1304	0.0199	0.0281	0.99	1.41	0	
3	13	1.17e-03	6.78e-04	0.1545	0.1226	0.0257	0.0241	1.29	1.21	2	

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
3	14	-1.41e-03	-1.52e-03	0.1098	0.1114	0.0229	0.0246	1.14	1.23	0	
3	15	7.51e-04	-1.37e-03	0.1192	0.1440	0.0244	0.0226	1.22	1.13	2	
3	16	1.59e-03	-3.52e-06	0.1315	0.1487	0.0222	0.0232	1.11	1.16	3	
4	1	4.39e-03	-1.38e-03	0.1669	0.0985	0.0272	0.0223	1.36	1.11	1	
4	2	2.82e-05	1.57e-03	0.1216	0.1286	0.0253	0.0231	1.26	1.16	0	
4	3	-5.80e-05	2.16e-05	0.0933	0.1054	0.0216	0.0248	1.08	1.24	0	
4	4	-6.35e-04	-1.02e-03	0.1445	0.1235	0.0280	0.0258	1.40	1.29	0	
4	5	-4.17e-04	3.76e-04	0.1493	0.1662	0.0269	0.0290	1.34	1.45	2	
4	6	-4.62e-04	-2.18e-03	0.0994	0.1257	0.0201	0.0235	1.01	1.17	1	
4	7	2.21e-03	-4.38e-04	0.1047	0.1366	0.0201	0.0254	1.00	1.27	2	
4	8	-1.75e-03	-2.56e-05	0.1377	0.1342	0.0242	0.0258	1.21	1.29	0	
4	9	-1.23e-04	1.71e-03	0.1613	0.1272	0.0259	0.0255	1.29	1.28	1	
4	10	-8.92e-04	-8.34e-04	0.1584	0.1354	0.0272	0.0240	1.36	1.20	2	
4	11	8.67e-04	-8.57e-04	0.1708	0.1524	0.0332	0.0305	1.66	1.52	0	
4	12	1.07e-03	2.82e-04	0.1574	0.1391	0.0267	0.0234	1.33	1.17	2	
4	13	8.15e-04	-5.67e-05	0.1450	0.1271	0.0277	0.0273	1.38	1.37	0	
4	14	1.74e-03	-3.44e-03	0.1370	0.1232	0.0258	0.0261	1.29	1.30	0	
4	15	1.89e-04	1.30e-03	0.1848	0.1236	0.0286	0.0235	1.43	1.18	1	
4	16	-1.23e-03	1.84e-03	0.1344	0.1363	0.0253	0.0267	1.26	1.33	1	
5	2	-1.56e-03	-3.83e-04	0.0605	0.0832	0.0112	0.0138	0.79	0.98	1	
5	4	-7.99e-04	-2.39e-04	0.0646	0.0535	0.0117	0.0122	0.82	0.86	0	
5	6	3.85e-04	8.86e-04	0.0567	0.0709	0.0121	0.0150	0.85	1.06	0	
5	8	1.25e-03	-1.25e-03	0.0592	0.0665	0.0124	0.0124	0.88	0.88	1	
5	10	1.43e-03	9.67e-04	0.0629	0.0719	0.0121	0.0132	0.86	0.93	0	
5	12	-4.30e-04	-1.90e-03	0.0689	0.0677	0.0124	0.0126	0.88	0.89	2	
5	14	1.59e-04	8.97e-04	0.0652	0.0509	0.0131	0.0113	0.93	0.80	0	
5	16	-3.04e-05	-4.44e-04	0.0627	0.0635	0.0130	0.0124	0.92	0.88	0	
6	2	4.03e-04	9.36e-04	0.0517	0.0617	0.0114	0.0123	0.80	0.87	0	
6	4	-5.03e-04	-1.25e-03	0.0582	0.0556	0.0121	0.0117	0.86	0.83	0	
6	6	-5.59e-04	7.43e-04	0.0878	0.0662	0.0172	0.0120	1.22	0.85	0	
6	8	2.12e-04	3.29e-04	0.0622	0.0662	0.0130	0.0126	0.92	0.89	0	
6	10	-3.93e-04	-1.39e-03	0.0659	0.0683	0.0130	0.0133	0.92	0.94	0	
6	12	-8.78e-05	3.35e-04	0.0669	0.0650	0.0116	0.0130	0.82	0.92	1	
6	14	-8.19e-04	3.42e-04	0.0691	0.0697	0.0126	0.0124	0.89	0.88	1	
6	16	8.18e-04	-7.66e-05	0.0562	0.0720	0.0126	0.0121	0.89	0.86	1	
7	2	4.33e-04	-1.49e-03	0.0745	0.0758	0.0137	0.0133	0.97	0.94	1	
7	4	6.40e-04	-3.07e-04	0.0643	0.0550	0.0123	0.0114	0.87	0.81	1	
7	6	2.22e-04	6.41e-04	0.0625	0.0715	0.0132	0.0129	0.94	0.91	1	
7	8	1.39e-03	-6.40e-04	0.0635	0.0614	0.0121	0.0123	0.86	0.87	0	
7	10	1.46e-03	6.81e-04	0.0570	0.0551	0.0125	0.0116	0.88	0.82	0	
7	12	-1.39e-03	-1.88e-03	0.0530	0.0586	0.0111	0.0124	0.79	0.88	0	
7	14	-7.34e-05	-1.55e-04	0.0673	0.0652	0.0133	0.0142	0.94	1.00	0	
7	16	8.64e-04	-2.76e-04	0.0666	0.0703	0.0115	0.0132	0.81	0.93	2	
8	2	-6.33e-04	1.26e-03	0.0651	0.0716	0.0119	0.0125	0.84	0.88	2	
8	4	-1.24e-03	-4.44e-04	0.0604	0.0698	0.0122	0.0143	0.86	1.01	0	

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
8	6	2.56e-03	8.83e-04	0.0637	0.0661	0.0139	0.0143	0.98	1.01	0	
8	8	-1.44e-04	3.18e-04	0.0713	0.0666	0.0133	0.0115	0.94	0.81	2	
8	10	-5.44e-04	-1.18e-03	0.0660	0.0506	0.0130	0.0116	0.92	0.82	0	
8	12	-1.33e-03	1.07e-03	0.0561	0.0649	0.0111	0.0124	0.79	0.87	0	
8	14	-1.01e-03	2.41e-04	0.0593	0.0811	0.0133	0.0171	0.94	1.21	0	
8	16	-6.26e-04	-1.41e-03	0.0535	0.0611	0.0114	0.0131	0.81	0.93	0	
9	4	2.74e-04	-2.59e-04	0.0274	0.0368	0.0059	0.0076	0.59	0.76	0	
9	8	6.55e-04	-2.85e-04	0.0554	0.0521	0.0098	0.0098	0.98	0.98	1	
9	12	-8.37e-04	8.26e-05	0.0410	0.0402	0.0071	0.0080	0.71	0.80	1	
9	16	2.35e-04	-2.29e-04	0.0360	0.0456	0.0069	0.0087	0.69	0.87	0	
10	4	-3.38e-04	-9.40e-05	0.0335	0.0407	0.0072	0.0076	0.72	0.76	0	
10	8	8.13e-04	-1.98e-05	0.0462	0.0538	0.0092	0.0103	0.92	1.03	0	
10	12	-6.24e-04	-5.21e-04	0.0430	0.0415	0.0094	0.0083	0.94	0.83	0	
10	16	5.05e-04	2.08e-04	0.0408	0.0416	0.0083	0.0077	0.83	0.77	0	
11	4	3.40e-04	-1.70e-04	0.0396	0.0407	0.0076	0.0077	0.76	0.77	1	
11	8	4.33e-04	3.96e-04	0.0447	0.0475	0.0098	0.0086	0.98	0.86	1	
11	12	6.00e-04	3.42e-04	0.0315	0.0392	0.0064	0.0068	0.64	0.68	1	
11	16	9.53e-04	3.22e-04	0.0392	0.0358	0.0081	0.0077	0.81	0.77	0	
12	4	2.47e-04	-2.40e-05	0.0424	0.0353	0.0091	0.0072	0.91	0.72	0	
12	8	-1.36e-04	-1.54e-04	0.0333	0.0382	0.0064	0.0090	0.64	0.90	0	
12	12	2.95e-04	3.42e-04	0.0393	0.0334	0.0067	0.0072	0.67	0.72	0	
12	16	-1.71e-04	-2.19e-04	0.0352	0.0404	0.0069	0.0078	0.69	0.78	0	
13	8	2.51e-04	5.52e-04	0.0336	0.0326	0.0059	0.0060	0.84	0.85	1	
13	16	2.68e-04	4.17e-04	0.0327	0.0363	0.0057	0.0067	0.80	0.95	1	
14	8	-1.50e-04	-6.05e-05	0.0333	0.0291	0.0059	0.0060	0.83	0.85	1	
14	16	4.70e-04	2.42e-05	0.0292	0.0280	0.0056	0.0057	0.80	0.81	0	
15	8	3.60e-05	3.98e-04	0.0259	0.0310	0.0054	0.0057	0.76	0.80	1	
15	16	6.55e-05	-3.99e-05	0.0309	0.0339	0.0061	0.0061	0.86	0.87	1	
16	8	3.53e-04	-1.42e-04	0.0288	0.0245	0.0059	0.0051	0.84	0.72	0	
16	16	-7.52e-05	1.84e-04	0.0285	0.0333	0.0059	0.0060	0.84	0.85	1	

Table 2.5.53 test case 30; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[\,f-f\,]}$	$\overline{\Im[\,f-f\,]}$	*range* $\Re[\,f-f\,]$	*range* $\Im[\,f-f\,]$	$\sigma_{\Re[\,f-f\,]}$	$\sigma_{\Im[\,f-f\,]}$	$\frac{\sigma_{\Re[\,f-f\,]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[\,f-f\,]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-1.59e-03	3.57e-03	0.4223	0.4910	0.0796	0.0817	1.09	0.96	null
1	2	6.60e-03	-1.32e-03	0.5023	0.5286	0.0903	0.0926	1.14	1.08	null
1	3	7.78e-04	7.98e-03	0.4938	0.3942	0.0886	0.0873	1.20	1.05	null
1	4	-8.27e-03	-3.50e-03	0.4072	0.4949	0.0688	0.0841	1.01	1.07	null
1	5	-1.65e-03	-3.67e-03	0.4190	0.3294	0.0879	0.0710	1.11	0.95	null
1	6	4.63e-03	-6.36e-04	0.4274	0.4381	0.0724	0.0820	0.95	0.89	null
1	7	1.70e-03	7.14e-03	0.5545	0.5194	0.0899	0.0999	1.01	1.03	null
1	8	-4.69e-03	-1.03e-04	0.3823	0.3960	0.0774	0.0767	0.93	1.03	null
1	9	4.73e-03	-7.03e-03	0.5437	0.3495	0.1026	0.0721	1.06	0.97	null
1	10	-1.67e-03	5.08e-03	0.3525	0.3283	0.0820	0.0704	0.95	1.09	null
1	11	-2.55e-03	-6.67e-03	0.3643	0.4170	0.0744	0.0858	1.01	1.00	null
1	12	1.44e-03	2.04e-03	0.3955	0.4179	0.0774	0.0731	0.99	0.91	null
1	13	-7.50e-04	1.30e-03	0.5021	0.4663	0.0905	0.0924	0.95	1.06	null
1	14	4.62e-03	7.36e-03	0.4555	0.4184	0.0807	0.0914	0.97	1.07	null
1	15	-9.78e-03	-1.04e-02	0.4117	0.3867	0.0798	0.0803	1.00	1.05	null
2	1	1.50e-03	-1.55e-03	0.3892	0.4536	0.0827	0.0848	1.07	0.93	null
2	2	-6.68e-04	-6.30e-03	0.3846	0.4422	0.0803	0.0837	1.04	1.03	null
2	3	3.40e-03	9.62e-03	0.3868	0.4606	0.0697	0.0911	0.89	1.33	null
2	4	-5.30e-03	-8.26e-03	0.5273	0.4471	0.0839	0.1029	0.97	1.25	null
2	5	1.28e-02	8.82e-03	0.3674	0.4108	0.0726	0.0865	0.86	1.11	null
2	6	-1.24e-02	-7.88e-03	0.6324	0.4798	0.1141	0.1006	1.26	1.22	null
2	7	-2.93e-03	8.20e-03	0.5377	0.3724	0.1058	0.0824	1.32	1.06	null
2	8	1.17e-02	-3.71e-04	0.3453	0.3888	0.0715	0.0805	1.03	1.17	null
2	9	-4.92e-03	-7.37e-03	0.3939	0.4137	0.0798	0.0827	1.15	1.03	null
2	10	-1.68e-03	6.17e-03	0.4008	0.4374	0.0815	0.0881	0.99	1.04	null
2	11	1.13e-03	6.36e-03	0.4140	0.4784	0.0873	0.0877	0.94	0.84	null
2	12	3.68e-03	-7.77e-03	0.4329	0.5237	0.0777	0.0904	0.95	0.91	null
2	13	-3.63e-03	2.32e-03	0.4146	0.4663	0.0853	0.0909	0.93	1.05	null
2	14	1.70e-04	-1.70e-03	0.4494	0.3436	0.0858	0.0691	1.05	0.95	null
2	15	5.42e-03	-5.59e-03	0.3770	0.4460	0.0788	0.0752	1.00	0.99	null
3	1	-9.73e-03	6.99e-04	0.4305	0.4102	0.0879	0.0868	0.98	0.94	null
3	2	1.08e-02	-1.18e-03	0.4137	0.3648	0.0788	0.0749	0.97	0.96	null
3	3	-2.74e-03	2.74e-03	0.3416	0.4733	0.0752	0.0887	0.97	0.89	null
3	4	-1.11e-03	2.32e-03	0.4244	0.4059	0.0849	0.0844	1.04	0.94	null
3	5	6.49e-03	-5.63e-03	0.4008	0.4982	0.0848	0.0919	1.12	1.04	null
3	6	-6.20e-03	1.29e-03	0.4511	0.4387	0.0838	0.0743	1.00	1.03	null
3	7	8.91e-03	4.17e-03	0.4291	0.4345	0.0877	0.0775	1.06	1.02	null
3	8	-1.72e-03	-9.32e-04	0.4283	0.4527	0.0899	0.0827	1.12	1.03	null
3	9	-7.91e-03	-4.04e-03	0.4967	0.4500	0.0881	0.0862	1.08	1.04	63
3	10	8.44e-03	-7.70e-04	0.3865	0.4552	0.0835	0.0970	1.02	1.12	null
3	11	-3.76e-03	3.02e-04	0.3967	0.3343	0.0813	0.0804	1.01	0.97	null
3	12	2.92e-03	4.17e-03	0.4083	0.5444	0.0730	0.0731	1.05	0.97	null
3	13	-1.47e-03	-1.19e-02	0.4281	0.4937	0.0808	0.0968	1.08	1.03	null
3	14	-5.49e-03	6.80e-03	0.3963	0.4515	0.0801	0.0896	1.04	0.96	null
3	15	9.81e-03	2.56e-03	0.3973	0.3578	0.0845	0.0743	1.04	0.98	null

chain	*dump*	$\overline{\Re[\,f-f\,]}$	$\overline{\Im[\,f-f\,]}$	*range* $\Re[\,f-f\,]$	*range* $\Im[\,f-f\,]$	$\sigma_{\Re[\,f-f\,]}$	$\sigma_{\Im[\,f-f\,]}$	$\frac{\sigma_{\Re[\,f-f\,]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[\,f-f\,]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
4	1	5.19e-04	3.57e-03	0.4313	0.4946	0.0931	0.0994	1.17	1.06	64
4	2	1.58e-03	9.53e-03	0.3789	0.4052	0.0835	0.0801	0.99	1.03	null
4	3	-1.59e-02	-1.17e-02	0.4930	0.4227	0.0911	0.0847	1.07	0.99	null
4	4	2.07e-02	6.63e-03	0.4573	0.4704	0.0914	0.0794	1.05	0.91	null
4	5	-1.02e-02	-4.67e-04	0.5579	0.4584	0.1090	0.0829	1.04	1.03	null
4	6	7.89e-03	-3.05e-03	0.4708	0.4614	0.0783	0.0833	1.06	1.05	null
4	7	-1.14e-02	-2.07e-03	0.4183	0.4523	0.0770	0.0941	1.03	1.11	null
4	8	-1.76e-03	1.97e-03	0.5274	0.4591	0.0913	0.0831	1.12	1.04	null
4	9	-5.92e-04	1.59e-03	0.5007	0.3197	0.1033	0.0779	1.08	1.03	null
4	10	1.20e-03	-2.76e-04	0.4534	0.4652	0.0980	0.0910	1.12	1.12	21
4	11	3.64e-03	-1.07e-03	0.4570	0.4884	0.0903	0.0796	0.89	0.95	null
4	12	-9.87e-04	2.09e-03	0.5264	0.4864	0.0935	0.0940	1.04	1.06	null
4	13	6.83e-04	-4.89e-03	0.5220	0.4468	0.0894	0.0827	1.00	1.06	null
4	14	2.80e-03	6.49e-03	0.5214	0.4122	0.1068	0.0856	1.12	0.99	null
4	15	-1.06e-02	-1.23e-02	0.4198	0.4304	0.0873	0.0876	0.92	1.03	null
5	2	2.18e-03	-1.42e-03	0.3498	0.3016	0.0562	0.0540	1.01	0.97	null
5	4	3.75e-03	1.66e-03	0.3063	0.2407	0.0589	0.0504	1.01	0.93	null
5	6	6.23e-04	-3.88e-04	0.3123	0.2257	0.0573	0.0467	1.08	0.96	null
5	8	-4.00e-03	5.58e-04	0.2863	0.2630	0.0540	0.0486	0.99	1.01	null
5	10	-3.81e-03	-6.06e-04	0.3657	0.3375	0.0694	0.0590	0.95	0.97	null
5	12	3.61e-03	-1.84e-03	0.2703	0.3581	0.0545	0.0747	0.98	1.01	null
5	14	-1.58e-03	3.76e-03	0.2666	0.3028	0.0582	0.0645	1.04	1.02	null
6	2	1.97e-03	-3.99e-03	0.2885	0.3165	0.0605	0.0703	1.09	1.02	null
6	4	-8.85e-04	3.25e-03	0.2668	0.3934	0.0630	0.0705	1.09	1.05	null
6	6	-4.00e-03	-1.21e-03	0.2919	0.3447	0.0600	0.0618	1.14	0.97	null
6	8	3.14e-03	-4.80e-03	0.3028	0.2561	0.0582	0.0562	1.14	0.96	null
6	10	3.44e-03	2.90e-03	0.3388	0.2527	0.0594	0.0566	0.99	0.97	null
6	12	-5.15e-03	4.68e-03	0.2336	0.2484	0.0519	0.0503	0.94	0.99	null
6	14	3.86e-03	-1.47e-03	0.2597	0.3337	0.0551	0.0598	1.01	0.91	null
7	2	-1.84e-03	3.72e-03	0.2585	0.3046	0.0502	0.0664	0.95	1.00	null
7	4	-2.61e-03	-1.27e-03	0.3160	0.2682	0.0578	0.0598	1.00	1.04	null
7	6	-2.40e-04	1.18e-03	0.3431	0.2907	0.0644	0.0533	0.99	0.97	null
7	8	3.31e-03	-2.28e-03	0.2642	0.2758	0.0600	0.0505	1.00	1.04	null
7	10	-1.76e-03	-1.17e-03	0.2983	0.3195	0.0626	0.0520	0.99	0.99	null
7	12	1.33e-03	2.44e-03	0.3023	0.2760	0.0539	0.0524	0.91	0.95	null
7	14	-5.77e-04	-8.79e-04	0.2253	0.3022	0.0442	0.0558	0.97	1.06	null
8	2	2.52e-03	2.26e-03	0.2978	0.2874	0.0621	0.0579	1.05	1.06	null
8	4	4.21e-03	1.54e-03	0.3066	0.3075	0.0583	0.0593	1.03	0.96	null
8	6	-4.25e-03	-8.00e-04	0.2663	0.2756	0.0592	0.0612	1.05	0.95	null
8	8	2.09e-04	-1.08e-03	0.3106	0.2863	0.0682	0.0612	1.05	0.98	null
8	10	-3.32e-03	3.72e-03	0.2924	0.3233	0.0575	0.0575	1.08	1.03	null
8	12	3.88e-03	9.17e-04	0.3039	0.3176	0.0592	0.0602	1.01	0.99	null
8	14	-6.60e-03	-6.12e-03	0.2881	0.2867	0.0623	0.0555	0.95	1.10	null
9	4	2.38e-04	-4.65e-03	0.1790	0.1887	0.0321	0.0387	1.01	1.03	null
9	8	-2.16e-03	5.93e-03	0.2080	0.2078	0.0415	0.0398	1.02	1.05	null

chain	*dump*	$\overline{\Re[\,f-f\,]}$	$\overline{\Im[\,f-f\,]}$	*range* $\Re[\,f-f\,]$	*range* $\Im[\,f-f\,]$	$\sigma_{\Re[\,f-f\,]}$	$\sigma_{\Im[\,f-f\,]}$	$\frac{\sigma_{\Re[\,f-f\,]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[\,f-f\,]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
9	12	-6.32e-04	-2.86e-03	0.2109	0.1878	0.0376	0.0398	1.00	1.00	null
10	4	4.36e-03	2.61e-03	0.1872	0.1915	0.0360	0.0388	0.94	0.97	null
10	8	-2.75e-03	-1.66e-03	0.2644	0.2137	0.0457	0.0428	0.95	1.04	null
10	12	-7.63e-04	2.76e-04	0.1690	0.1960	0.0362	0.0407	0.95	1.00	null
11	4	1.13e-03	4.30e-03	0.1844	0.1718	0.0354	0.0347	1.07	0.96	null
11	8	4.67e-03	-3.34e-03	0.2111	0.1929	0.0445	0.0397	1.05	0.99	null
11	12	-9.97e-04	-3.70e-04	0.1976	0.1954	0.0439	0.0360	1.02	0.95	null
12	4	-3.51e-03	-1.31e-06	0.1794	0.1858	0.0369	0.0320	1.06	0.92	null
12	8	1.97e-03	2.56e-06	0.1666	0.1580	0.0365	0.0360	0.95	1.05	null
12	12	-1.24e-03	2.95e-03	0.2184	0.2098	0.0467	0.0444	1.01	1.02	null
13	8	-2.25e-03	1.70e-03	0.1243	0.1336	0.0241	0.0249	1.00	1.01	null
14	8	-1.85e-03	-4.26e-04	0.1138	0.1341	0.0233	0.0255	0.96	0.98	null
15	8	-2.94e-03	-2.30e-04	0.1090	0.1233	0.0255	0.0227	1.01	0.99	null
16	8	1.02e-03	-1.44e-03	0.1347	0.1300	0.0245	0.0245	0.99	0.99	null

Table 2.5.54 test case 30; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	-7.16e-04	-4.83e-04	0.0184	0.0148	2.65e-03	2.30e-03	4.4	3.8	0	
1	2	-1.85e-04	-8.83e-05	0.0166	0.0111	2.84e-03	2.23e-03	4.7	3.7	0	
1	3	9.28e-04	6.71e-04	0.0276	0.0128	3.42e-03	2.49e-03	5.7	4.2	0	
1	4	1.78e-04	-2.34e-04	0.0170	0.0114	2.46e-03	2.09e-03	4.1	3.5	0	
1	5	-1.64e-04	4.58e-04	0.0124	0.0190	2.36e-03	2.84e-03	3.9	4.7	0	
1	6	-2.83e-04	3.49e-04	0.0276	0.0112	4.09e-03	2.19e-03	6.8	3.7	0	
1	7	-8.10e-05	-3.89e-04	0.0172	0.0185	3.07e-03	2.48e-03	5.1	4.1	0	
1	8	4.20e-04	1.47e-04	0.0197	0.0127	2.58e-03	1.82e-03	4.3	3.0	0	
1	9	2.99e-04	-4.06e-04	0.0158	0.0134	2.35e-03	2.53e-03	3.9	4.2	0	
1	10	-9.43e-04	3.79e-04	0.0188	0.0220	2.74e-03	2.80e-03	4.6	4.7	0	
1	11	-4.21e-04	3.69e-04	0.0141	0.0087	2.84e-03	1.76e-03	4.7	2.9	0	
1	12	7.02e-04	-9.87e-04	0.0199	0.0229	2.81e-03	3.33e-03	4.7	5.5	0	
1	13	-3.45e-06	-5.39e-04	0.0119	0.0136	2.23e-03	2.36e-03	3.7	3.9	0	
1	14	-1.27e-04	3.65e-04	0.0170	0.0134	2.79e-03	2.45e-03	4.6	4.1	0	
1	15	-1.65e-04	5.28e-04	0.0129	0.0282	2.64e-03	3.29e-03	4.4	5.5	0	
1	16	-2.35e-04	-1.79e-04	0.0178	0.0151	2.66e-03	2.32e-03	4.4	3.9	0	
2	1	6.72e-04	-1.47e-05	0.0216	0.0345	3.08e-03	3.88e-03	5.1	6.5	0	
2	2	-1.59e-04	-4.90e-05	0.0196	0.0127	3.19e-03	2.56e-03	5.3	4.3	0	
2	3	-6.66e-04	-4.16e-05	0.0145	0.0317	2.60e-03	3.64e-03	4.3	6.1	0	
2	4	4.40e-04	3.96e-04	0.0216	0.0367	2.63e-03	4.03e-03	4.4	6.7	0	
2	5	-2.56e-06	-2.95e-04	0.0206	0.0290	2.99e-03	3.63e-03	5.0	6.0	0	
2	6	1.39e-05	-4.15e-04	0.0165	0.0183	2.73e-03	2.59e-03	4.6	4.3	0	*
2	7	1.82e-04	4.26e-04	0.0110	0.0415	2.40e-03	4.55e-03	4.0	7.6	0	*
2	8	-4.84e-04	-5.54e-05	0.0139	0.0359	2.48e-03	4.02e-03	4.1	6.7	0	
2	9	5.34e-04	3.99e-05	0.0221	0.0103	3.01e-03	2.08e-03	5.0	3.5	0	
2	10	-6.20e-05	4.92e-04	0.0172	0.0208	2.79e-03	2.63e-03	4.7	4.4	0	
2	11	8.24e-05	-1.35e-03	0.0131	0.0437	2.27e-03	4.69e-03	3.8	7.8	0	
2	12	-4.55e-04	7.27e-04	0.0234	0.0324	3.24e-03	3.84e-03	5.4	6.4	0	
2	13	2.11e-04	1.22e-04	0.0133	0.0133	2.63e-03	2.65e-03	4.4	4.4	0	
2	14	1.15e-04	-1.11e-03	0.0139	0.0241	2.73e-03	3.17e-03	4.5	5.3	0	
2	15	4.21e-05	1.27e-03	0.0205	0.0238	2.73e-03	3.38e-03	4.5	5.6	0	
2	16	-1.20e-04	-5.65e-04	0.0327	0.0190	3.65e-03	2.42e-03	6.1	4.0	0	
3	1	2.17e-04	2.72e-04	0.0146	0.0170	2.55e-03	2.99e-03	4.2	5.0	0	
3	2	4.87e-04	-3.40e-04	0.0139	0.0172	2.53e-03	2.86e-03	4.2	4.8	0	
3	3	-3.70e-04	1.49e-04	0.0155	0.0238	2.58e-03	2.92e-03	4.3	4.9	0	
3	4	-3.23e-04	3.00e-04	0.0149	0.0243	2.62e-03	2.79e-03	4.4	4.6	0	
3	5	4.79e-04	1.27e-05	0.0231	0.0168	3.05e-03	2.66e-03	5.1	4.4	0	
3	6	-3.42e-04	-1.75e-04	0.0103	0.0283	2.38e-03	3.58e-03	4.0	6.0	0	
3	7	-3.19e-04	-7.12e-04	0.0106	0.0156	2.24e-03	2.86e-03	3.7	4.8	0	
3	8	-1.35e-04	6.52e-04	0.0191	0.0154	2.75e-03	2.32e-03	4.6	3.9	0	
3	9	2.22e-04	-1.03e-04	0.0123	0.0185	2.40e-03	2.72e-03	4.0	4.5	0	
3	10	-3.09e-04	4.67e-04	0.0260	0.0186	3.02e-03	2.95e-03	5.0	4.9	0	
3	11	-7.30e-05	3.10e-04	0.0126	0.0228	2.51e-03	2.94e-03	4.2	4.9	0	
3	12	-1.06e-05	5.56e-05	0.0142	0.0201	2.64e-03	2.79e-03	4.4	4.6	0	
3	13	2.04e-04	-2.16e-04	0.0178	0.0131	2.79e-03	2.31e-03	4.6	3.9	0	

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
3	14	-3.59e-05	3.14e-04	0.0135	0.0142	2.58e-03	2.45e-03	4.3	4.1	0	
3	15	-2.50e-04	-7.29e-04	0.0130	0.0152	2.74e-03	2.45e-03	4.6	4.1	0	
3	16	-1.69e-06	5.11e-04	0.0177	0.0129	2.56e-03	2.33e-03	4.3	3.9	0	
4	1	1.40e-03	-1.54e-04	0.0143	0.0099	2.79e-03	2.11e-03	4.6	3.5	0	
4	2	3.39e-05	9.74e-04	0.0187	0.0135	2.68e-03	2.25e-03	4.5	3.8	0	
4	3	-6.45e-04	-4.90e-04	0.0207	0.0142	2.82e-03	2.26e-03	4.7	3.8	0	
4	4	2.64e-04	-9.53e-04	0.0164	0.0270	2.92e-03	3.30e-03	4.9	5.5	0	
4	5	5.13e-04	1.20e-03	0.0166	0.0323	2.88e-03	3.86e-03	4.8	6.4	0	
4	6	-4.95e-05	2.40e-04	0.0129	0.0121	2.27e-03	2.15e-03	3.8	3.6	0	
4	7	-1.50e-04	-9.95e-04	0.0147	0.0157	2.34e-03	2.33e-03	3.9	3.9	0	
4	8	1.76e-04	-5.46e-04	0.0206	0.0126	2.93e-03	2.23e-03	4.9	3.7	0	
4	9	7.29e-04	1.00e-03	0.0220	0.0157	3.05e-03	2.47e-03	5.1	4.1	0	
4	10	-2.11e-04	3.50e-04	0.0141	0.0232	2.61e-03	3.06e-03	4.4	5.1	0	
4	11	-2.59e-04	-1.29e-03	0.0140	0.0283	2.41e-03	3.77e-03	4.0	6.3	0	
4	12	7.87e-04	7.36e-04	0.0129	0.0141	2.54e-03	2.30e-03	4.2	3.8	0	
4	13	-1.04e-03	9.34e-04	0.0155	0.0264	2.36e-03	3.31e-03	3.9	5.5	0	
4	14	-6.13e-04	-1.19e-03	0.0229	0.0159	3.20e-03	2.36e-03	5.3	3.9	0	
4	15	1.56e-03	-3.96e-04	0.0165	0.0184	2.96e-03	2.66e-03	4.9	4.4	0	
4	16	-2.30e-04	1.09e-03	0.0124	0.0220	2.52e-03	2.99e-03	4.2	5.0	0	
5	2	-4.97e-06	1.40e-04	0.0072	0.0095	1.42e-03	1.25e-03	3.3	2.9	0	
5	4	-1.51e-04	-2.69e-05	0.0079	0.0066	1.38e-03	1.18e-03	3.2	2.8	0	
5	6	2.06e-04	-2.35e-05	0.0099	0.0081	1.52e-03	1.39e-03	3.6	3.3	0	
5	8	-4.47e-05	6.88e-05	0.0065	0.0104	1.23e-03	1.48e-03	2.9	3.5	0	
5	10	2.82e-06	3.08e-05	0.0115	0.0062	1.38e-03	1.01e-03	3.3	2.4	0	
5	12	6.52e-05	2.20e-04	0.0076	0.0072	1.16e-03	1.21e-03	2.7	2.9	0	
5	14	1.65e-05	-9.14e-06	0.0070	0.0079	1.22e-03	1.35e-03	2.9	3.2	0	
5	16	-2.29e-05	-9.16e-05	0.0081	0.0098	1.44e-03	1.38e-03	3.4	3.3	0	
6	2	2.90e-05	1.28e-04	0.0089	0.0069	1.26e-03	1.19e-03	3.0	2.8	0	
6	4	-2.64e-06	-2.72e-04	0.0083	0.0089	1.12e-03	1.32e-03	2.6	3.1	0	
6	6	-3.55e-04	-1.65e-04	0.0119	0.0076	1.86e-03	1.38e-03	4.4	3.3	0	
6	8	4.20e-04	2.44e-05	0.0083	0.0089	1.39e-03	1.31e-03	3.3	3.1	0	
6	10	2.14e-05	-1.96e-04	0.0083	0.0093	1.39e-03	1.42e-03	3.3	3.3	0	
6	12	1.03e-04	9.01e-05	0.0072	0.0070	1.28e-03	1.27e-03	3.0	3.0	0	
6	14	1.21e-04	1.48e-04	0.0127	0.0082	1.50e-03	1.15e-03	3.5	2.7	0	
6	16	-4.83e-05	1.10e-05	0.0095	0.0065	1.46e-03	1.16e-03	3.4	2.7	0	
7	2	5.06e-05	2.62e-04	0.0092	0.0106	1.41e-03	1.45e-03	3.3	3.4	0	
7	4	9.74e-05	1.50e-05	0.0059	0.0071	1.09e-03	1.36e-03	2.6	3.2	0	
7	6	-1.63e-04	-5.61e-05	0.0094	0.0078	1.52e-03	1.23e-03	3.6	2.9	0	
7	8	1.64e-04	-1.07e-04	0.0108	0.0046	1.52e-03	1.05e-03	3.6	2.5	0	
7	10	-2.04e-04	1.65e-04	0.0106	0.0074	1.46e-03	1.26e-03	3.4	3.0	0	
7	12	2.48e-04	2.83e-05	0.0076	0.0068	1.22e-03	1.34e-03	2.9	3.2	0	
7	14	-3.33e-05	-2.31e-04	0.0114	0.0117	1.48e-03	1.56e-03	3.5	3.7	0	
7	16	-1.77e-04	-4.22e-04	0.0057	0.0095	1.19e-03	1.43e-03	2.8	3.4	0	
8	2	7.34e-05	-7.81e-05	0.0098	0.0056	1.41e-03	1.21e-03	3.3	2.9	0	
8	4	-2.95e-05	7.81e-05	0.0089	0.0084	1.40e-03	1.31e-03	3.3	3.1	0	

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*	*plot*
8	6	-1.28e-05	-2.07e-05	0.0125	0.0080	1.86e-03	1.27e-03	4.4	3.0	0	
8	8	5.55e-05	8.26e-05	0.0065	0.0083	1.16e-03	1.17e-03	2.7	2.8	0	
8	10	8.01e-05	3.14e-04	0.0101	0.0068	1.51e-03	1.36e-03	3.5	3.2	0	
8	12	-2.32e-04	1.21e-04	0.0075	0.0062	1.26e-03	1.11e-03	3.0	2.6	0	
8	14	-1.07e-04	-2.67e-04	0.0153	0.0076	1.86e-03	1.34e-03	4.4	3.1	0	
8	16	4.03e-04	-3.06e-04	0.0065	0.0102	1.20e-03	1.49e-03	2.8	3.5	0	
9	4	1.91e-04	-7.02e-05	0.0066	0.0032	9.18e-04	6.30e-04	3.1	2.1	0	
9	8	6.63e-05	-9.27e-05	0.0104	0.0053	1.36e-03	7.77e-04	4.5	2.6	0	
9	12	-8.05e-06	1.38e-04	0.0056	0.0034	8.88e-04	6.42e-04	3.0	2.1	0	
9	16	-3.02e-05	-4.79e-05	0.0059	0.0025	8.92e-04	5.65e-04	3.0	1.9	0	
10	4	-1.42e-04	-1.18e-04	0.0059	0.0044	7.69e-04	6.79e-04	2.6	2.3	0	
10	8	3.71e-05	4.21e-05	0.0081	0.0061	1.14e-03	9.46e-04	3.8	3.2	0	
10	12	1.07e-04	6.25e-05	0.0087	0.0064	9.03e-04	9.22e-04	3.0	3.1	0	
10	16	-1.07e-07	1.65e-04	0.0063	0.0046	8.93e-04	7.82e-04	3.0	2.6	0	
11	4	-1.55e-04	-4.93e-05	0.0049	0.0045	8.05e-04	7.53e-04	2.7	2.5	0	
11	8	6.85e-05	-4.11e-05	0.0088	0.0046	1.15e-03	7.89e-04	3.8	2.6	0	
11	12	1.97e-05	4.15e-05	0.0055	0.0037	7.43e-04	6.34e-04	2.5	2.1	0	
11	16	4.35e-05	-1.00e-04	0.0059	0.0056	9.19e-04	7.81e-04	3.1	2.6	0	
12	4	-1.32e-04	4.61e-05	0.0078	0.0030	1.02e-03	5.16e-04	3.4	1.7	0	
12	8	-3.05e-05	2.63e-05	0.0070	0.0037	9.69e-04	6.30e-04	3.2	2.1	0	
12	12	1.47e-04	-3.07e-05	0.0070	0.0028	8.72e-04	5.34e-04	2.9	1.8	0	
12	16	2.15e-05	-6.85e-05	0.0076	0.0027	9.36e-04	5.19e-04	3.1	1.7	0	
13	8	3.47e-06	-6.51e-05	0.0042	0.0023	7.29e-04	4.46e-04	3.4	2.1	0	
13	16	6.06e-05	4.78e-05	0.0063	0.0026	8.09e-04	4.08e-04	3.8	1.9	0	
14	8	-2.67e-05	5.48e-05	0.0039	0.0019	6.97e-04	3.62e-04	3.3	1.7	0	
14	16	4.53e-05	-4.91e-05	0.0047	0.0030	6.85e-04	4.26e-04	3.2	2.0	0	
15	8	5.78e-05	-9.18e-05	0.0041	0.0028	5.96e-04	4.42e-04	2.8	2.1	0	
15	16	-5.19e-05	1.04e-04	0.0044	0.0029	6.65e-04	4.56e-04	3.1	2.1	0	
16	8	-6.03e-05	3.89e-05	0.0039	0.0022	6.72e-04	3.86e-04	3.2	1.8	0	
16	16	4.84e-05	9.29e-06	0.0047	0.0025	7.45e-04	4.62e-04	3.5	2.2	0	

Table 2.5.55 test case 30; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*applied dump - bias: ! OVR*	*bias: OVR*	*applied dump - bias: OVR*
0	12000	11724	276	0	n-a
1	12000	11724	276	0	n-a
2	24000	23724	276	0	n-a
3	24000	23724	276	0	n-a
4	48000	47724	276	0	n-a
5	48000	47724	276	0	n-a
6	96000	95724	276	0	n-a
7	96000	95724	276	0	n-a

Table 2.5.56 test case 30: applied dump and bias.

```
#-------------------------------------------------------------------------------
# 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.
#
# -------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10100110  CCCSCR_0_1:   CCC1: Y=sec X=sec   CCC0: Y=pri X=sec
10010101  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=pri X=pri
10100110  CCCSCR_4_5:   CCC5: Y=sec X=sec   CCC4: Y=pri X=sec
10010101  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=pri X=pri
10100110  CCCSCR_8_9:   CCC9: Y=sec X=sec   CCC8: Y=pri X=sec
10010101  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=pri X=pri
10100110  CCCSCR_12_13: CCC13: Y=sec X=sec CCC12: Y=pri X=sec
10010101  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=pri X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.34 content of file mcb_reg30.txt

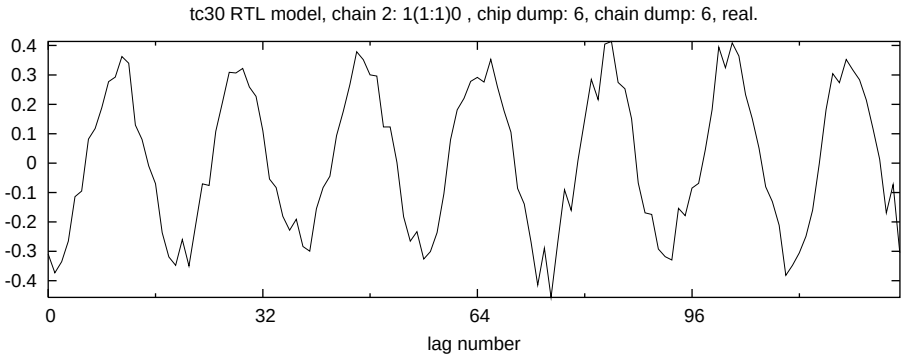

Illustration 2.5.349

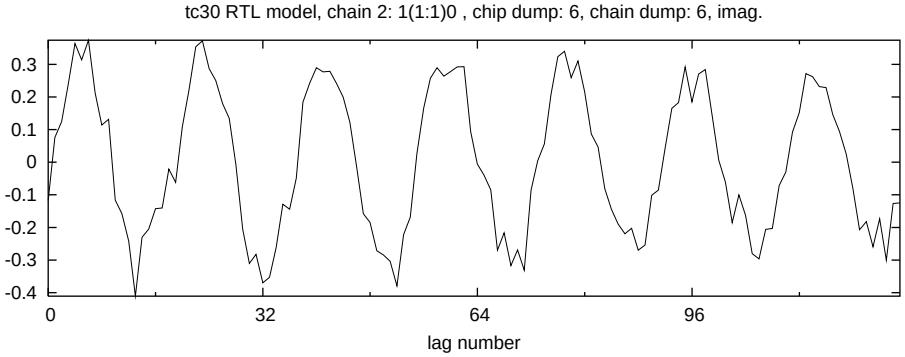

Illustration 2.5.350

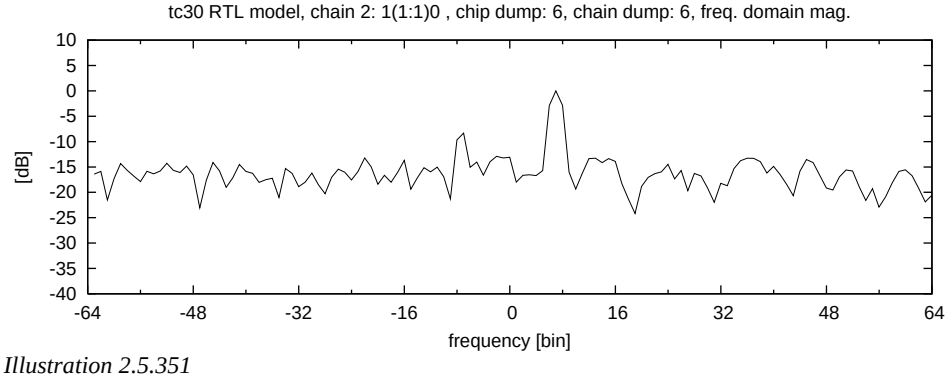

Illustration 2.5.351

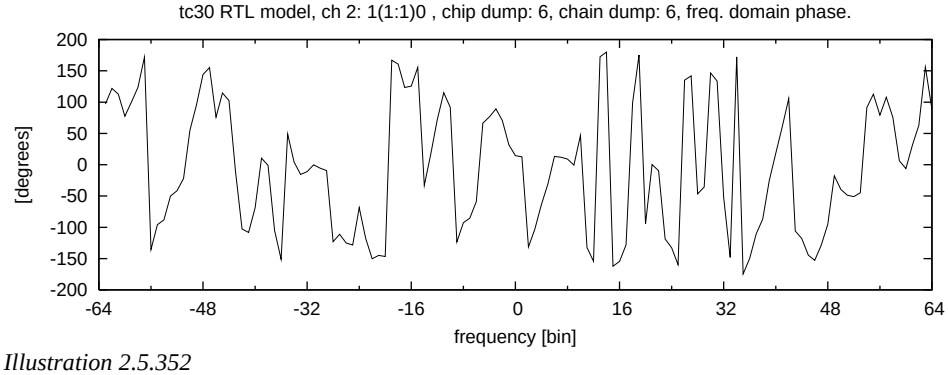

Illustration 2.5.352

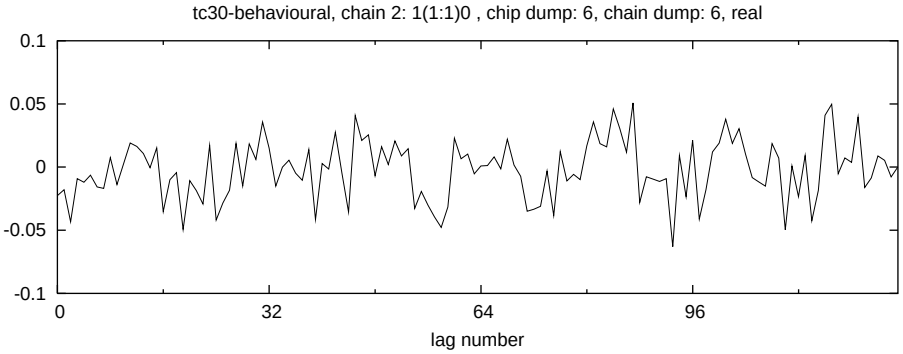

Illustration 2.5.353

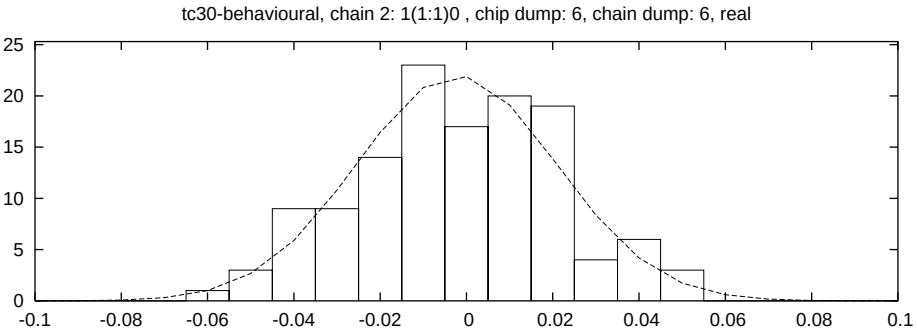

Illustration 2.5.354

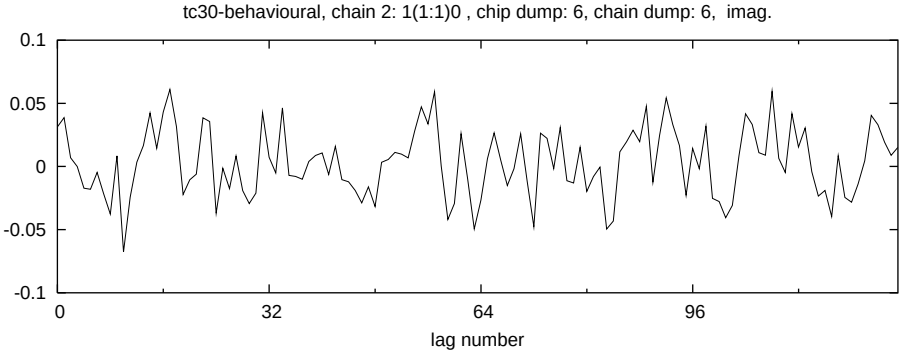

Illustration 2.5.355

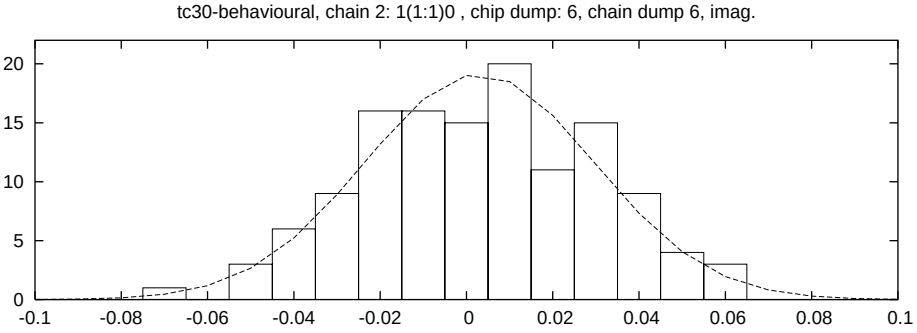

Illustration 2.5.356

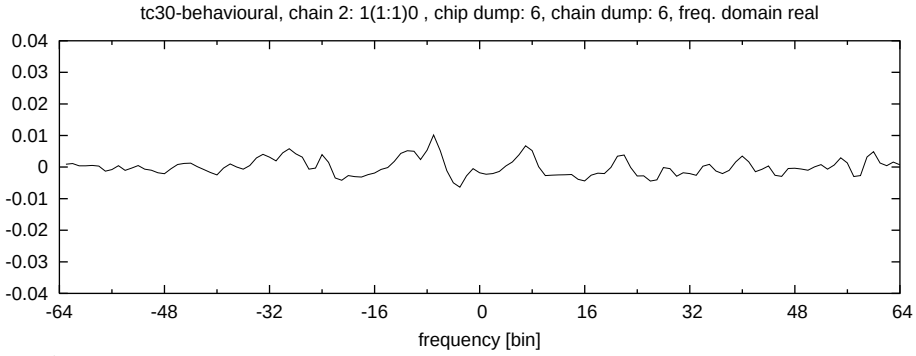

Illustration 2.5.357

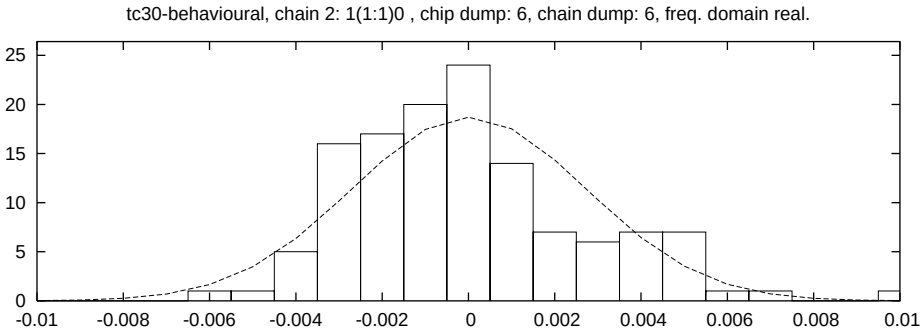

Illustration 2.5.358

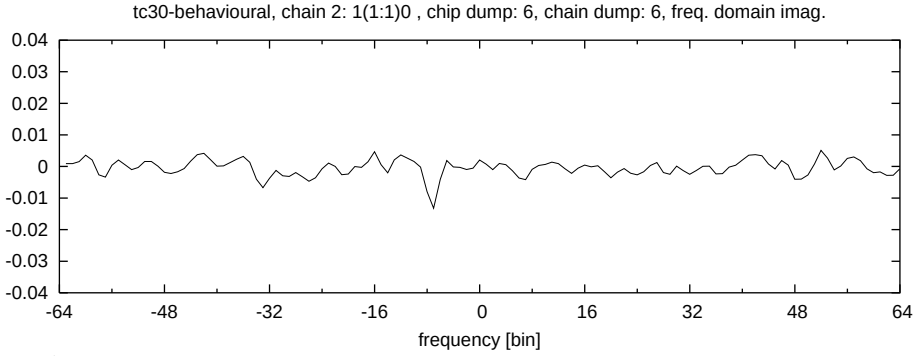

Illustration 2.5.359

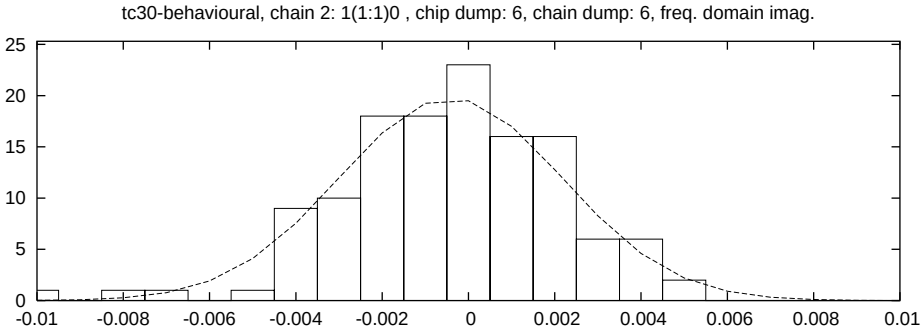

Illustration 2.5.360

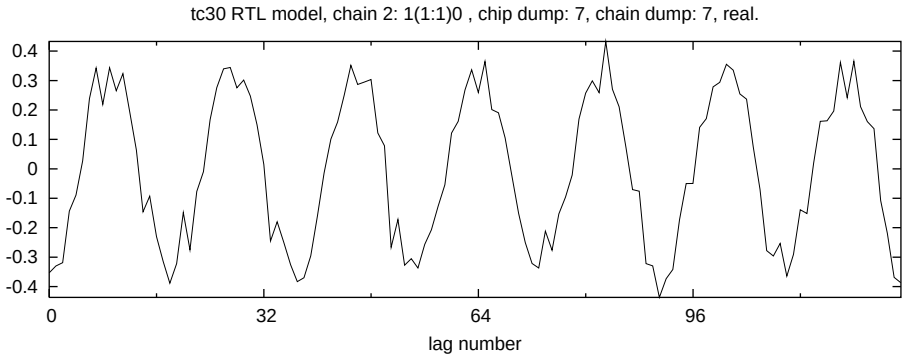

Illustration 2.5.361

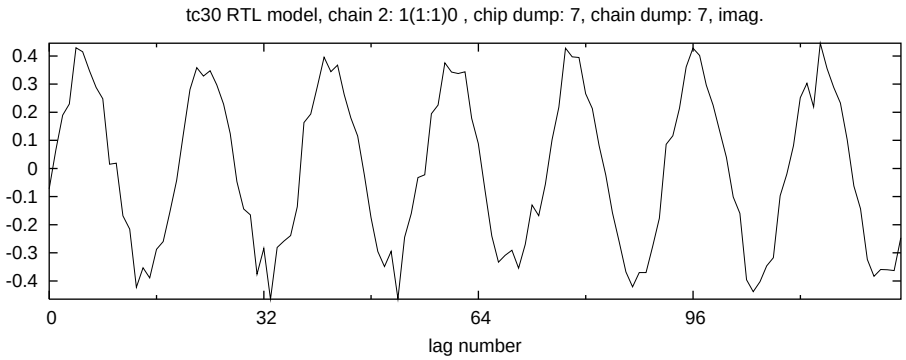

Illustration 2.5.362

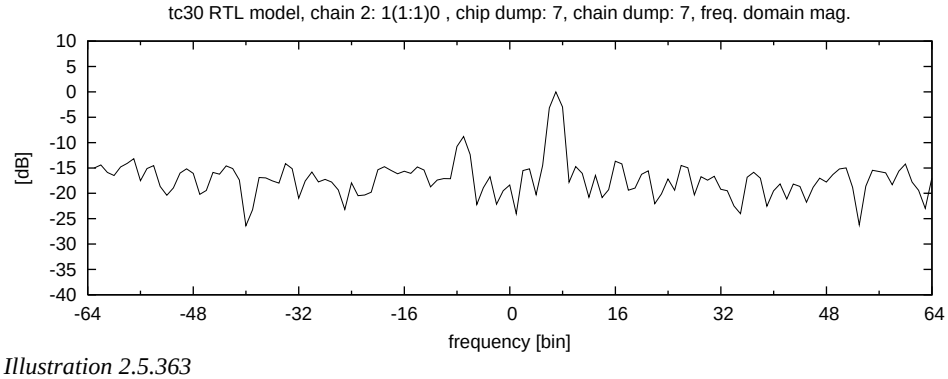

Illustration 2.5.363

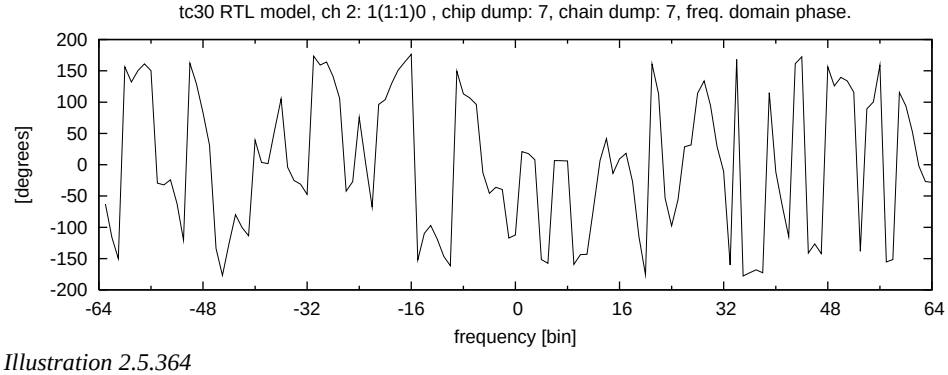

Illustration 2.5.364

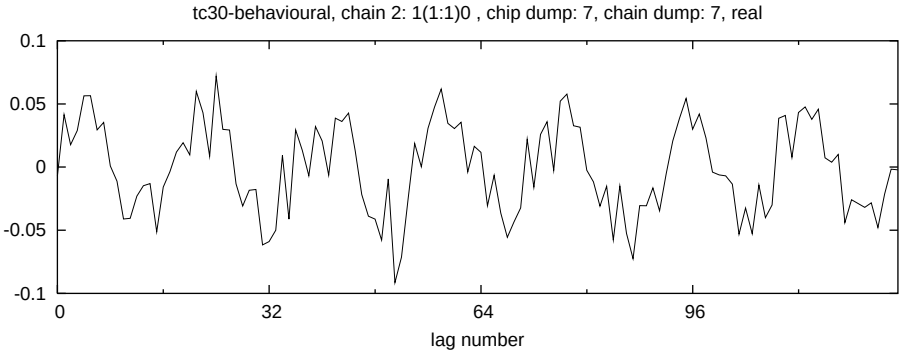

Illustration 2.5.365

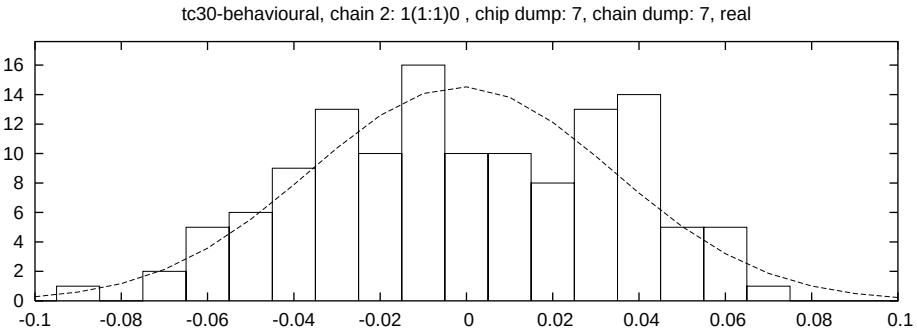

Illustration 2.5.366

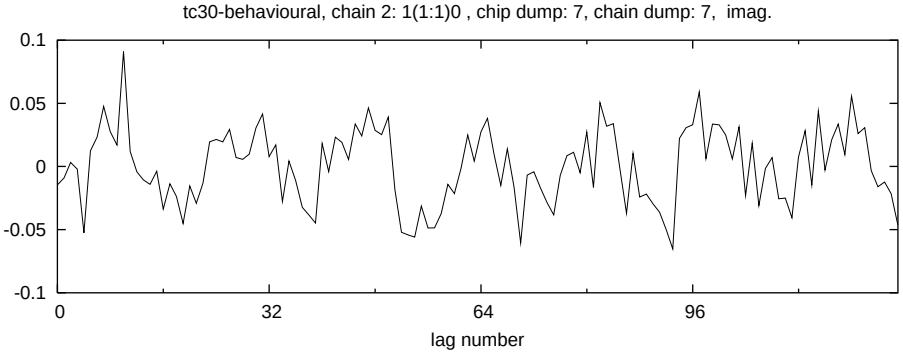

Illustration 2.5.367

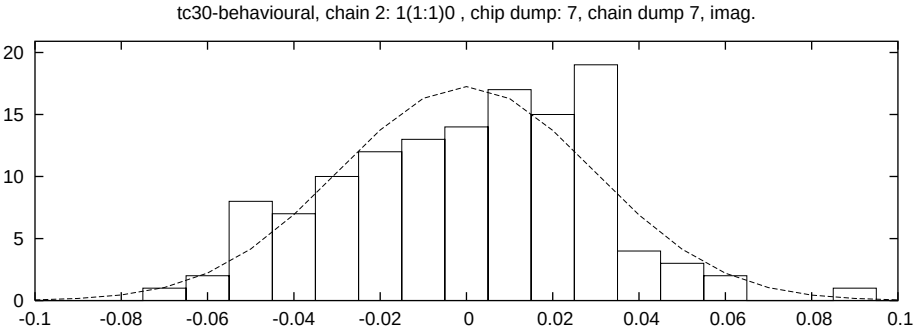

Illustration 2.5.368

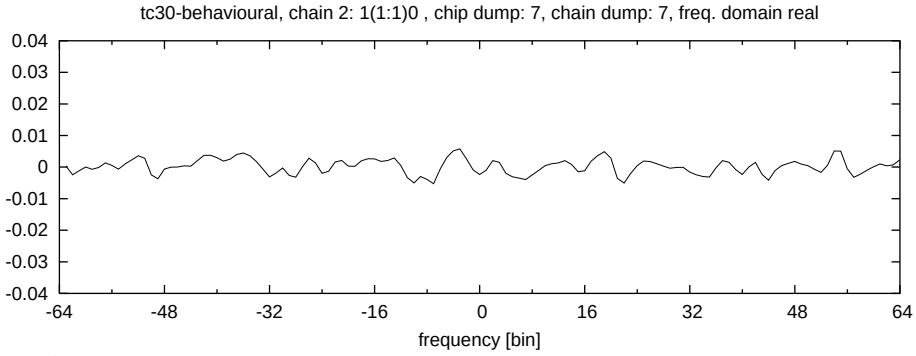

Illustration 2.5.369

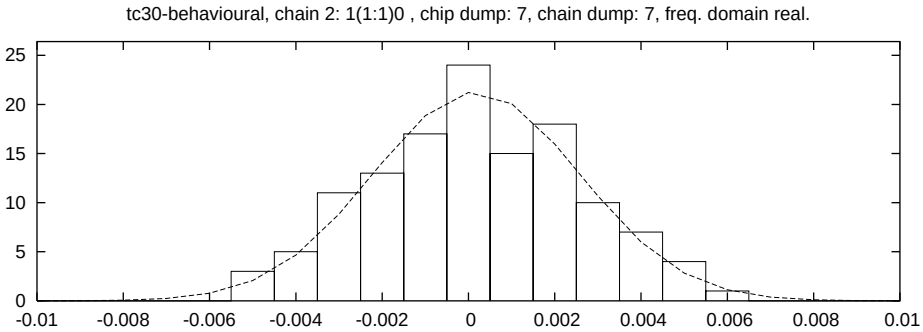

Illustration 2.5.370

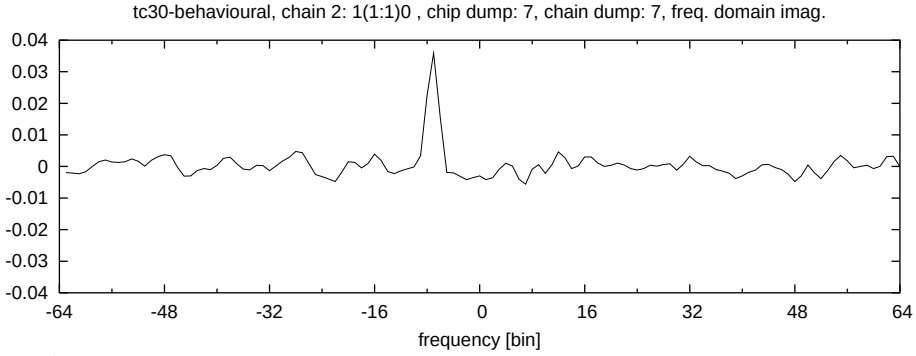

Illustration 2.5.371

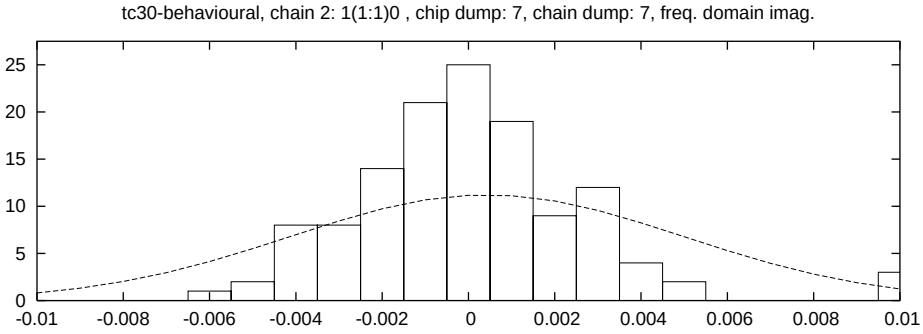

Illustration 2.5.372

2.5.15 test case 64

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc64_postsim.m : V2.00 run time: Mon Sep 05 14:35:48 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 64.
test case version: v2,05apr24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 2.
0(0:7)0  1(8:15)1

Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc64
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
```

Text 2.5.35 console log for scripts tc64_postsim.

chain	dump	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	range $\Re[f-b]$	range $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	peak count	plot
1	1	6.60e-04	-6.28e-04	0.2091	0.2238	0.0237	0.0249	1.08	1.13	7	*
1	2	-1.25e-03	4.96e-04	0.1695	0.1521	0.0235	0.0231	1.07	1.05	6	
2	1	8.84e-05	1.36e-04	0.1589	0.1413	0.0216	0.0215	0.99	0.98	9	*
2	2	-4.35e-04	2.56e-04	0.1519	0.1391	0.0228	0.0227	1.04	1.04	4	

Table 2.5.57 test case 64; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	dump	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	range $\Re[f-f]$	range $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	co-dump peaks
1	1	-2.02e-03	2.40e-03	0.6122	0.4927	0.0902	0.0880	0.94	0.93	511 512
2	1	-2.10e-03	1.09e-03	0.5531	0.5240	0.0881	0.0866	0.99	0.97	null

Table 2.5.58 test case 64; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	dump	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	range $\Re[f^F-b^F]$	range $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	peak count	plot
1	1	-3.11e-05	2.70e-04	0.0091	0.0255	8.12e-04	1.21e-03	1.2	1.8	2	*
1	2	-1.71e-04	-1.61e-04	0.0147	0.0137	9.67e-04	8.65e-04	1.5	1.3	2	
2	1	3.22e-05	-6.05e-05	0.0065	0.0133	7.15e-04	7.65e-04	1.1	1.2	1	*
2	2	8.51e-05	9.27e-05	0.0143	0.0125	8.96e-04	8.74e-04	1.4	1.3	2	

Table 2.5.59 test case 64; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	applied dump	bias:!OVR	applied dump - bias: ! OVR	bias: OVR	applied dump - bias: OVR
0	20000	19724	276	0	n-a
1	20000	19724	276	0	n-a

Table 2.5.60 test case 64; applied dump interval and bias.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
10000000  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

01000000  CCQR_0_1: Q1: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
10000010  CCQR_2_3: Q3: YSw1=sec YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=sec XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111100  XSTATUS: X-SE_CLK: 11111100
11111100  YSTATUS: Y-SE_CLK: 11111100
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.5.36 content of file mcb_reg64.txt

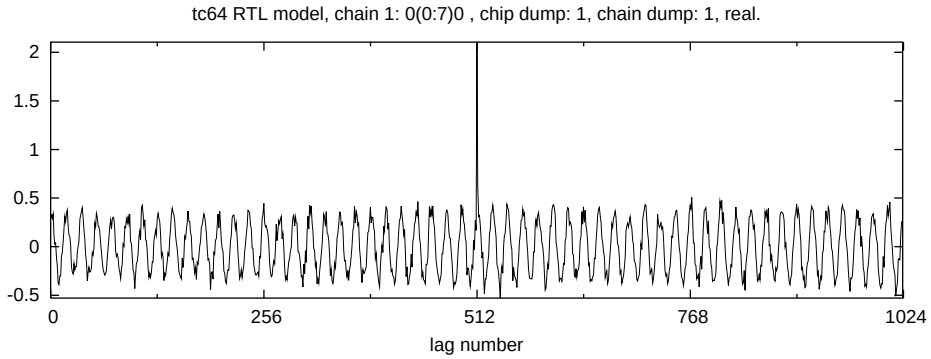

Illustration 2.5.373

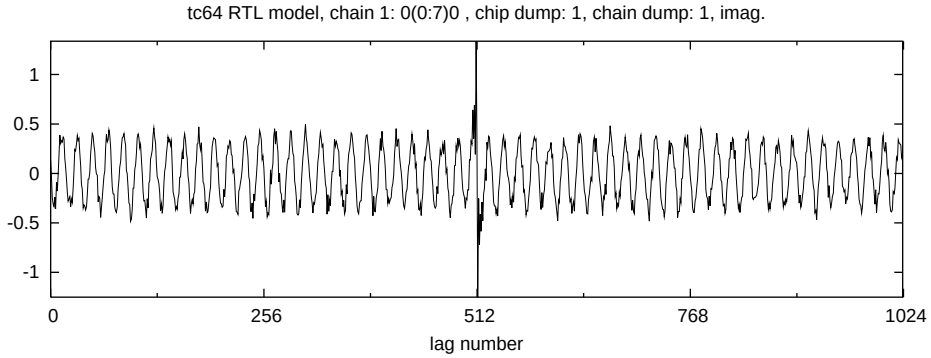

Illustration 2.5.374

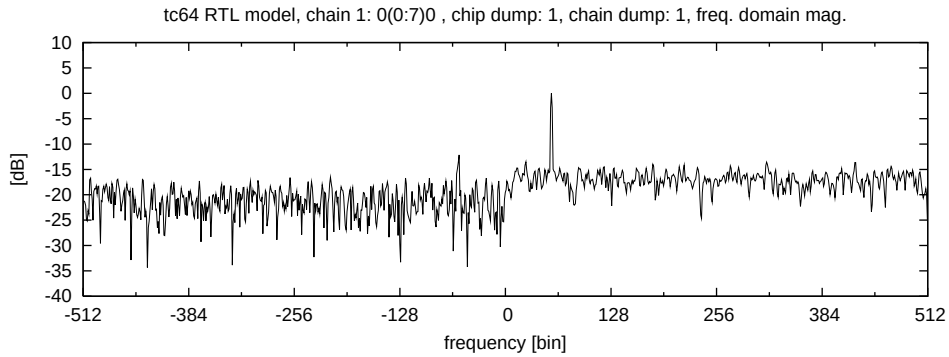

Illustration 2.5.375

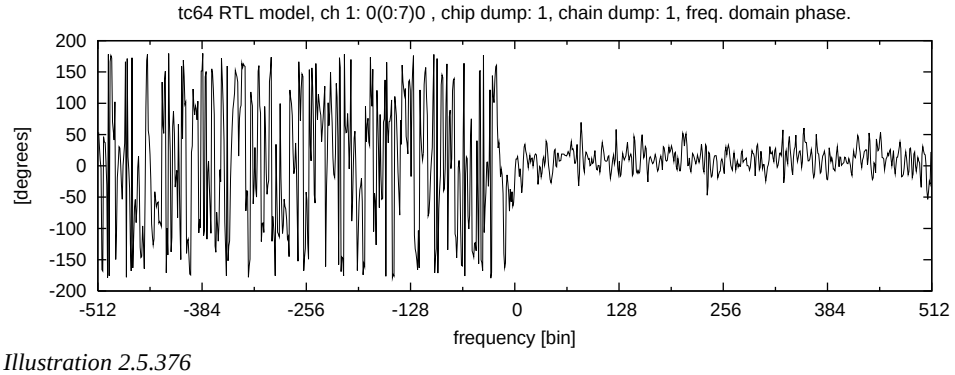

Illustration 2.5.376

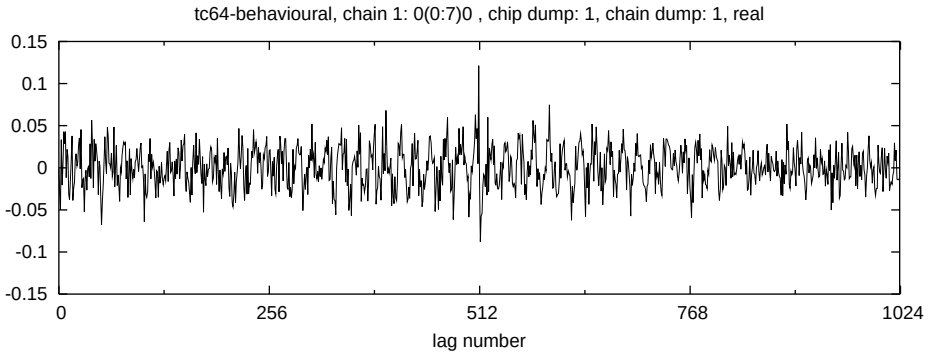

Illustration 2.5.377

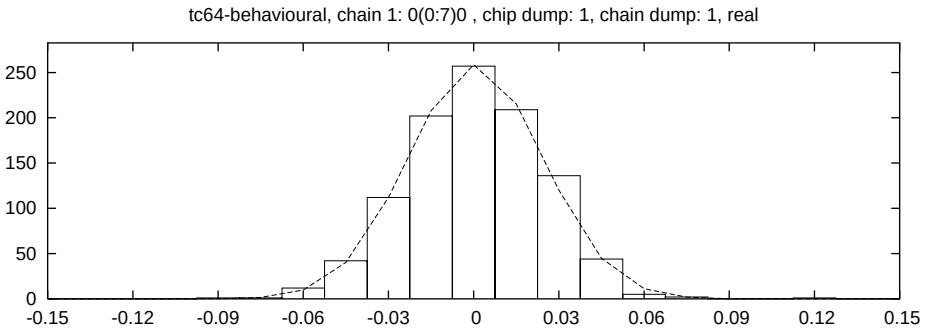

Illustration 2.5.378

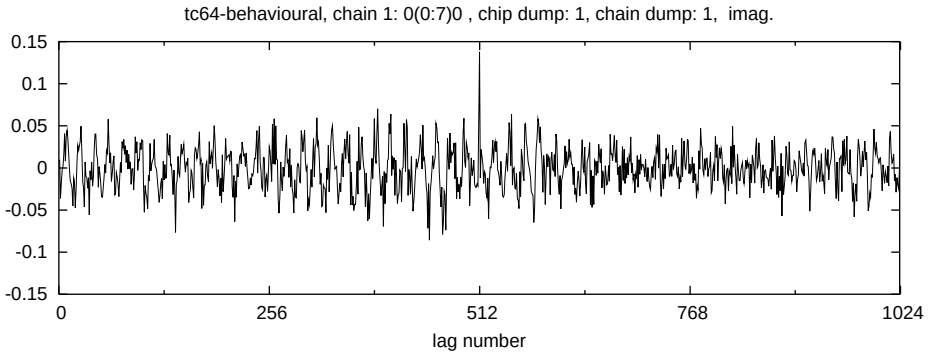

Illustration 2.5.379

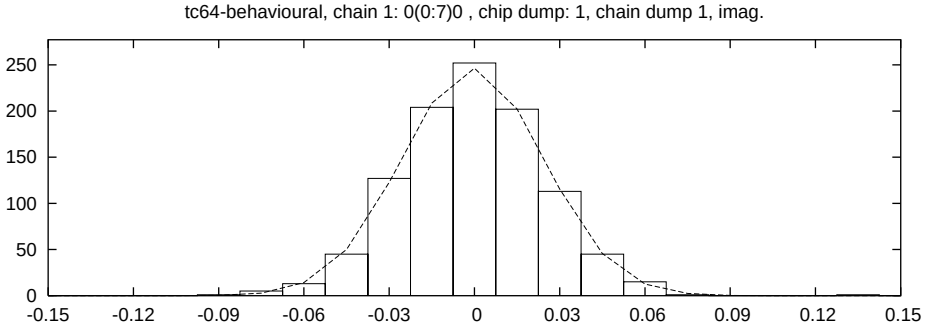

Illustration 2.5.380

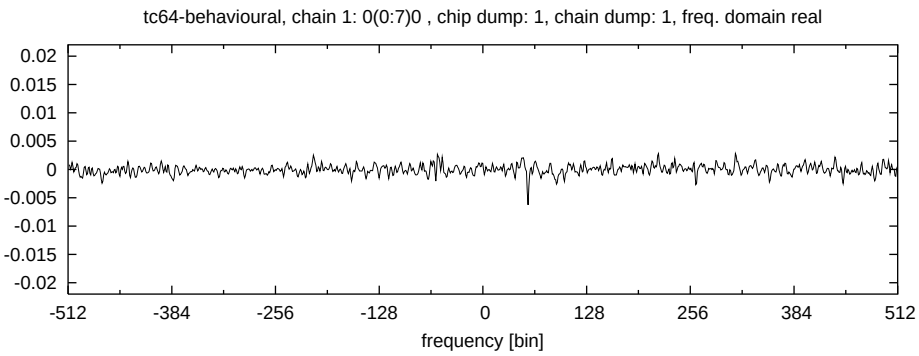

Illustration 2.5.381

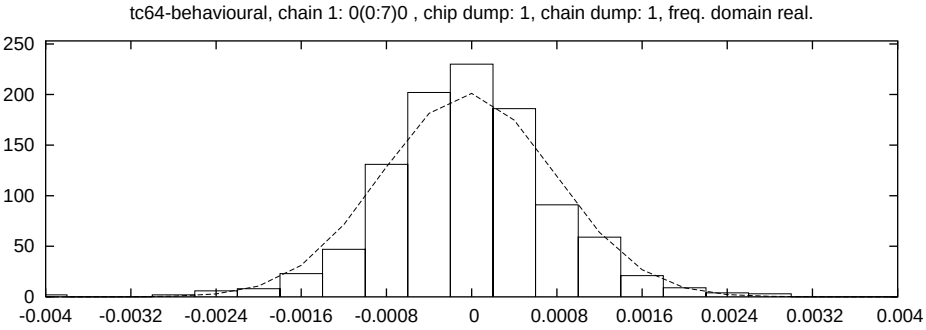

Illustration 2.5.382

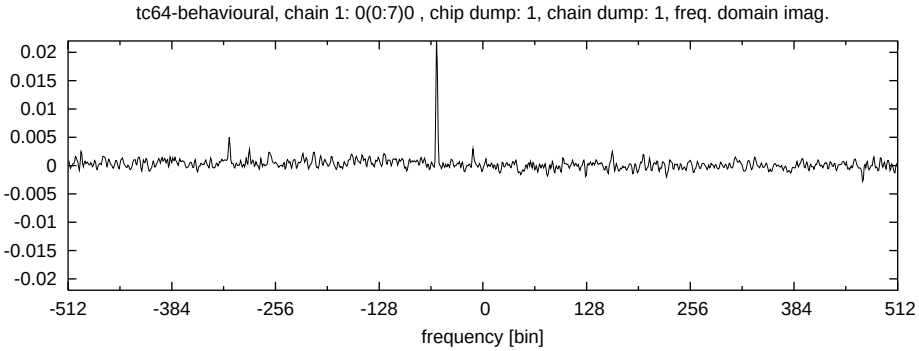

Illustration 2.5.383

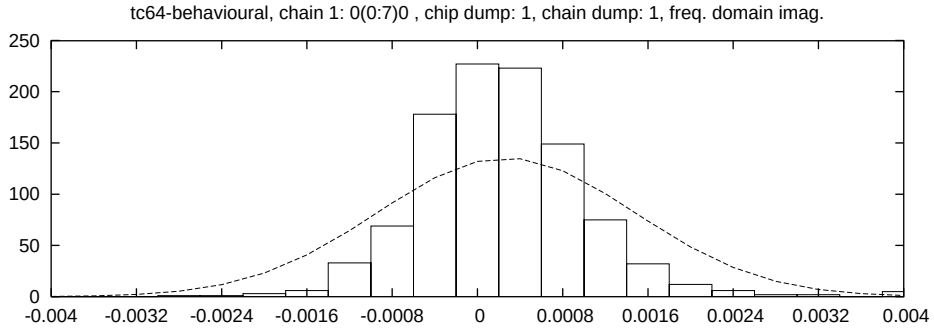

Illustration 2.5.384

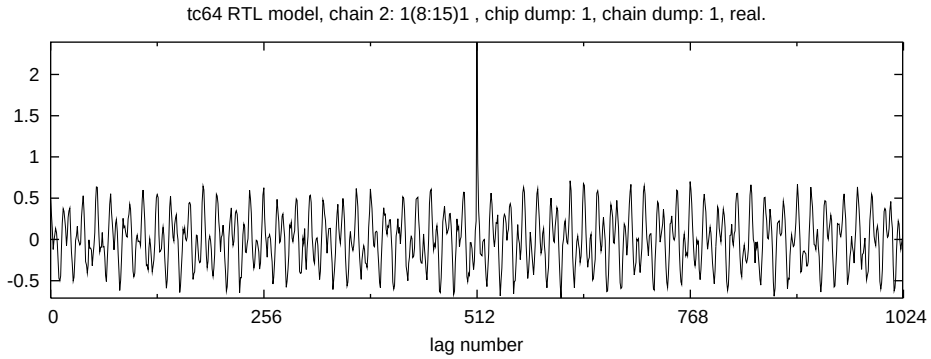

Illustration 2.5.385

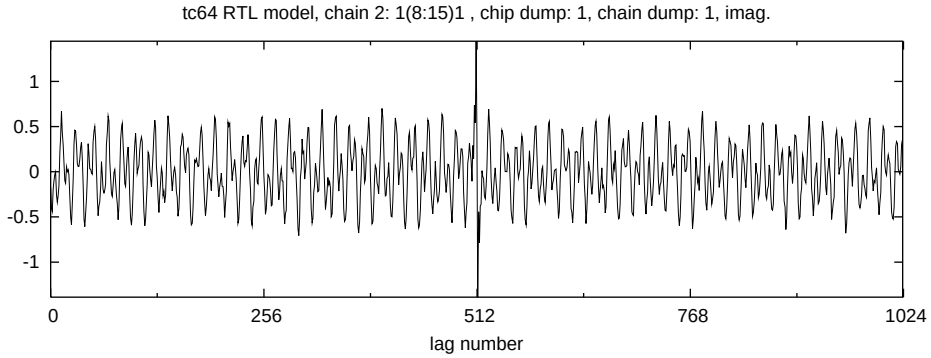

Illustration 2.5.386

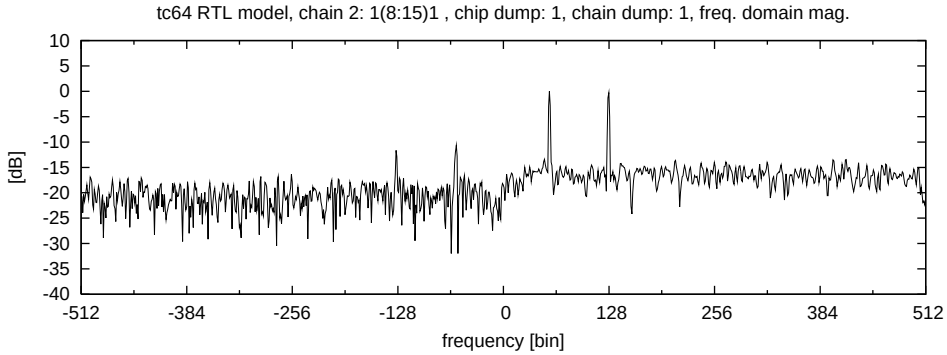

Illustration 2.5.387

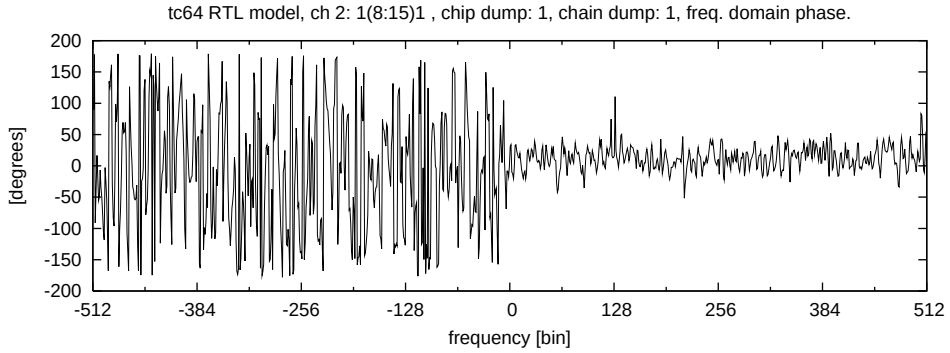

Illustration 2.5.388

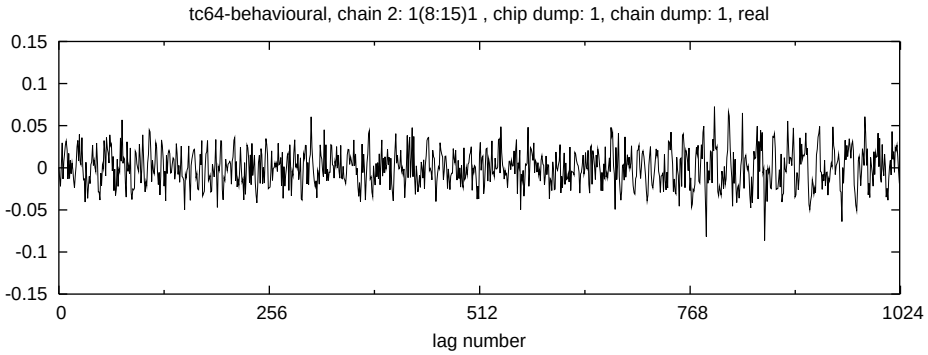

Illustration 2.5.389

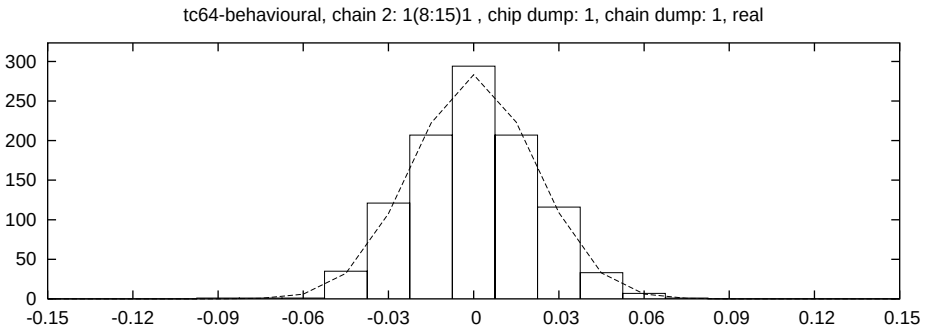

Illustration 2.5.390

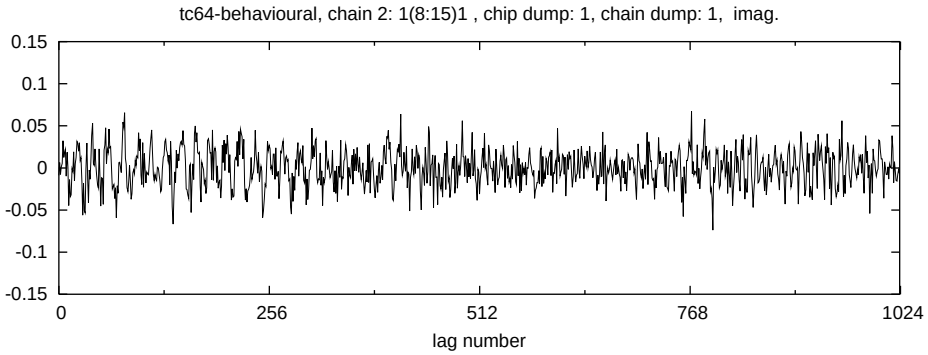

Illustration 2.5.391

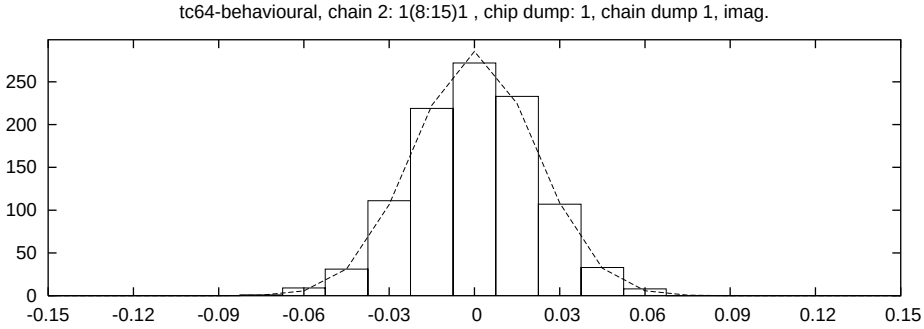

Illustration 2.5.392

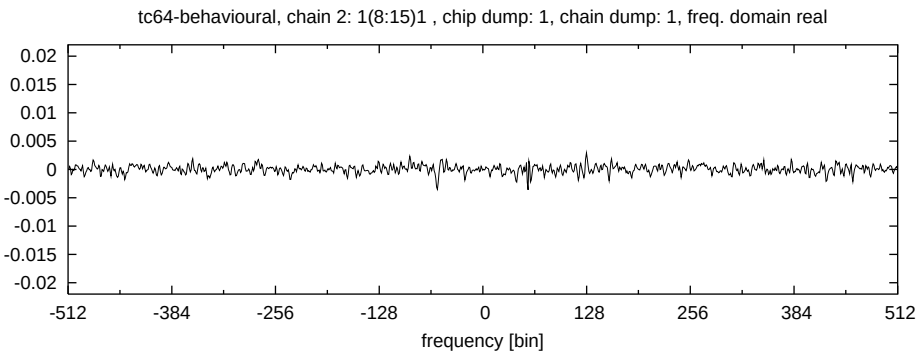

Illustration 2.5.393

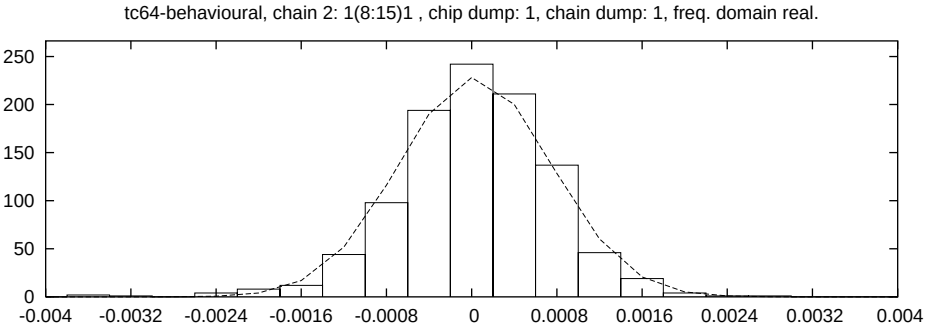

Illustration 2.5.394

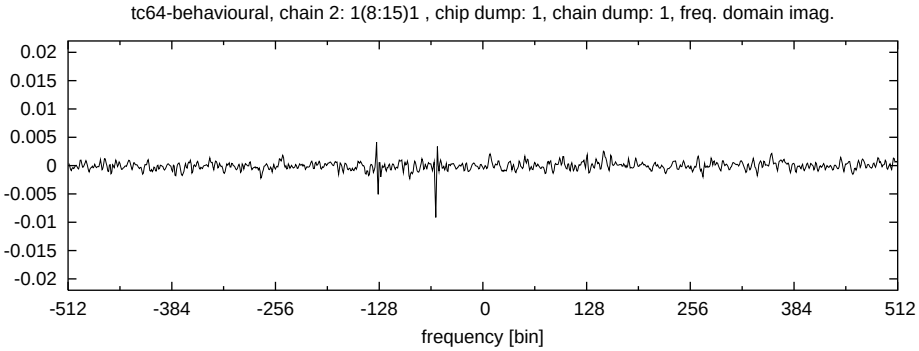

Illustration 2.5.395

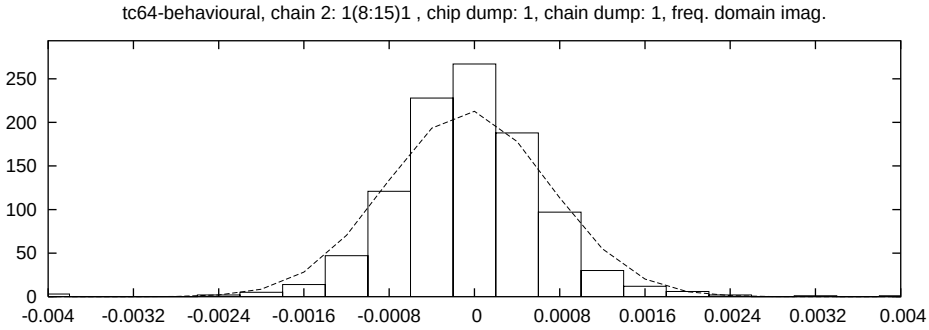

Illustration 2.5.396

2.6 shift enable clock (X|Y-SE_CLK)

Test cases: 21, 22

2.6.1 organization

Validation of the shift enable clock (**X|Y-SE_CLK**) operation occurs coincidently with that of the **CMAM** in section 2.3 and *normal dump and readout* in 2.5. An additional result is presented in 2.7.3. Test cases of this section were created to specifically to further demonstrate operation of the shift enable clock. The material of this section is interpreted and organized as described in 2.5.1 with the exception that plots are not presented.

2.6.2 test case 21

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged. Statistical parameters tabulated in Table 2.6.1 to Table 2.6.3 are all within expected limits.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc21_postsim.m : V2.00 run time: Sun Oct 02 15:02:15 2005
Using postsim_lib: 1.40: 05-08-18

48 frames read for test case 21.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 2.
0(0:7)3  4(8:15)7

Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.10: 05-10-02
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc21
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
```

Text 2.6.1: console log for script tc21_postsim.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	2	3.16e-04	6.30e-04	0.0947	0.0992	0.0169	0.0159	0.85	0.80	2	
1	4	-1.34e-05	-7.34e-04	0.1246	0.1299	0.0200	0.0198	1.00	0.99	8	
2	1	-5.54e-04	6.17e-04	0.1431	0.1309	0.0214	0.0216	1.07	1.08	6	
2	2	3.03e-04	1.71e-04	0.1319	0.1445	0.0228	0.0235	1.14	1.17	3	
2	3	1.62e-04	7.90e-04	0.1457	0.1398	0.0206	0.0227	1.03	1.14	7	
2	4	-5.02e-04	-7.47e-04	0.1324	0.1472	0.0220	0.0239	1.10	1.19	6	

Table 2.6.1: test case 21; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	2	-6.46e-04	-3.23e-03	0.4459	0.5779	0.0782	0.0818	1.03	1.07	null
2	1	-1.87e-03	-7.09e-04	0.4699	0.5896	0.0771	0.0800	1.00	1.01	null
2	2	-8.87e-04	-2.09e-04	0.5576	0.4933	0.0804	0.0818	1.00	1.02	null
2	3	8.13e-04	-1.66e-03	0.5305	0.5178	0.0770	0.0751	1.01	1.02	508

Table 2.6.2: test case 21; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	2	-2.23e-06	-3.02e-05	0.0057	0.0091	6.06e-04	6.66e-04	1.0	1.1	0	
1	4	3.16e-06	4.54e-05	0.0104	0.0095	7.34e-04	8.02e-04	1.2	1.3	0	
2	1	-4.75e-05	4.98e-07	0.0109	0.0090	8.25e-04	7.95e-04	1.4	1.3	0	
2	2	3.54e-05	7.45e-05	0.0110	0.0124	8.26e-04	8.98e-04	1.4	1.5	0	
2	3	-5.75e-05	-3.51e-05	0.0071	0.0115	7.34e-04	8.63e-04	1.2	1.4	0	
2	4	3.72e-06	5.22e-05	0.0124	0.0123	8.35e-04	8.79e-04	1.4	1.5	0	

Table 2.6.3: test case 21; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

stream	*applied dump*	*bias:!OVR*	*dump-bias:!OVR*	*bias:OVR*	*dump-bias:OVR*
0	24000	23724	276	0	n-a
4	12000	11724	276	0	n-a

Table 2.6.4: test case 21; applied dump interval and bias.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000001  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=pri
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11101110  XSTATUS: X-SE_CLK: 11101110
01110111  YSTATUS: Y-SE_CLK: 01110111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.6.2: content of file mcb_reg21.txt.

2.6.3 test case 22

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged. Statistical parameters tabulated in Table 2.6.5 to Table 2.6.7 are all within expected limits.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc22_postsim.m : V2.00 run time: Sun Oct 02 15:10:44 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 22.
test case version: v3,05jul24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 2.
0(0:7)3  4(8:15)7

Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.10: 05-10-02
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc22
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
```

Text 2.6.3: console log for script tc22_postsim.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	2.19e-04	-1.13e-04	0.0741	0.0775	0.0118	0.0116	0.83	0.82	6	
1	2	-1.67e-04	2.67e-04	0.0833	0.0794	0.0122	0.0120	0.86	0.85	8	
2	1	-2.24e-04	2.85e-04	0.1127	0.0997	0.0165	0.0164	0.82	0.82	5	
2	2	4.82e-04	3.49e-04	0.1183	0.1126	0.0161	0.0175	0.81	0.87	7	

Table 2.6.5: test case 22; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-1.22e-04	-3.78e-04	0.3380	0.3870	0.0553	0.0595	1.00	1.02	331
2	1	-2.02e-03	-2.00e-04	0.4514	0.5900	0.0759	0.0801	0.99	1.01	null

Table 2.6.6: test case 22; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	7.94e-06	-5.80e-07	0.0052	0.0046	4.39e-04	4.38e-04	1.0	1.0	0	
1	2	1.37e-05	1.38e-07	0.0043	0.0039	4.54e-04	4.62e-04	1.1	1.1	0	
2	1	-1.31e-05	-1.06e-05	0.0088	0.0069	6.65e-04	5.70e-04	1.1	1.0	0	
2	2	4.42e-05	3.63e-05	0.0095	0.0083	6.48e-04	6.12e-04	1.1	1.0	0	

Table 2.6.7: test case 22; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

tream	*applied dump*	*bias:!OVR*	*dump-bias:!OVR*	*bias:OVR*	*dump-bias:OVR*
0	24000	23724	276	0	n-a
4	24000	23724	276	0	n-a

Table 2.6.8: test case 22; applied dump interval and bias.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000001  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=pri
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11101110  XSTATUS: X-SE_CLK: 11101110
01110111  YSTATUS: Y-SE_CLK: 01110111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.6.4: content of file mcb_reg22.txt.

2.7 input configuration switches

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

2.7.1 organization

This section is divided into subsections according to test case. Each subsection, with exception of the last, contains (in approximately the following order):

(1) a post simulation synopsis;

(2) optional notes, if any, that detail outstanding features or conditions of the post simulation result;

(3) a *post simulation log* that documents script execution;

(4) a table of statistical parameters describing the arithmetic difference between the RTL simulation and behavioural model lag coefficients;

(5) with the exception of single dump test cases, a table of statistical parameters describing the arithmetic difference, between adjacent dumps, of the RTL simulation and behavioural model lag coefficients;

(6) a table of statistical parameters describing the arithmetic difference between the RTL simulation and behavioural model frequency domain coefficients;

(7) a table of bias (expressed in number of samples) accumulated by cells associated with each stream;

(8) any miscellaneous test case specific tables;

(9) frequency domain plots of the RTL simulation result;

(10) content of the MCB register file that is provided by the test bench.

The nomenclature for the table headings are described in 2.2 and 2.5.1. The tables check agreement with the behavioural model; spectral line counts in the frequency domain plots increase confidence in the validation of data stream switch operation. Test cases 17,18,19,20 are common to the sections on data capture (2.17) and timestamp capture (2.16). Tables produced by the post simulation scripts for these test cases will be presented in their respective sections. The final subsection, 2.7.8, presents the result of a supplemental experiment using the configuration of test case 20.

Note: the term *functional model* is used in place of RTL model in the titles of plots for test cases 17, 18, 19, 20, a result of incorporating early versions of code into these scripts.

2.7.2 test case 15

Synopsis: No extraordinary table entries are noted and the single advisory reported in the captured data and register content log is expected. All 3 chains that generated lags and share a common input data stream passed a bit exact comparison between chains (logged in Text 2.7.1).

Notes: The Y input to the first chain is a *no-data* connection and does not generate lags, hence, the advisory reported by *check_register_capture* is expected.

```
***--------------------------------------------------------------------------***
tc15_postsim.m : V2.00 run time: Tue Sep 20 11:20:11 2005
Using postsim_lib: 1.40: 05-08-18

12 frames read for test case 15.
test case version: v3,05apr22
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
7(0:3)- 1(4:7)1 1(8:11)1 1(12:15)1


Initial (single dump) timestamp (word 0): 222333444

ADVISORIES encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc15
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***

octave:11> tc15_similarity_check

No differences exceeding the floating point precision limit of this
        machine were found between similar chains of test case 15.
```

Text 2.7.1 console log for scripts tc15_postsim, tc15_similarity_check.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

ADVISORY from check_register_capture: Chain of
index 1 is not dumped. Skipping check of
captured data for this chain.

Check OK: Captured X,Y ID data (SID, SBID, BBID) are set
          to expected values.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                              template of expected states.
```

Text 2.7.2 test case 15; check register and captured data log; ADVISORY noted.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
2	1	-5.78e-04	-6.54e-04	0.1534	0.1875	0.0284	0.0295	1.30	1.35	3	
3	1	-5.78e-04	-6.54e-04	0.1534	0.1875	0.0284	0.0295	1.30	1.35	3	
4	1	-5.78e-04	-6.54e-04	0.1534	0.1875	0.0284	0.0295	1.30	1.35	3	

Table 2.7.1 test case 15; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
2	1	-1.56e-04	-7.31e-05	0.0182	0.0104	1.77e-03	1.38e-03	2.7	2.1	0	
3	1	-1.56e-04	-7.31e-05	0.0182	0.0104	1.77e-03	1.38e-03	2.7	2.1	0	
4	1	-1.56e-04	-7.31e-05	0.0182	0.0104	1.77e-03	1.38e-03	2.7	2.1	0	

Table 2.7.2 test case 15; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000010  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=sec
01000000  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=adj
00000010  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=sec
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
01000000  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

11111111  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=sec YSw0=sec XSw1=sec XSw0=sec
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.7.3 content of file mcb_reg15.txt.

2.7.3 test case 16

Synopsis: No extraordinary table entries are noted and no warnings or advisories are reported in the captured data and register content log. All 4 chains that share a common input data stream passed a bit exact comparison between chains (logged in Text 2.7.4). The normalized standard deviation in lag and frequency domain statistics tables (Table 2.7.3 and Table 2.7.4) compare closely with their respective counterparts of test case 15 (Table 2.7.1 and Table 2.7.2) thereby further validating operation of the shift enable clock (**SE_CLK**).

Notes: n-a

```
***--------------------------------------------------------------------------***
tc16_postsim.m : V2.00 run time: Tue Sep 20 11:33:43 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 16.
test case version: v3,05apr22
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Input configuration from MCSR regsister settings:
List chains from 1 to 4.
1(0:3)1 1(4:7)1 1(8:11)1 1(12:15)1


Initial (single dump) timestamp (word 0): 222333444

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc16
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***

octave:17> tc16_similarity_check

No differences exceeding the floating point precision limit of this
        machine were found between similar chains of test case 16.
```

Text 2.7.4 console log for scripts tc16_postsim, tc16_similarity_check.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	-1.26e-04	-1.25e-03	0.2492	0.2419	0.0392	0.0428	1.27	1.38	1	
2	1	-1.26e-04	-1.25e-03	0.2492	0.2419	0.0392	0.0428	1.27	1.38	1	
3	1	-1.26e-04	-1.25e-03	0.2492	0.2419	0.0392	0.0428	1.27	1.38	1	
4	1	-1.26e-04	-1.25e-03	0.2492	0.2419	0.0392	0.0428	1.27	1.38	1	

Table 2.7.3 test case 16; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	5.69e-04	-1.82e-04	0.0312	0.0180	2.46e-03	1.80e-03	2.6	1.9	1	
2	1	5.69e-04	-1.82e-04	0.0312	0.0180	2.46e-03	1.80e-03	2.6	1.9	1	
3	1	5.69e-04	-1.82e-04	0.0312	0.0180	2.46e-03	1.80e-03	2.6	1.9	1	
4	1	5.69e-04	-1.82e-04	0.0312	0.0180	2.46e-03	1.80e-03	2.6	1.9	1	

Table 2.7.4 test case 16; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000010  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=sec
01000000  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=adj
00000010  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=sec
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
01000000  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.7.5 content of file mcb_reg16.txt.

2.7.4 test case 17

Synopsis: No extraordinary table entries are noted and no warnings or advisories are reported in the captured data and register content log. Continuum correlation is seen only in chain 1 and two dominant spectral lines are present in the plots of Illustration 2.7.1 to Illustration 2.7.8 as expected.

Notes: Tables of captured times stamps and ID data are found in (2.16.2 and 2.17.2).

```
***--------------------------------------------------------------------------***
tc17_postsim.m : V2.00 run time: Tue Sep 20 12:03:48 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 17.
test case version: v3,05jul27
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
1(0:3)1 1(4:7)3 1(8:11)5 1(12:15)7


Initial (dump= 1) timestamp (word 0): 2863311530
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Writing frequency domain plot files.
Writing data ID capture tables.
Writing timestamp capture tables.
***--------------------------------------------------------------------------***

octave:21> mcb17.MCSR.XYDS
ans = 1
octave:22> tc17_make_fb_stat_tables

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc17
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.
```

Text 2.7.6 console log for scripts tc17_postsim, tc17_make_fb_stat_tables.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	-1.03e-03	-7.44e-05	0.1382	0.1413	0.0233	0.0211	1.17	1.05	4	
1	2	1.27e-04	6.25e-04	0.1390	0.1620	0.0240	0.0242	1.20	1.21	5	
2	1	2.88e-04	3.39e-05	0.1384	0.1446	0.0234	0.0240	1.17	1.20	1	
2	2	-6.85e-04	-3.66e-04	0.1408	0.1493	0.0247	0.0235	1.24	1.18	2	
3	1	-1.04e-04	-8.57e-05	0.1272	0.1571	0.0221	0.0229	1.11	1.14	4	
3	2	7.35e-04	4.88e-05	0.1380	0.1449	0.0245	0.0245	1.23	1.22	1	
4	1	1.03e-05	2.33e-04	0.1335	0.1381	0.0218	0.0223	1.09	1.12	3	
4	2	-5.70e-05	-3.16e-04	0.1416	0.1222	0.0242	0.0211	1.21	1.06	0	

Table 2.7.5 test case 17; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-3.61e-03	2.94e-03	0.5274	0.4959	0.0809	0.0830	1.04	0.96	null
2	1	-1.75e-03	3.13e-05	0.4972	0.5655	0.0868	0.0914	0.96	0.94	null
3	1	1.60e-03	1.48e-04	0.5010	0.4869	0.0862	0.0834	0.99	1.00	null
4	1	1.99e-04	-1.22e-03	0.5622	0.4896	0.0825	0.0826	1.03	1.00	null

Table 2.7.6 test case 17; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	3.42e-05	-1.15e-04	0.0104	0.0132	1.28e-03	1.20e-03	2.1	2.0	0	
1	2	-7.82e-05	-1.03e-04	0.0144	0.0088	1.49e-03	1.13e-03	2.5	1.9	0	
2	1	6.14e-05	-3.02e-05	0.0121	0.0135	1.31e-03	1.23e-03	2.2	2.1	0	
2	2	-1.07e-05	1.66e-04	0.0122	0.0131	1.27e-03	1.27e-03	2.1	2.1	0	
3	1	6.67e-05	-8.14e-05	0.0111	0.0152	1.14e-03	1.28e-03	1.9	2.1	0	
3	2	-1.36e-04	2.10e-04	0.0181	0.0098	1.44e-03	1.26e-03	2.4	2.1	0	
4	1	-1.47e-05	4.90e-05	0.0118	0.0095	1.26e-03	1.14e-03	2.1	1.9	0	
4	2	2.40e-05	-2.49e-05	0.0139	0.0093	1.30e-03	1.12e-03	2.2	1.9	0	

Table 2.7.7 test case 17; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

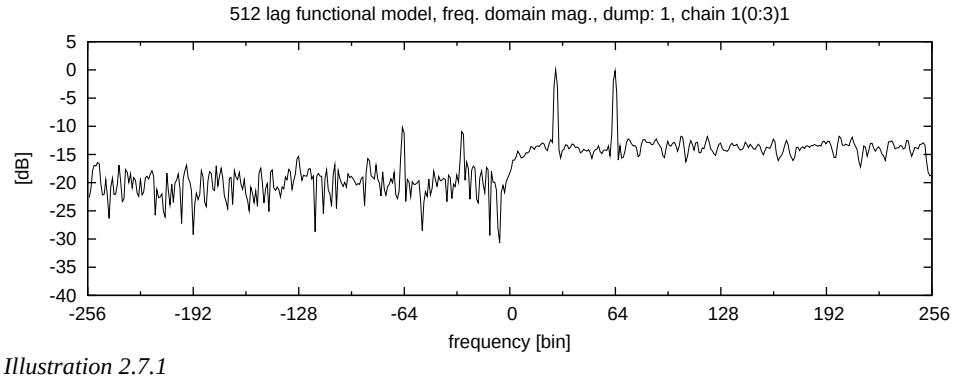

Illustration 2.7.1

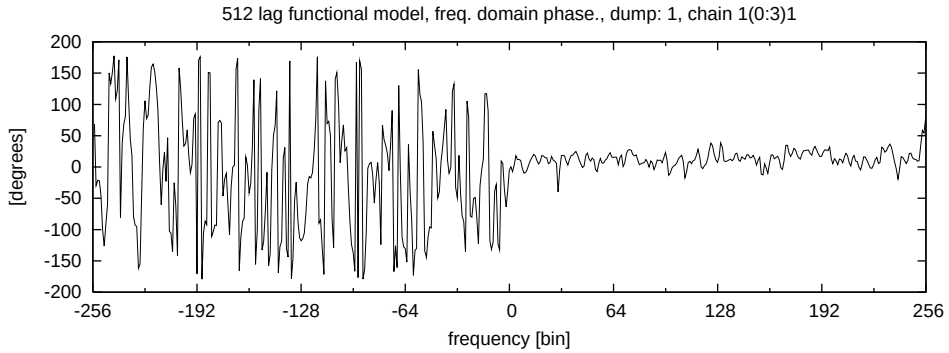

Illustration 2.7.2

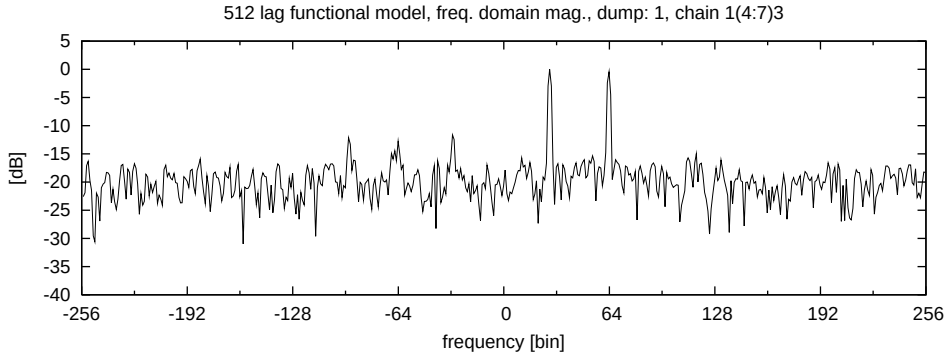

Illustration 2.7.3

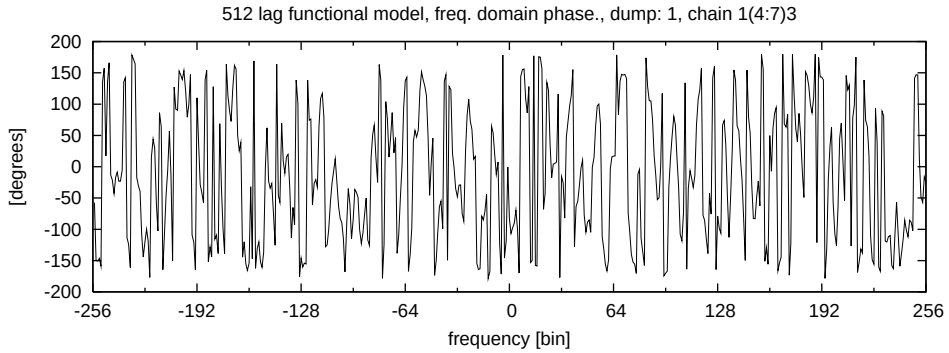

Illustration 2.7.4

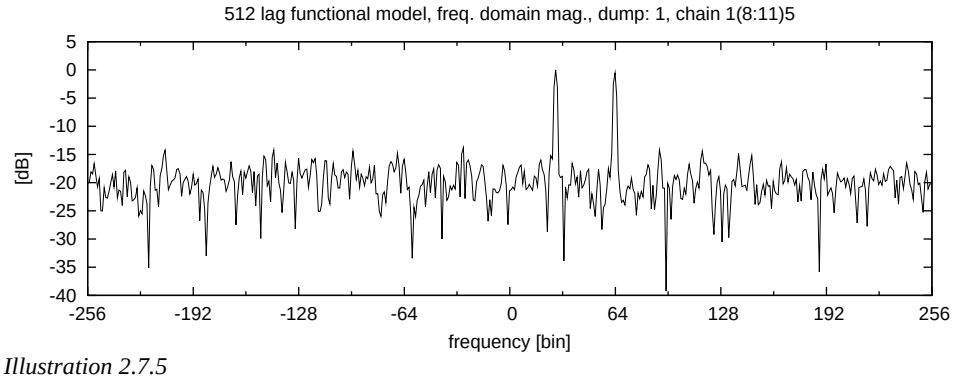

Illustration 2.7.5

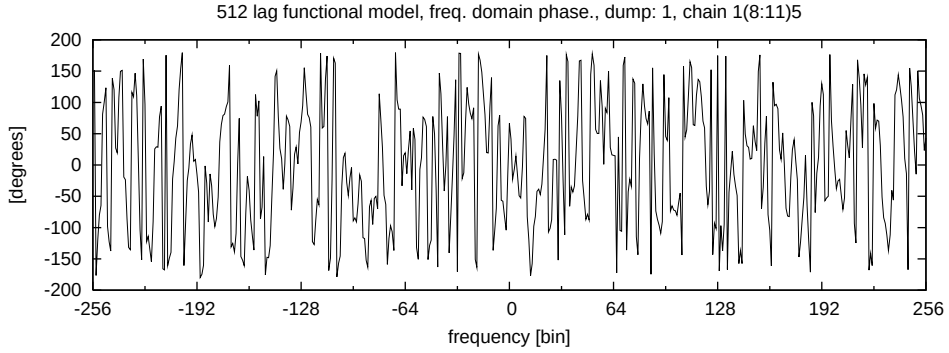

Illustration 2.7.6

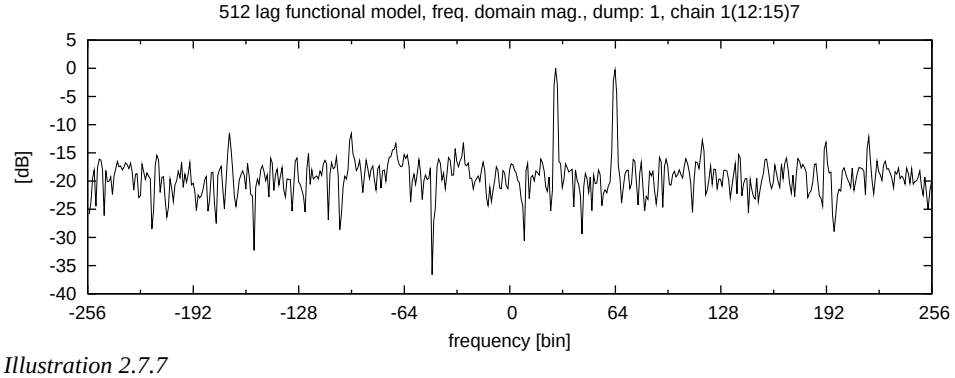

Illustration 2.7.7

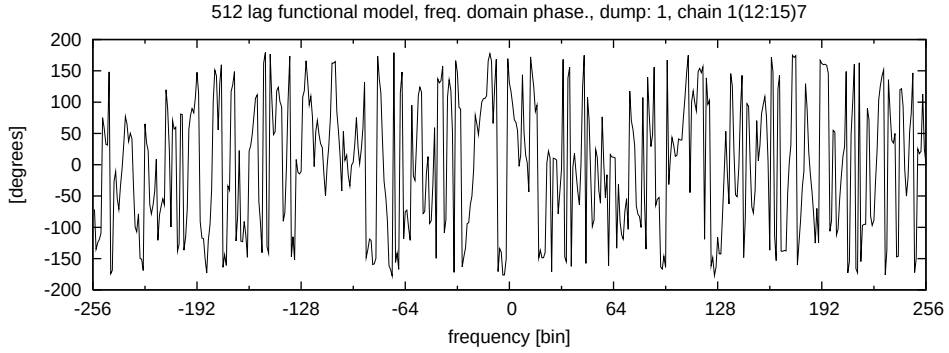

Illustration 2.7.8

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000010  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=sec
01000000  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=adj
00000010  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=sec
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
01000000  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

01100000  CCQR_0_1: Q1: YSw1=pri YSw0=sec XSw1=sec XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00110011  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=sec XSw0=sec  Q2: YSw1=pri YSw0=pri XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111101  XSTATUS: X-SE_CLK: 11111101
01010101  YSTATUS: Y-SE_CLK: 01010101
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.7.7 content of file mcb_reg17.txt.

2.7.5 test case 18

Synopsis: No extraordinary table entries are noted and no warnings or advisories are reported in the captured data and register content log. Continuum correlation is seen only in chain 1 and two dominant spectral lines are present in the plots of Illustration 2.7.9 to Illustration 2.7.16 as expected.

Notes: Tables of captured times stamps and ID data are found in (2.16.3 and 2.17.3).

```
**------------------------------------------------------------------------***
tc18_postsim.m : V2.00 run time: Tue Sep 20 12:16:15 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 18.
test case version: v3,05jul27
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
1(0:3)1 3(4:7)1 5(8:11)1 7(12:15)1


Initial (dump= 1) timestamp (word 0): 1431655765
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Writing frequency domain plot files.
Writing data ID capture tables.
Writing timestamp capture tables.
***------------------------------------------------------------------------***

octave:30> mcb18.MCSR.XYDS
ans = 1
octave:31> tc18_make_fb_stat_tables

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc18
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.
```

Text 2.7.8 console log for scripts tc18_postsim, tc18_make_fb_stat_tables.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	-1.03e-03	-7.44e-05	0.1382	0.1413	0.0233	0.0211	1.17	1.05	4	
1	2	1.27e-04	6.25e-04	0.1390	0.1620	0.0240	0.0242	1.20	1.21	5	
2	1	1.32e-03	1.13e-05	0.1549	0.1329	0.0231	0.0220	1.16	1.10	4	
2	2	-7.44e-04	2.10e-05	0.1671	0.1583	0.0269	0.0259	1.34	1.29	3	
3	1	6.36e-04	2.70e-04	0.1568	0.1588	0.0247	0.0252	1.23	1.26	2	
3	2	-6.38e-04	-5.99e-04	0.1766	0.1546	0.0264	0.0273	1.32	1.37	2	
4	1	1.52e-04	7.87e-04	0.1711	0.1945	0.0306	0.0326	1.53	1.63	1	
4	2	-7.14e-04	3.14e-04	0.1984	0.1921	0.0329	0.0337	1.64	1.69	1	

Table 2.7.8 test case 18; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-3.61e-03	2.94e-03	0.5274	0.4959	0.0809	0.0830	1.04	0.96	null
2	1	-6.00e-03	-5.91e-05	0.4379	0.4345	0.0794	0.0765	1.06	0.98	null
3	1	3.35e-03	-2.20e-03	0.5410	0.4700	0.0781	0.0786	0.96	0.95	null
4	1	-8.28e-04	2.96e-03	0.4396	0.4027	0.0799	0.0715	1.01	0.92	null

Table 2.7.9 test case 18; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	3.42e-05	-1.15e-04	0.0104	0.0132	1.28e-03	1.20e-03	2.1	2.0	0	
1	2	-7.82e-05	-1.03e-04	0.0144	0.0088	1.49e-03	1.13e-03	2.5	1.9	0	
2	1	1.24e-04	3.03e-06	0.0121	0.0094	1.19e-03	1.15e-03	2.0	1.9	0	
2	2	-1.03e-05	-3.30e-05	0.0155	0.0154	1.41e-03	1.35e-03	2.4	2.2	0	
3	1	1.81e-04	-4.15e-05	0.0194	0.0143	1.42e-03	1.19e-03	2.4	2.0	1	
3	2	2.50e-05	-5.71e-05	0.0200	0.0102	1.50e-03	1.26e-03	2.5	2.1	0	
4	1	-3.93e-05	7.92e-05	0.0109	0.0301	1.16e-03	2.07e-03	1.9	3.5	1	
4	2	-6.27e-05	7.62e-05	0.0341	0.0149	2.20e-03	1.31e-03	3.7	2.2	1	

Table 2.7.10 test case 18; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

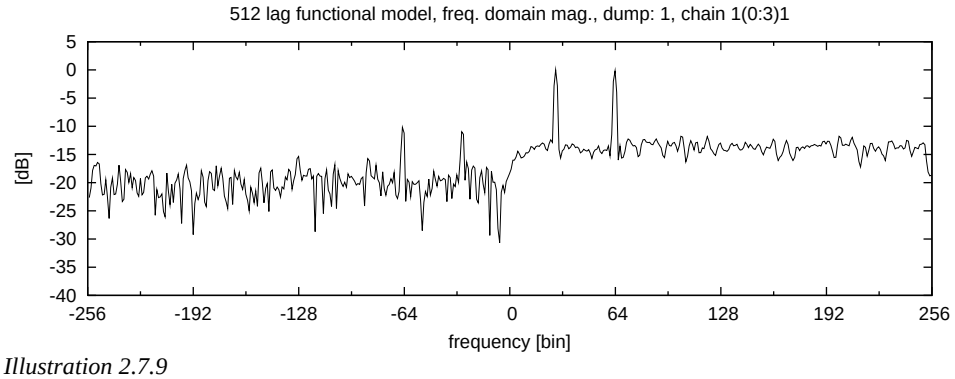

Illustration 2.7.9

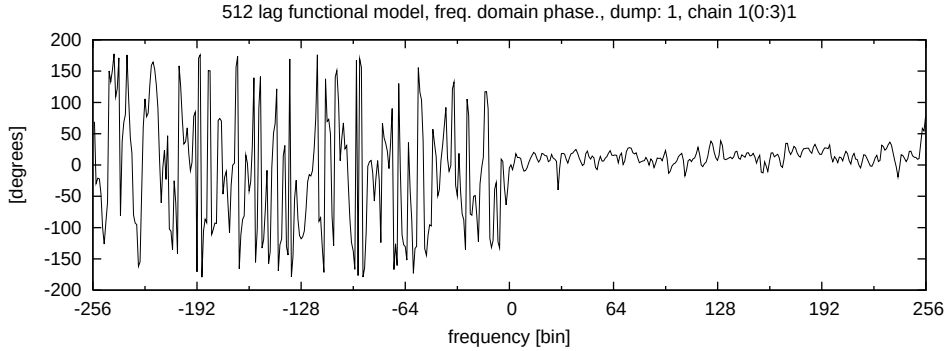

Illustration 2.7.10

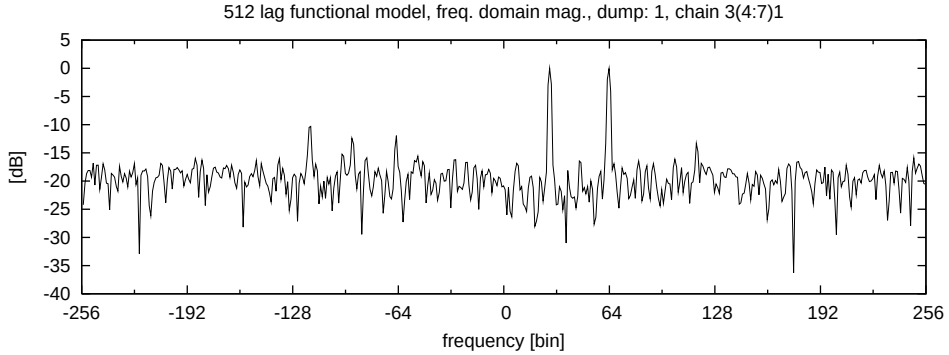

Illustration 2.7.11

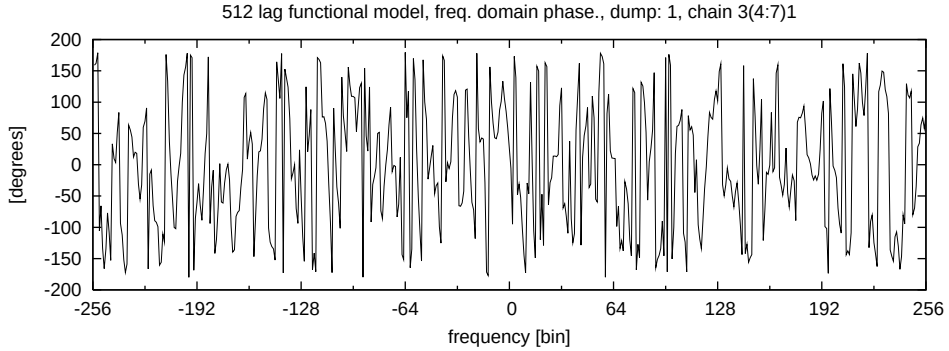

Illustration 2.7.12

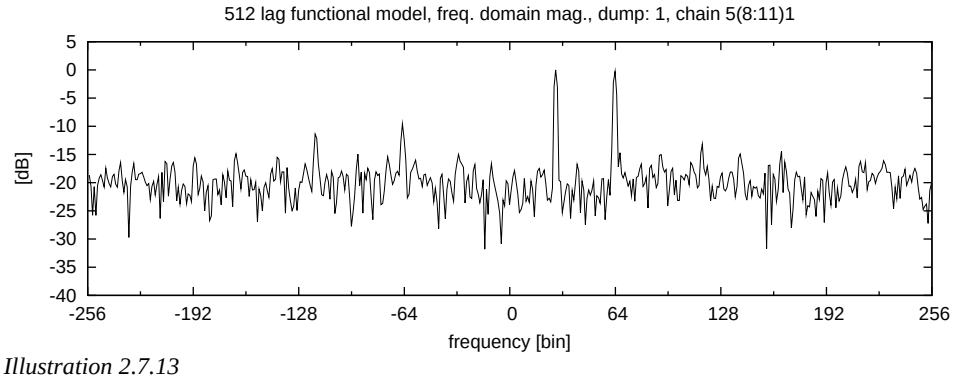

Illustration 2.7.13

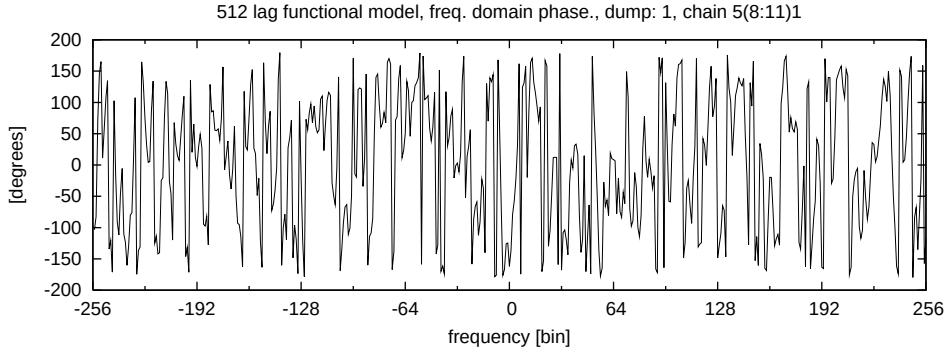

Illustration 2.7.14

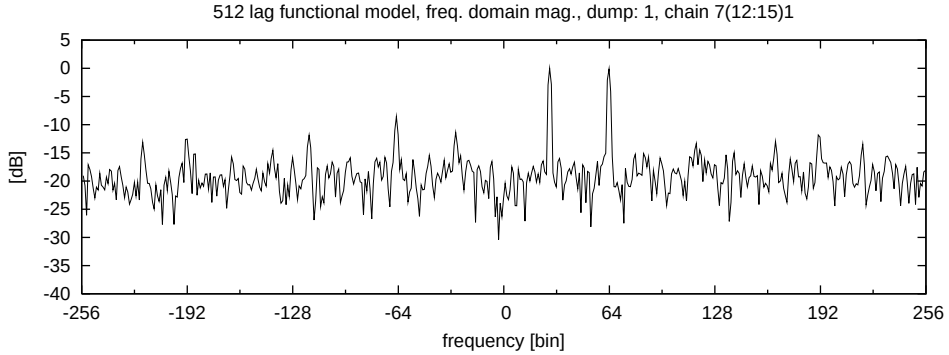

Illustration 2.7.15

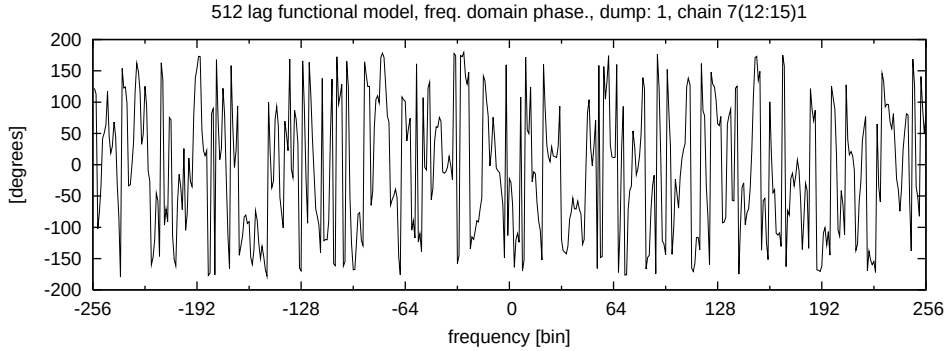

Illustration 2.7.16

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000010  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=sec
01000000  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=adj
00000010  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=sec
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
01000000  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

11000000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11001100  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=sec YSw0=sec XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
01010101  XSTATUS: X-SE_CLK: 01010101
11111101  YSTATUS: Y-SE_CLK: 11111101
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.7.9 content of file mcb_reg18.txt.

2.7.6 test case 19

Synopsis: No extraordinary table entries are noted; no warnings or advisories are reported in the captured data and register content log. Continuum correlation is seen only in chain 1 and a single dominant spectral lines is present in the plots of Illustration 2.7.17 to Illustration 2.7.24 as expected. Plots of Illustration 2.7.33 to Illustration 2.7.64 are included as a comparative reference to respective plots in section 2.7.7.

Notes: Tables of captured times stamps and ID data are found in (2.16.4 and 2.17.4).

```
***--------------------------------------------------------------------------***
tc19_postsim.m : V2.00 run time: Tue Sep 20 12:27:14 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 19.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
0(0:3)0 0(4:7)2 0(8:11)4 0(12:15)6


Initial (dump= 1) timestamp (word 0): 2863311530
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Writing frequency domain plot files.
Writing data ID capture tables.
Writing timestamp capture tables.
***--------------------------------------------------------------------------***

octave:35> mcb19.MCSR.XYDS
ans = 0
octave:36> tc19_make_fb_stat_tables

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc19
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.
```

Text 2.7.10 console log for scripts tc19_postsim, tc19_make_fb_stat_tables.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	1.74e-03	-4.40e-04	0.1795	0.1298	0.0266	0.0224	1.33	1.12	2	
1	2	-7.87e-04	1.61e-04	0.1593	0.1512	0.0269	0.0241	1.35	1.21	3	
2	1	2.28e-04	2.74e-04	0.1383	0.1378	0.0247	0.0236	1.24	1.18	2	
2	2	1.03e-03	3.34e-04	0.1606	0.1311	0.0238	0.0239	1.19	1.19	2	
3	1	1.43e-04	-2.60e-04	0.1339	0.1383	0.0226	0.0239	1.13	1.20	3	
3	2	6.57e-04	1.09e-03	0.1399	0.1440	0.0216	0.0212	1.08	1.06	5	
4	1	8.45e-04	9.22e-05	0.1278	0.1473	0.0241	0.0246	1.20	1.23	1	
4	2	1.10e-03	-3.58e-04	0.1299	0.1555	0.0228	0.0238	1.14	1.19	4	

Table 2.7.11 test case 19; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-3.57e-03	4.05e-03	0.4976	0.5204	0.0912	0.0941	1.07	1.04	256
2	1	2.51e-03	-5.38e-03	0.4811	0.5004	0.0756	0.0765	1.03	0.98	null
3	1	-1.70e-03	5.27e-03	0.4939	0.4625	0.0792	0.0852	1.03	0.96	null
4	1	6.25e-04	-6.45e-04	0.5730	0.4539	0.0867	0.0759	1.01	0.96	null

Table 2.7.12 test case 19; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	3.99e-04	6.36e-05	0.0140	0.0096	1.44e-03	1.13e-03	2.4	1.9	0	
1	2	-1.13e-04	3.10e-04	0.0212	0.0092	1.56e-03	1.16e-03	2.6	1.9	1	
2	1	-8.25e-05	-3.37e-06	0.0108	0.0140	1.28e-03	1.37e-03	2.1	2.3	0	
2	2	1.92e-04	3.04e-05	0.0171	0.0086	1.44e-03	1.16e-03	2.4	1.9	0	
3	1	-1.87e-05	-3.32e-06	0.0154	0.0126	1.26e-03	1.25e-03	2.1	2.1	0	
3	2	5.08e-05	1.30e-04	0.0080	0.0116	1.10e-03	1.19e-03	1.8	2.0	0	
4	1	1.49e-04	4.92e-05	0.0150	0.0128	1.38e-03	1.32e-03	2.3	2.2	0	
4	2	-7.31e-05	7.86e-06	0.0143	0.0089	1.37e-03	1.15e-03	2.3	1.9	0	

Table 2.7.13 test case 19; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

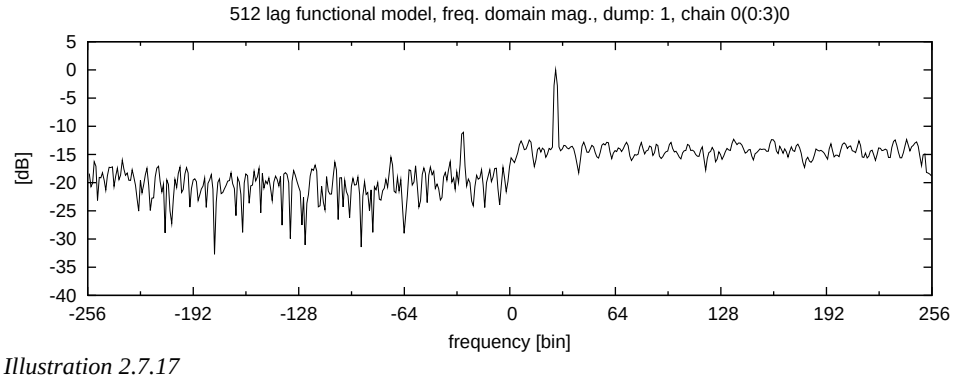

Illustration 2.7.17

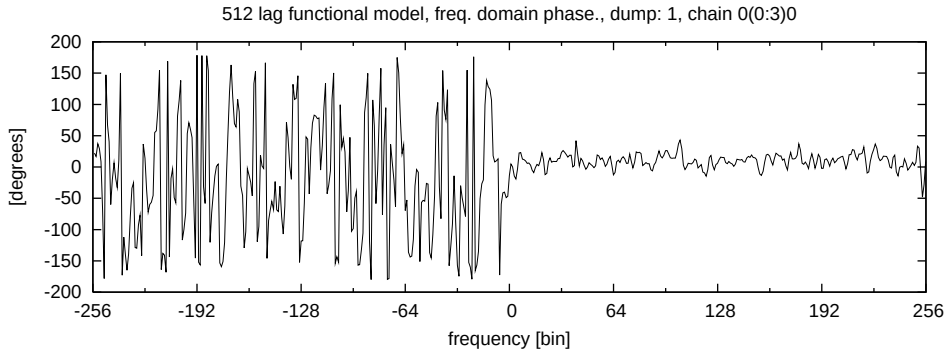

Illustration 2.7.18

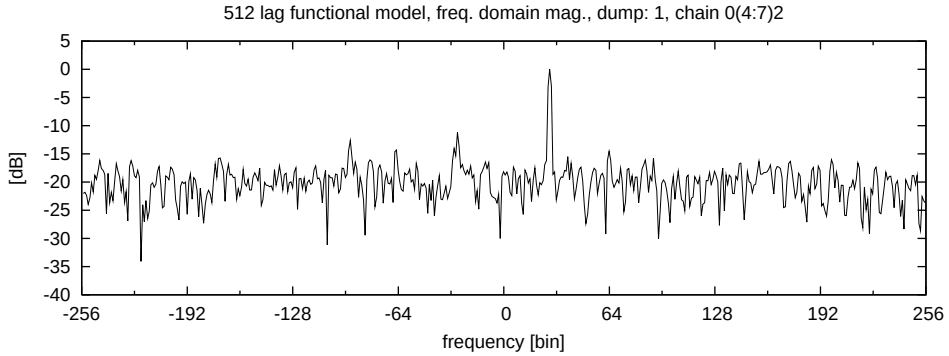

Illustration 2.7.19

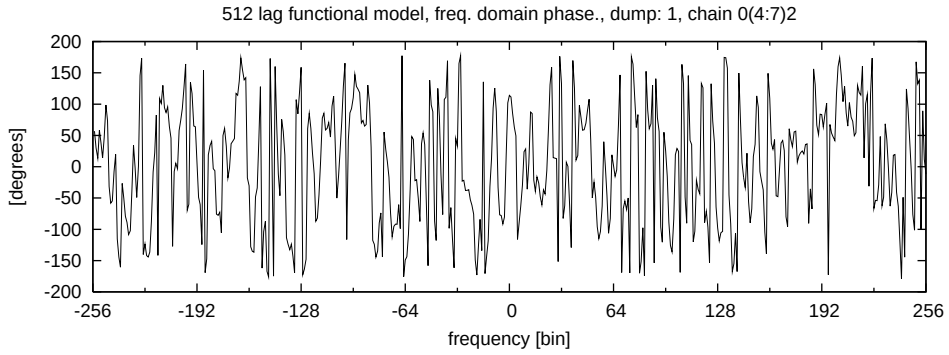

Illustration 2.7.20

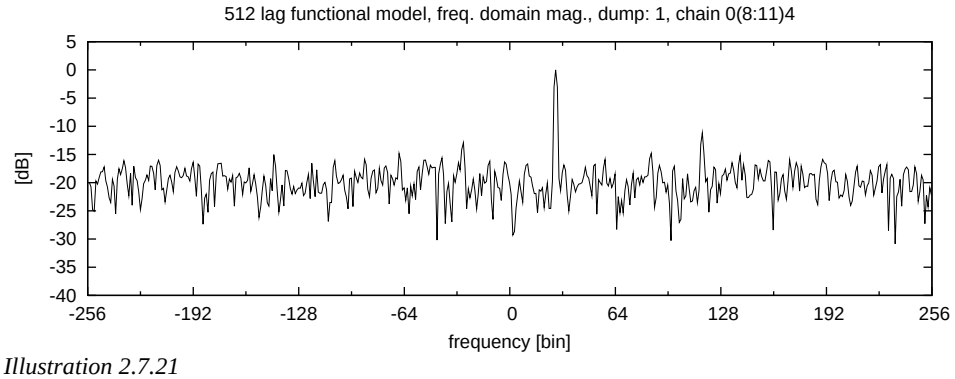

Illustration 2.7.21

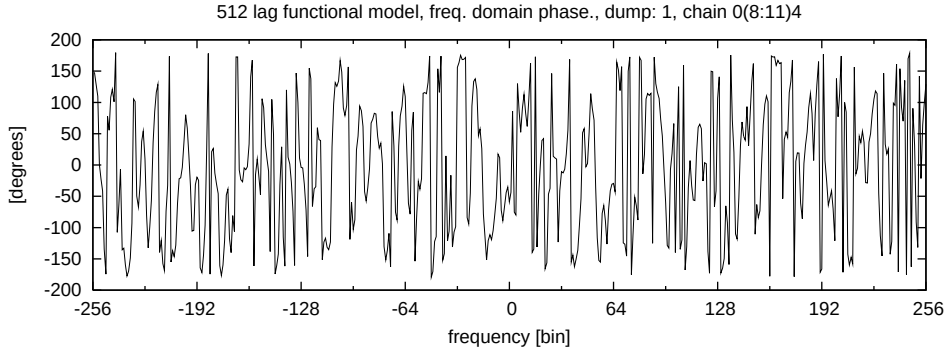

Illustration 2.7.22

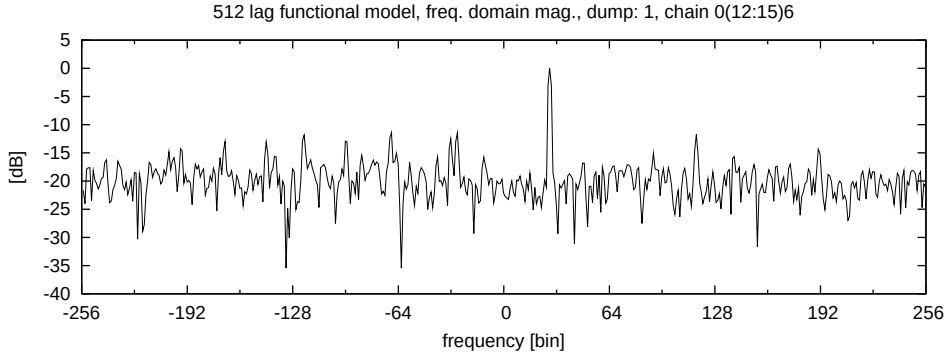

Illustration 2.7.23

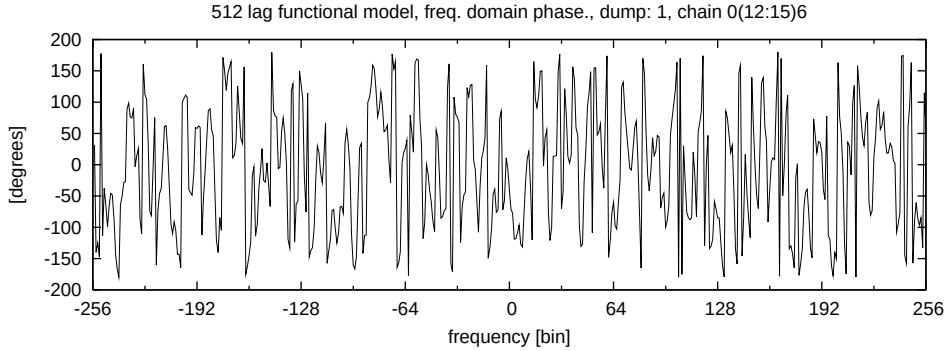

Illustration 2.7.24

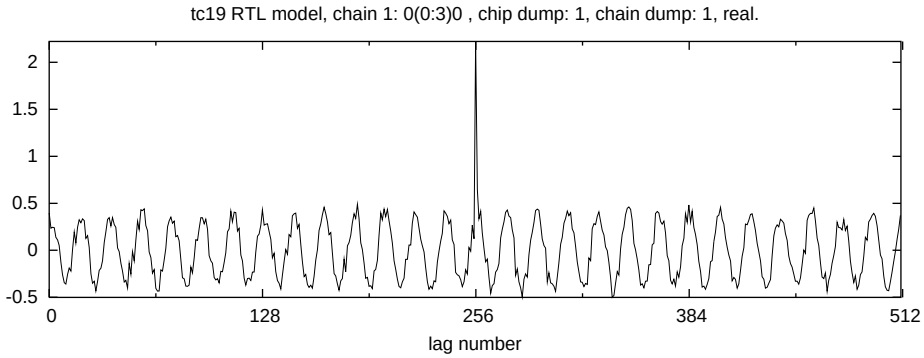

Illustration 2.7.25

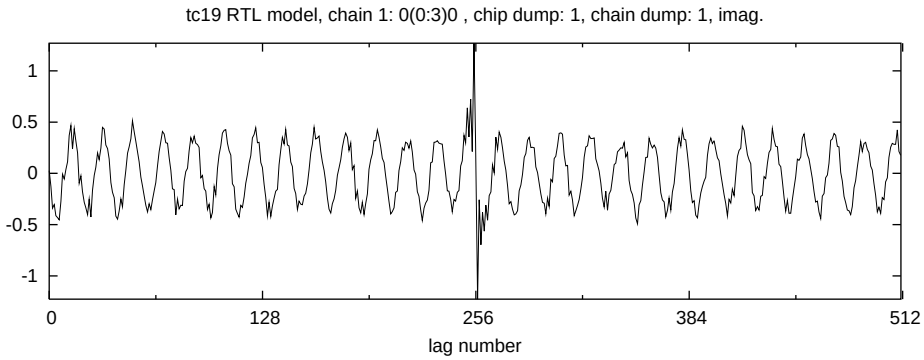

Illustration 2.7.26

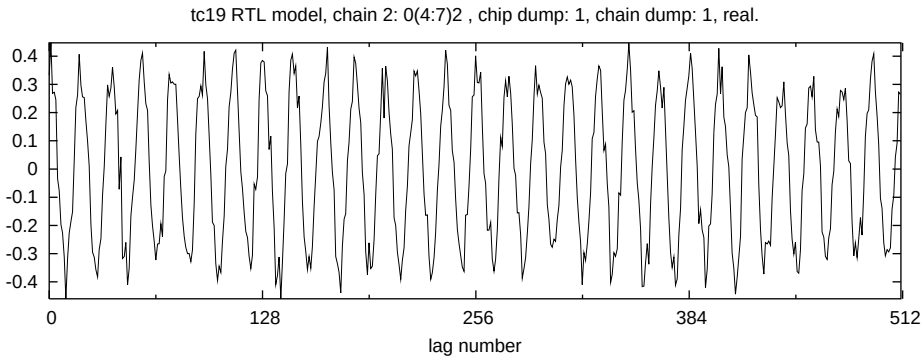

Illustration 2.7.27

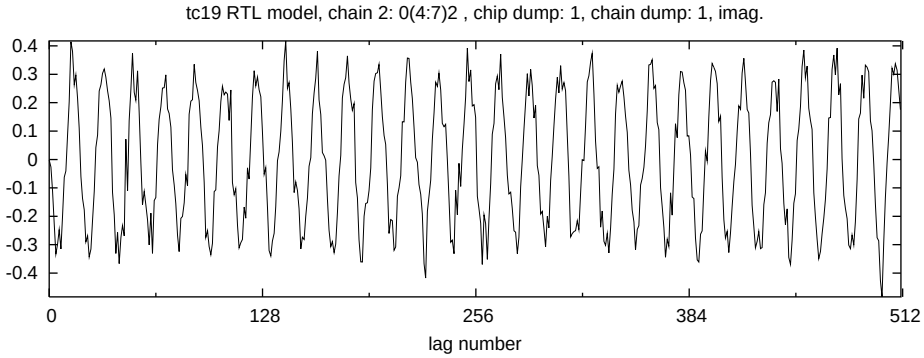

Illustration 2.7.28

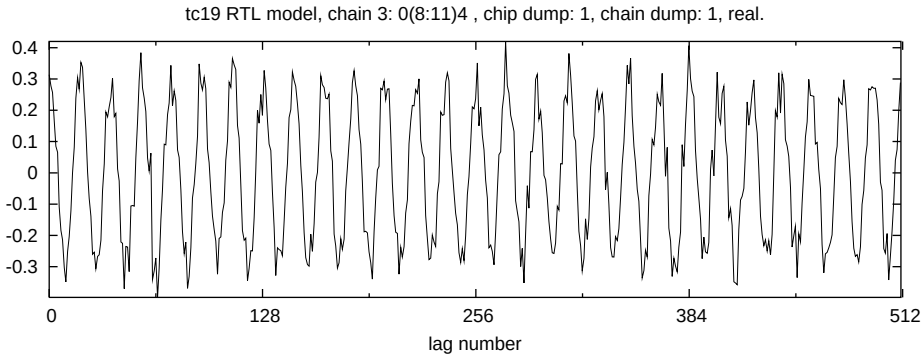

Illustration 2.7.29

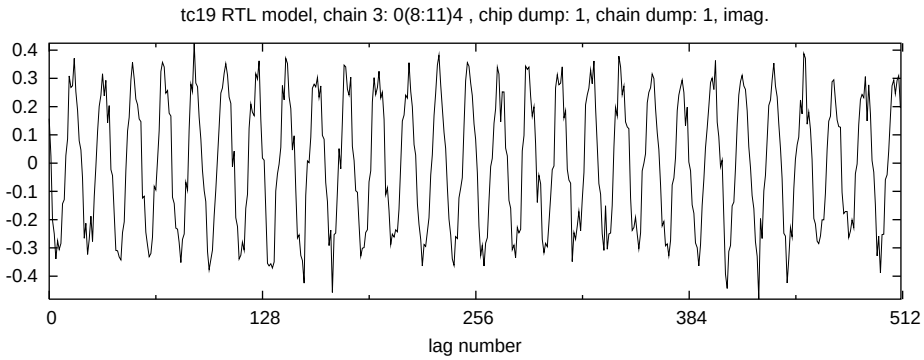

Illustration 2.7.30

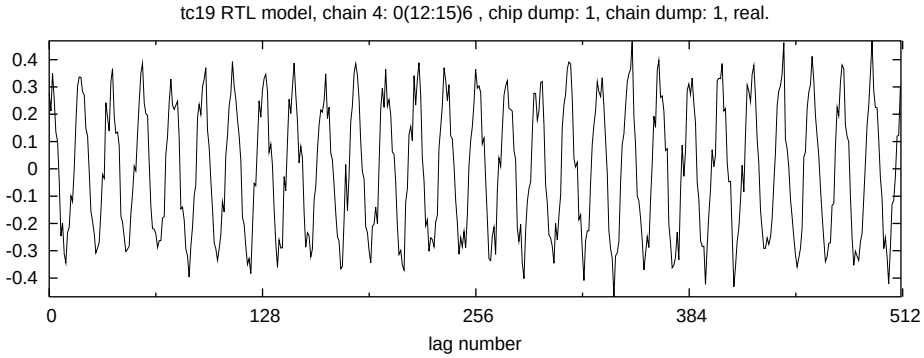

Illustration 2.7.31

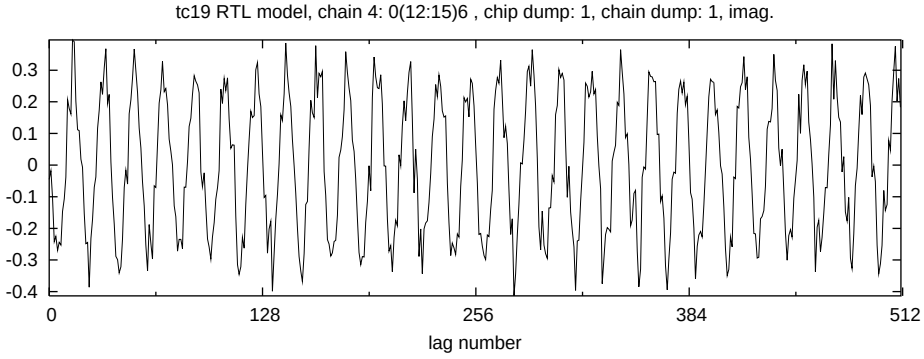

Illustration 2.7.32

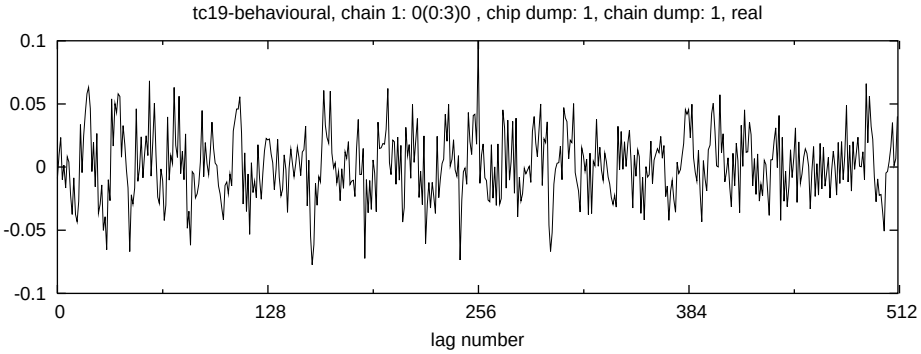

Illustration 2.7.33

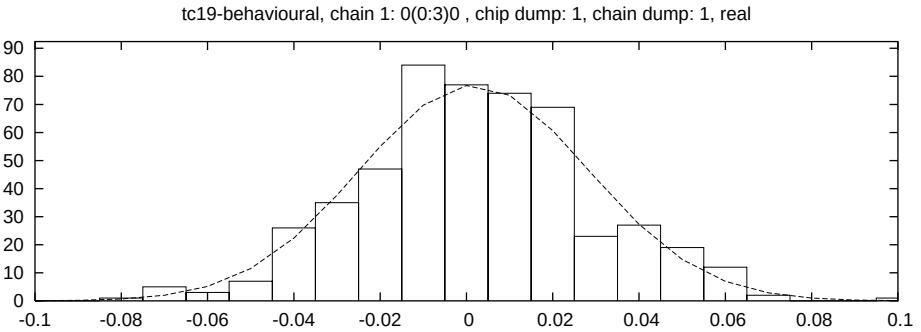

Illustration 2.7.34

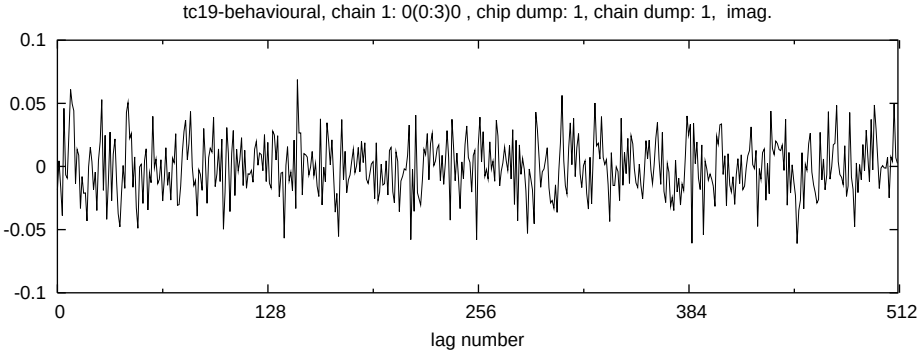

Illustration 2.7.35

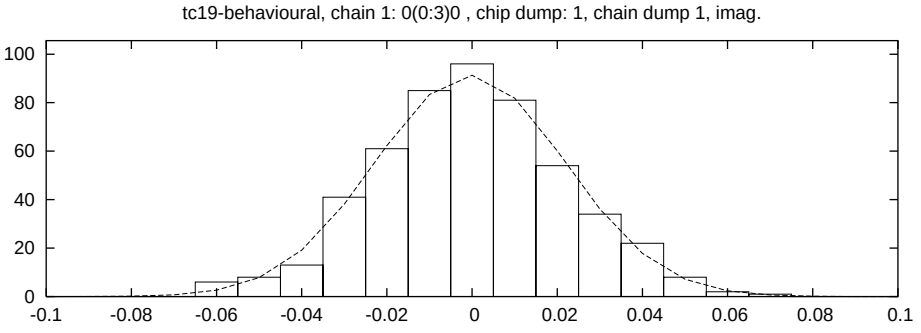

Illustration 2.7.36

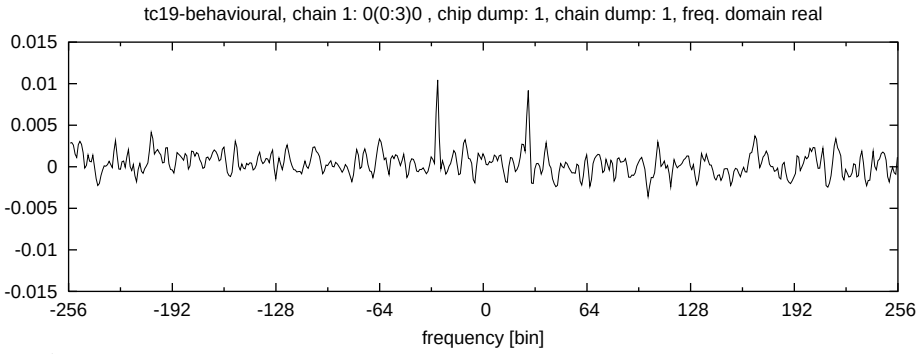

Illustration 2.7.37

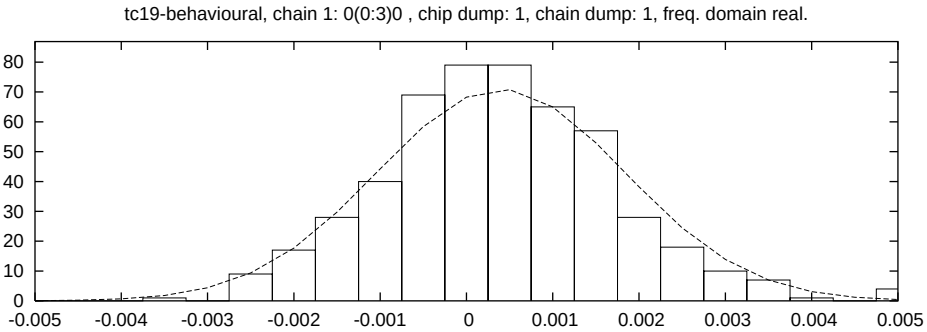

Illustration 2.7.38

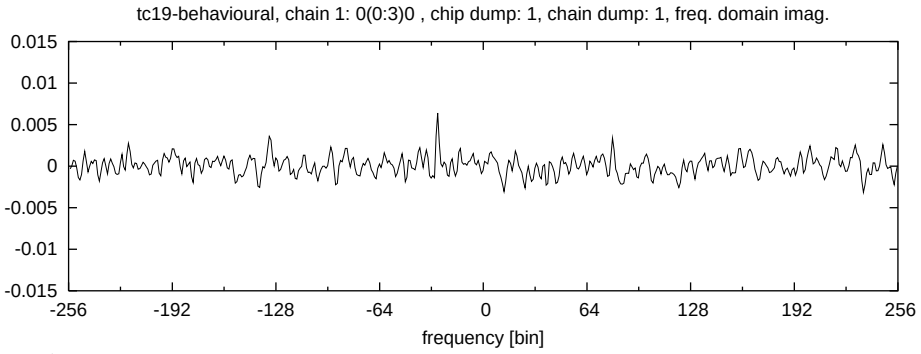

Illustration 2.7.39

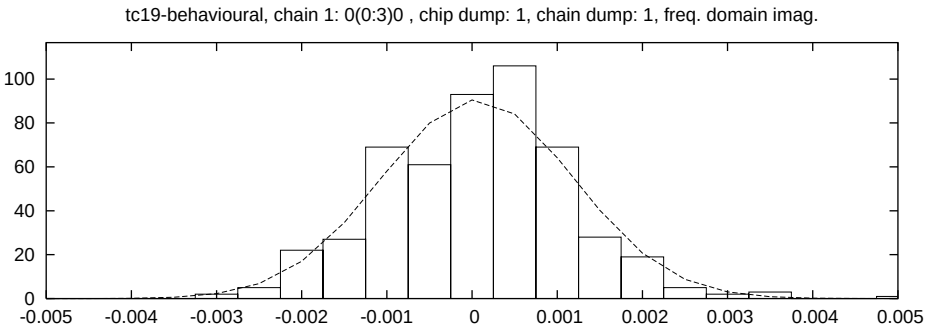

Illustration 2.7.40

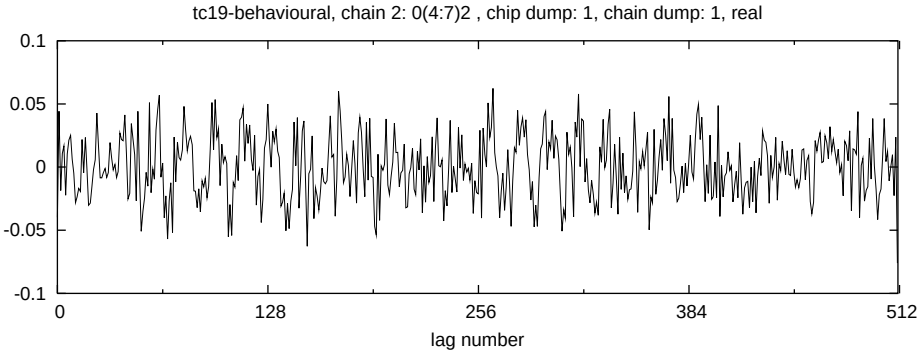

Illustration 2.7.41

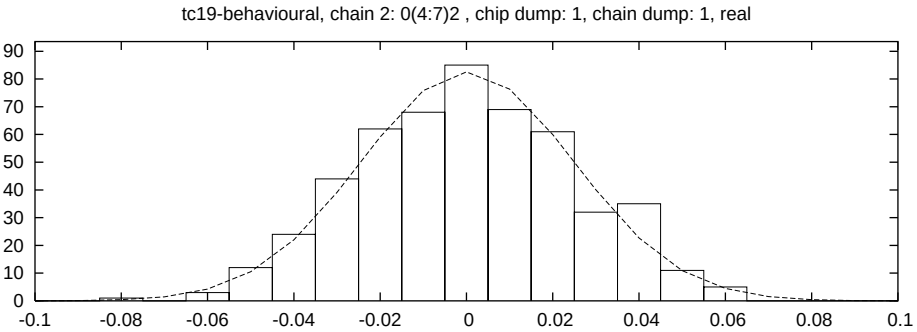

Illustration 2.7.42

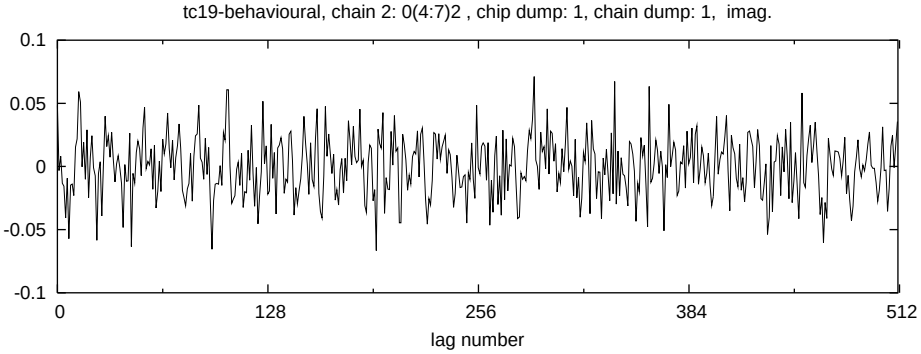

Illustration 2.7.43

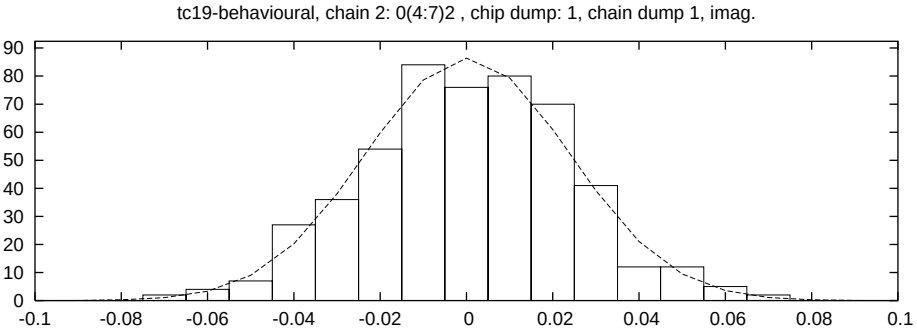

Illustration 2.7.44

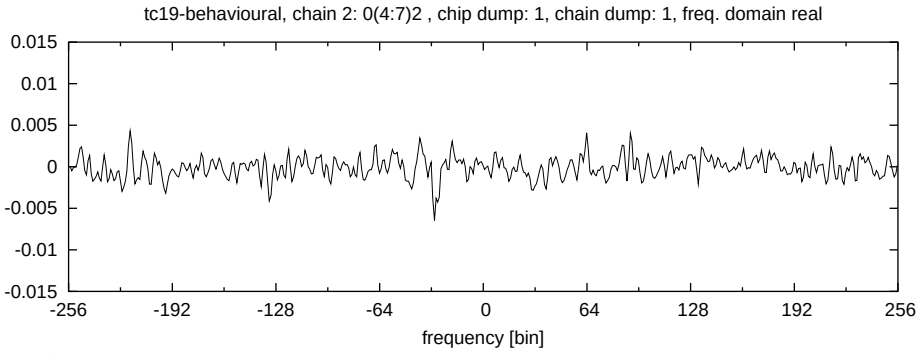

Illustration 2.7.45

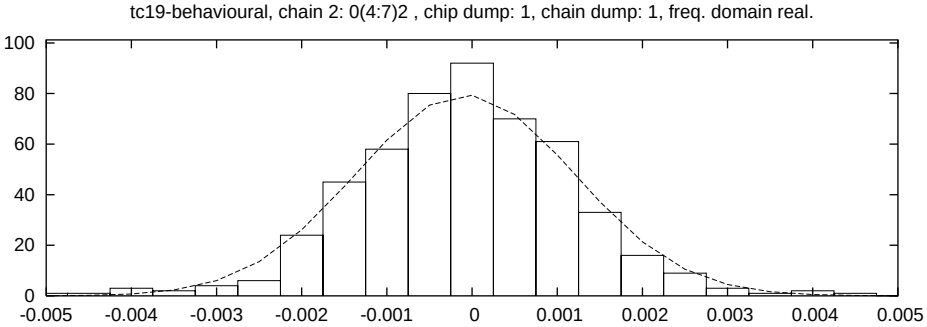

Illustration 2.7.46

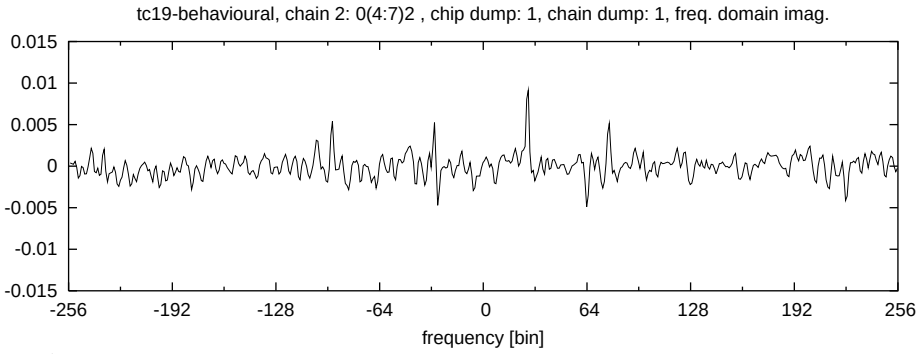

Illustration 2.7.47

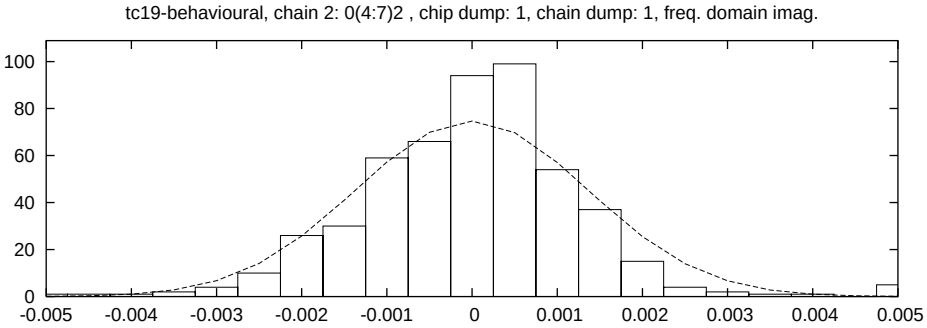

Illustration 2.7.48

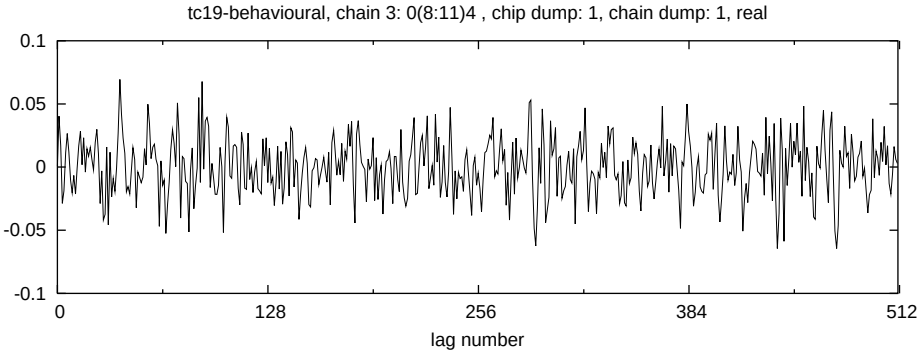

Illustration 2.7.49

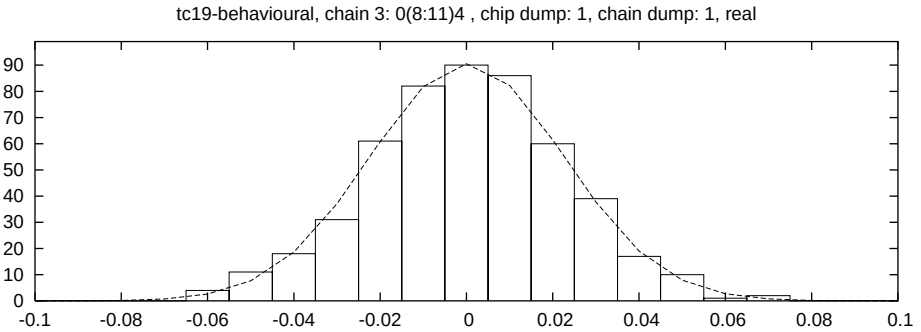

Illustration 2.7.50

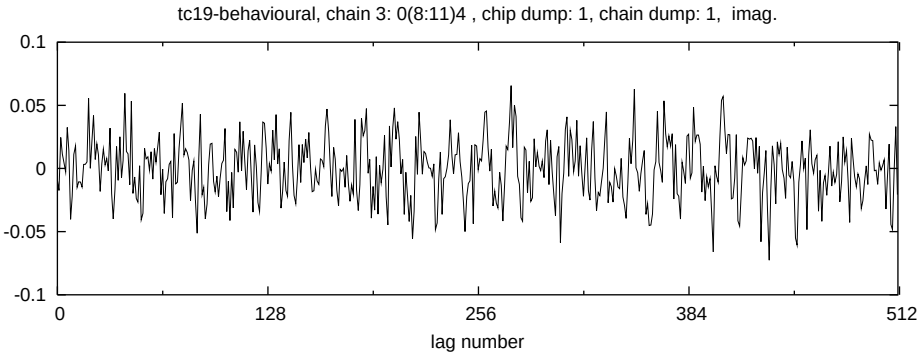

Illustration 2.7.51

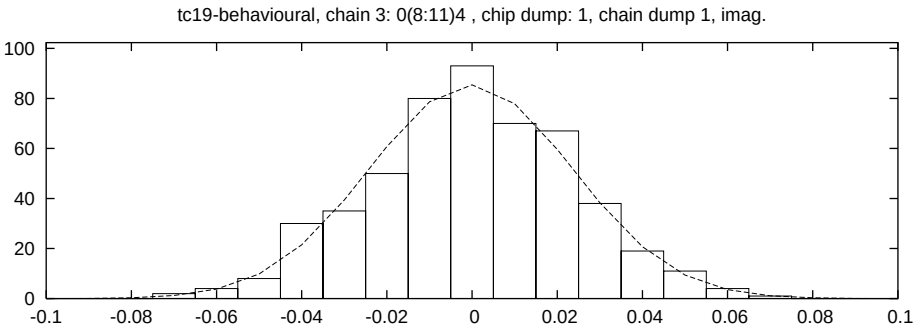

Illustration 2.7.52

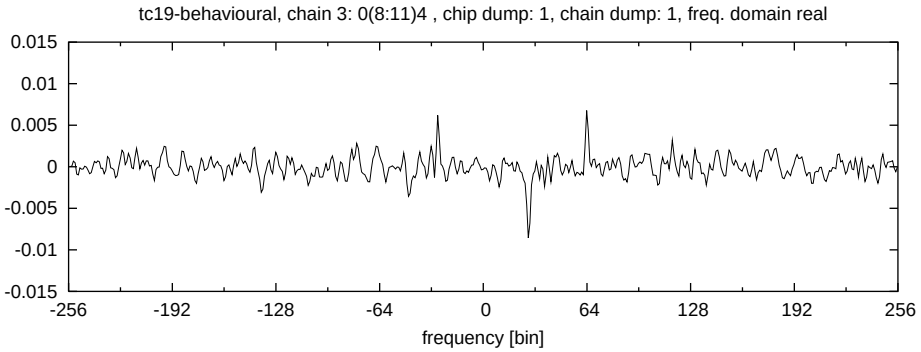

Illustration 2.7.53

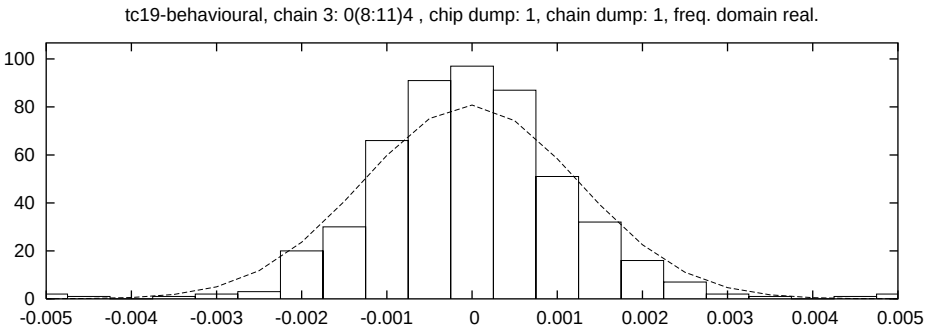

Illustration 2.7.54

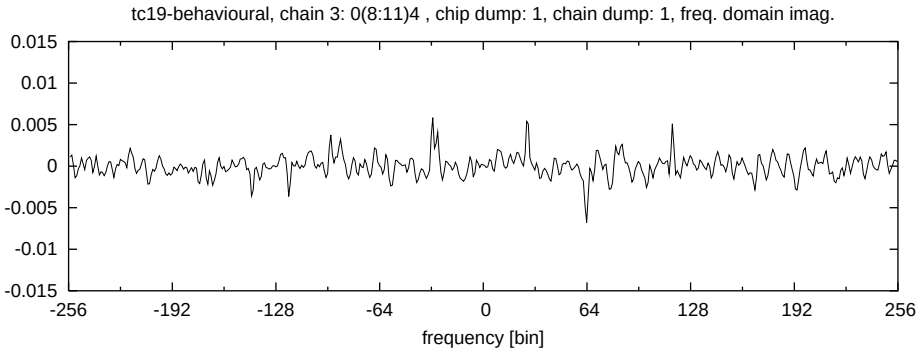

Illustration 2.7.55

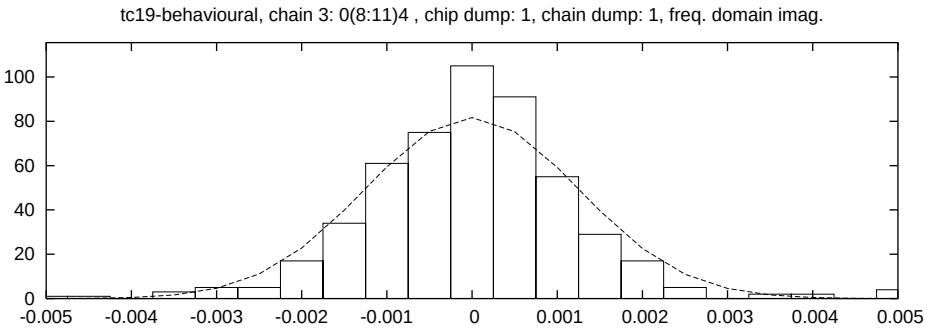

Illustration 2.7.56

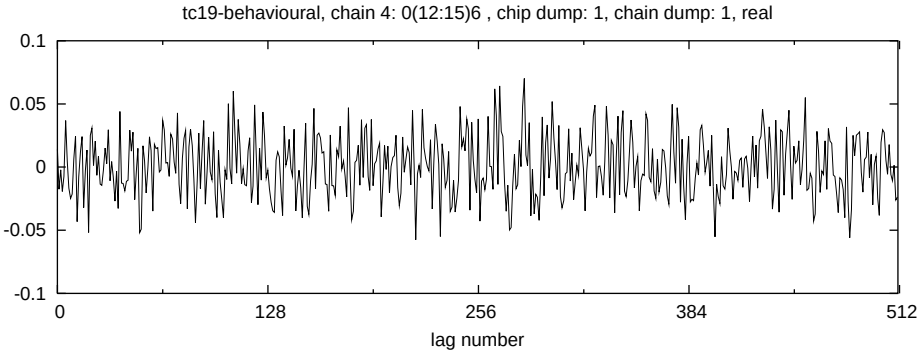

Illustration 2.7.57

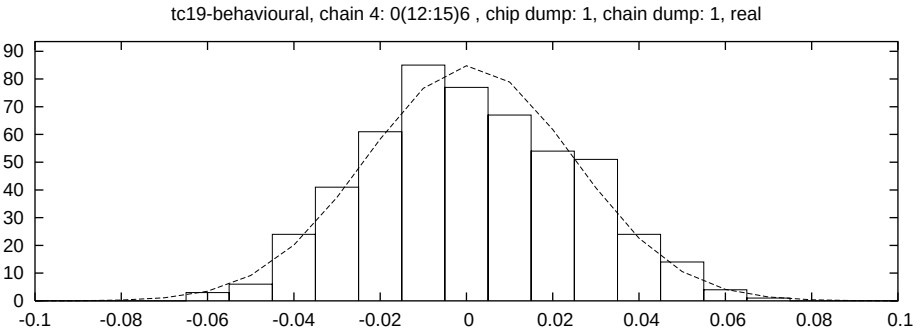

Illustration 2.7.58

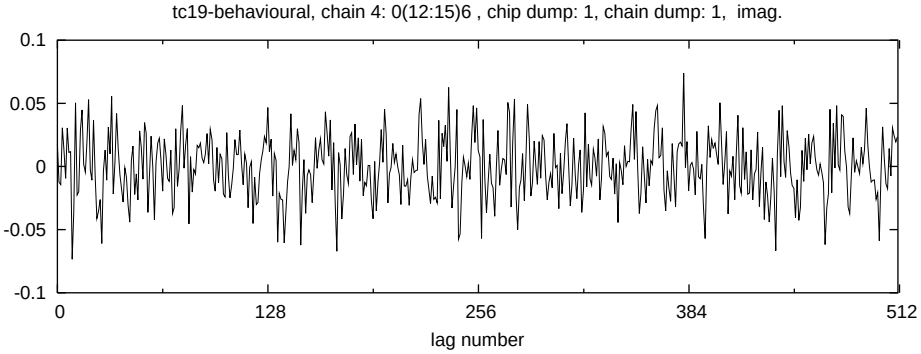

Illustration 2.7.59

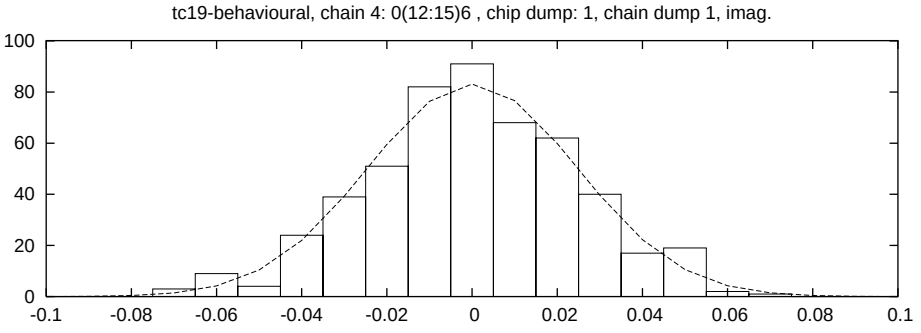

Illustration 2.7.60

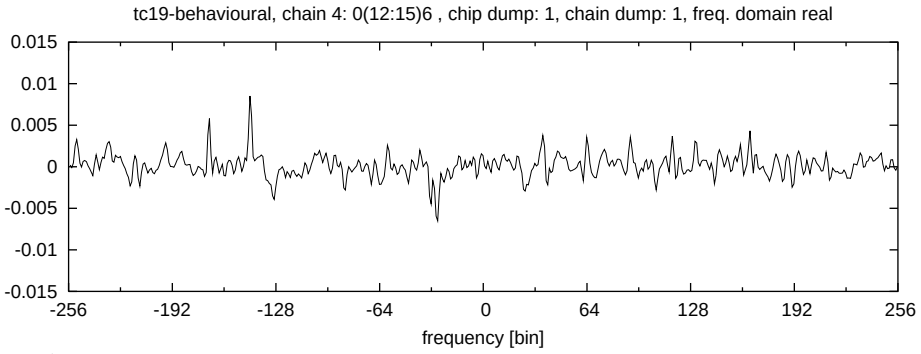

Illustration 2.7.61

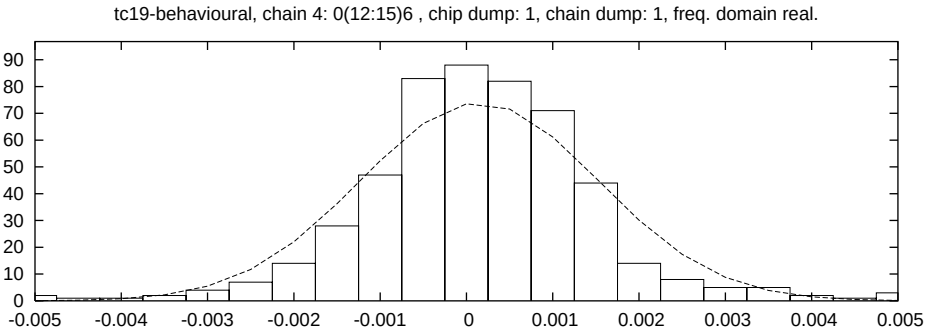

Illustration 2.7.62

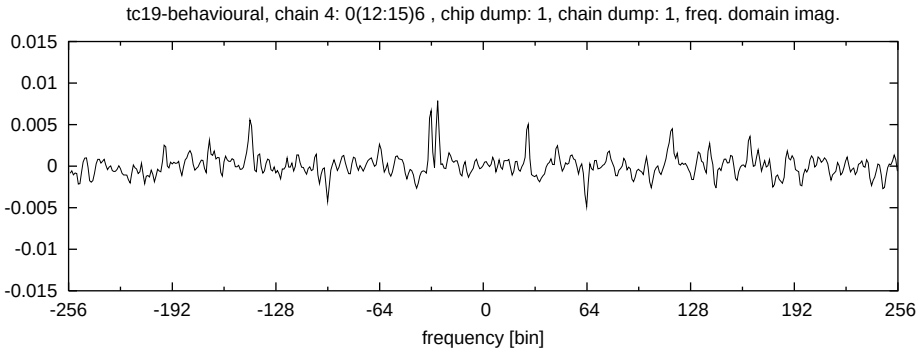

Illustration 2.7.63

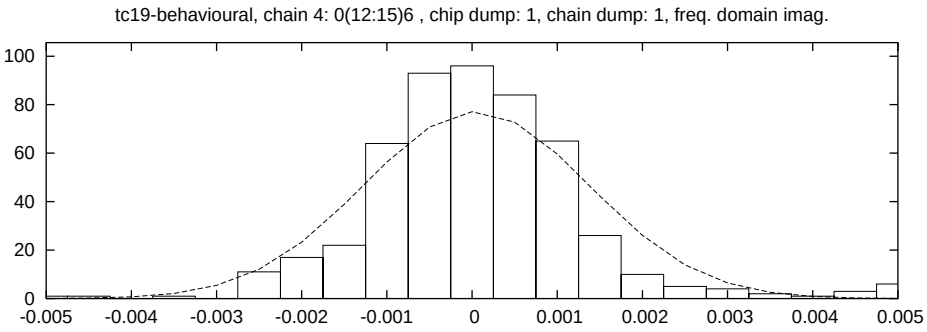

Illustration 2.7.64

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
00000101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
10000000  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=adj
00000001  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=pri
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000001  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=pri
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00010000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00010001  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=sec  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11111110  XSTATUS: X-SE_CLK: 11111110
10101010  YSTATUS: Y-SE_CLK: 10101010
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.7.11 content of file mcb_reg19.txt.

2.7.7 test case 20

Synopsis: No warnings or advisories are reported in the captured data and register content log. Continuum correlation is seen only in chain 1 and and a single dominant spectral lines is present in the plots of Illustration 2.7.65 to Illustration 2.7.72 as expected. High values for the normalized standard deviation, notably for chain 3, found in Table 2.7.14 and peak count in Table 2.7.16 prompted the generation of plots in Illustration 2.7.81 to Illustration 2.7.120. Residual structure in the lag difference plots is unexpected and manifested as spectral lines in the negative frequency domain. Similar model difference plots are presented for test case 19 in section 2.7.6 although levels are not high enough to cause concern in themselves. A set of experiments using a special test vector set applied to the behavioural model suggest that the above artifacts are a consequence of the particular combination of chip and test bench configuration used in test case 20 and not a result of difference between RTL and behavioural models. Details are found in section 2.7.8.

Notes: Tables of captured times stamps and ID data are found in (2.16.5 and 2.17.5).

```
***--------------------------------------------------------------------------***
tc20_postsim.m : V2.00 run time: Tue Sep 20 12:36:21 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 20.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
0(0:3)0 2(4:7)0 4(8:11)0 6(12:15)0


Initial (dump= 1) timestamp (word 0): 1431655765
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Writing frequency domain plot files.
Writing data ID capture tables.
Writing timestamp capture tables.
***--------------------------------------------------------------------------***

octave:40> mcb20.MCSR.XYDS
ans = 0
octave:41> tc20_make_fb_stat_tables

Using model_diff_chains: 1.00: 05-07-20
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics; using name: tc20
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.
```

Text 2.7.12 console log for scripts tc20_postsim, tc20_make_fb_stat_tables.

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*	*plot*
1	1	1.74e-03	-4.40e-04	0.1795	0.1298	0.0266	0.0224	1.33	1.12	2	
1	2	-7.87e-04	1.61e-04	0.1593	0.1512	0.0269	0.0241	1.35	1.21	3	
2	1	-2.37e-04	6.74e-04	0.1951	0.1463	0.0374	0.0295	1.87	1.47	0	
2	2	-4.14e-04	-1.12e-03	0.2069	0.1800	0.0361	0.0343	1.81	1.72	1	
3	1	3.42e-04	-2.77e-04	0.1607	0.2595	0.0322	0.0365	1.61	1.82	2	
3	2	-1.19e-04	-8.90e-04	0.2010	0.2128	0.0399	0.0408	2.00	2.04	0	
4	1	4.77e-04	-1.44e-03	0.1536	0.1773	0.0276	0.0291	1.38	1.45	1	
4	2	-2.18e-04	2.25e-04	0.1873	0.1769	0.0330	0.0314	1.65	1.57	0	

Table 2.7.14 test case 20; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
1	1	-3.57e-03	4.05e-03	0.4976	0.5204	0.0912	0.0941	1.07	1.04	256
2	1	1.14e-03	-9.48e-04	0.4696	0.5084	0.0909	0.0885	1.12	1.05	null
3	1	1.39e-03	-4.21e-04	1.0102	1.0409	0.2376	0.2500	1.02	1.01	null
4	1	5.42e-03	4.90e-03	0.5108	0.4406	0.0809	0.0814	1.01	0.98	null

Table 2.7.15 test case 20; statistical parameters of the difference between adjacent dumps of the RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[f^F-b^F]}$	$\overline{\Im[f^F-b^F]}$	*range* $\Re[f^F-b^F]$	*range* $\Im[f^F-b^F]$	$\sigma_{\Re[f^F-b^F]}$	$\sigma_{\Im[f^F-b^F]}$	$\frac{\sigma_{\Re[f^F-b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F-b^F]}}{\sigma_N}$	*peak count*	*plot*
1	1	3.99e-04	6.36e-05	0.0140	0.0096	1.44e-03	1.13e-03	2.4	1.9	0	
1	2	-1.13e-04	3.10e-04	0.0212	0.0092	1.56e-03	1.16e-03	2.6	1.9	1	
2	1	1.92e-04	1.58e-04	0.0202	0.0421	1.43e-03	2.36e-03	2.4	3.9	2	
2	2	-3.38e-05	-2.93e-04	0.0157	0.0371	1.40e-03	2.36e-03	2.3	3.9	2	
3	1	-1.06e-04	-1.10e-04	0.0349	0.0293	1.88e-03	1.94e-03	3.1	3.2	2	
3	2	8.32e-05	2.53e-04	0.0281	0.0471	1.79e-03	2.83e-03	3.0	4.7	2	
4	1	1.53e-04	-6.29e-06	0.0190	0.0218	1.32e-03	1.63e-03	2.2	2.7	1	
4	2	-9.99e-05	-8.81e-06	0.0243	0.0273	1.70e-03	1.71e-03	2.8	2.8	1	

Table 2.7.16 test case 20; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

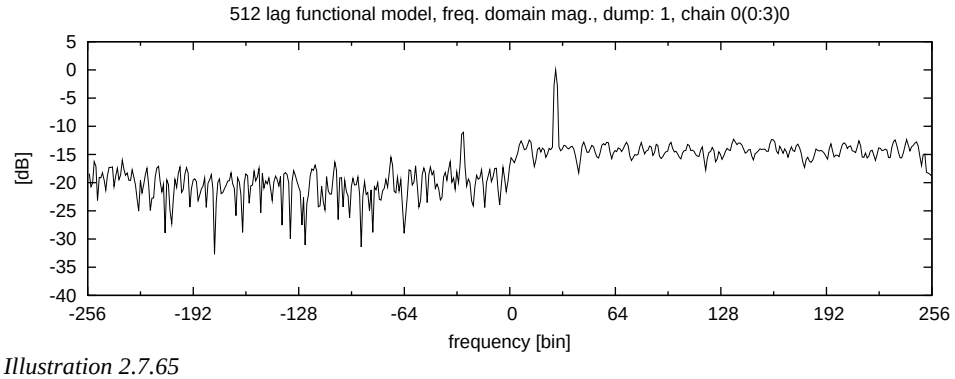

Illustration 2.7.65

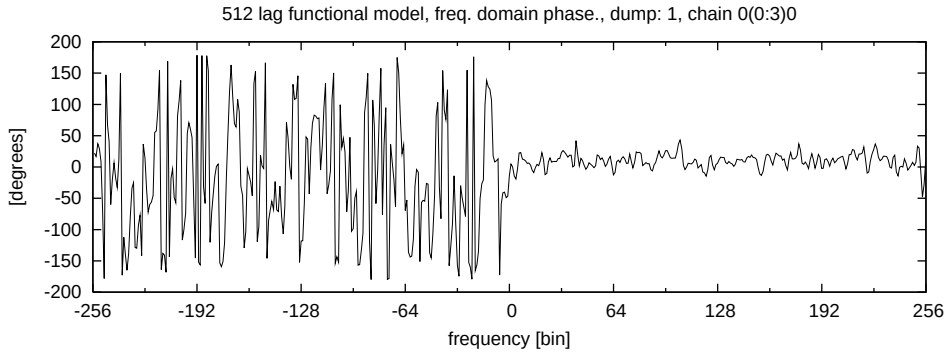

Illustration 2.7.66

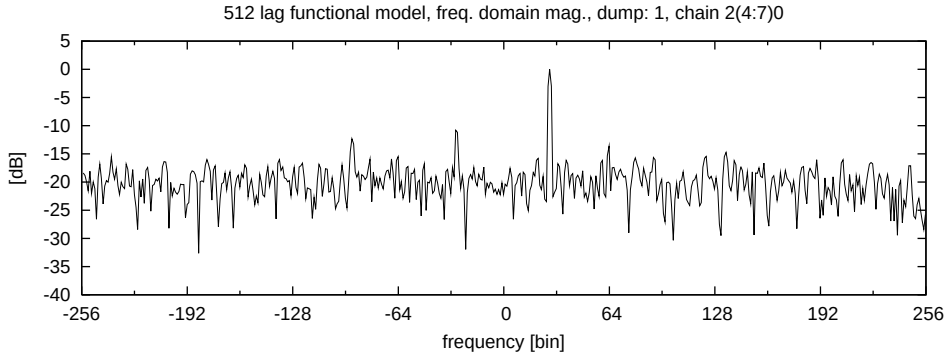

Illustration 2.7.67

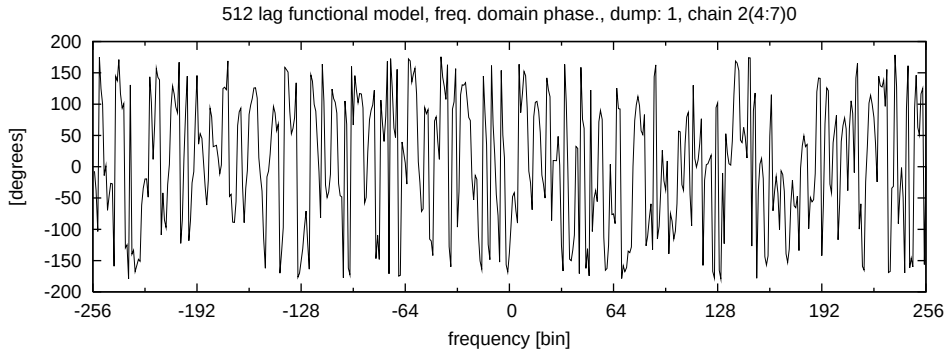

Illustration 2.7.68

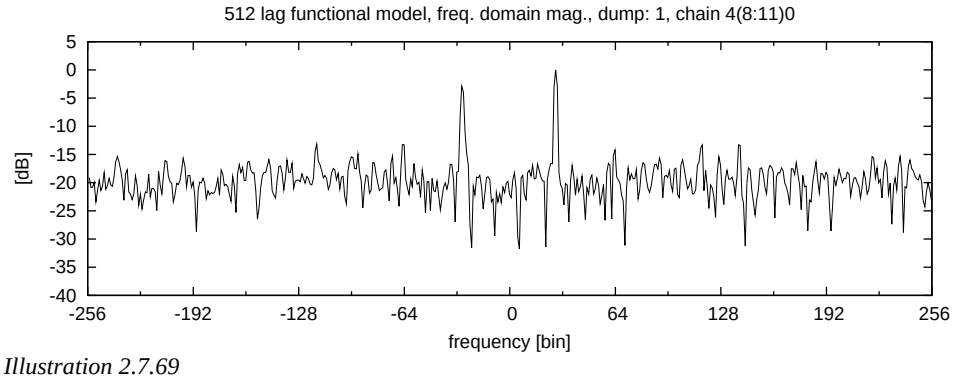

Illustration 2.7.69

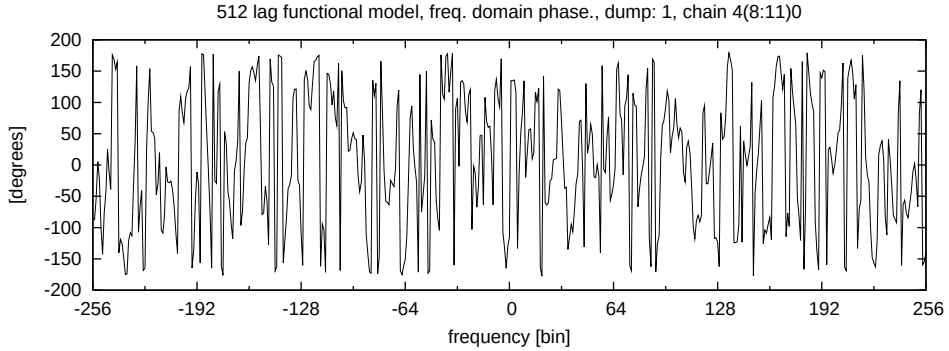

Illustration 2.7.70

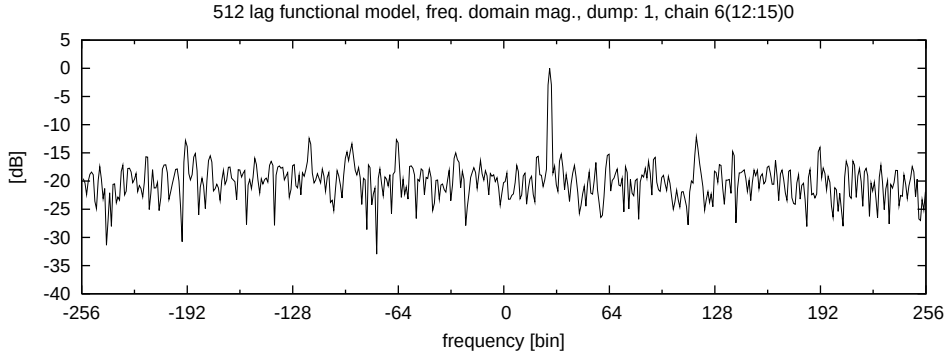

Illustration 2.7.71

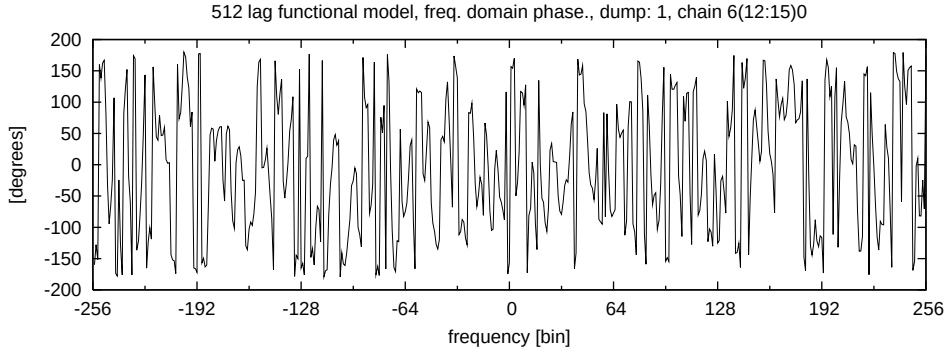

Illustration 2.7.72

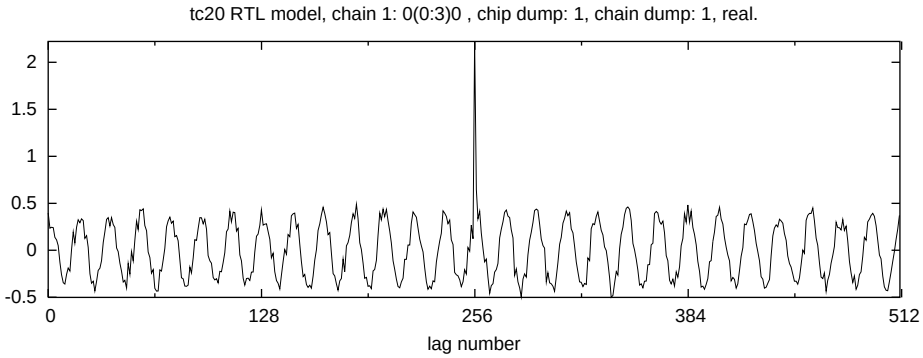

Illustration 2.7.73

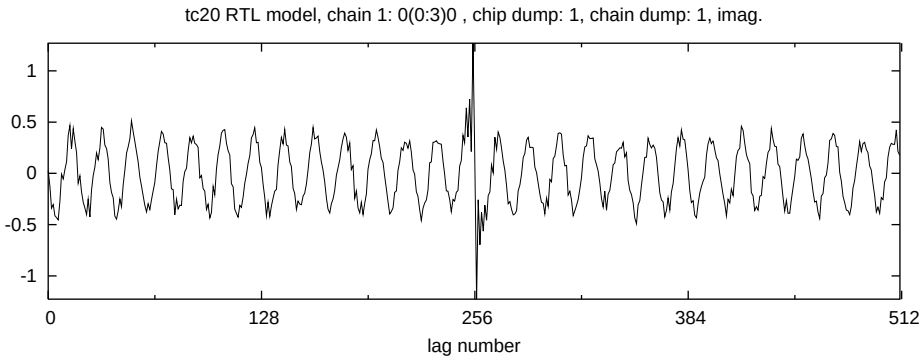

Illustration 2.7.74

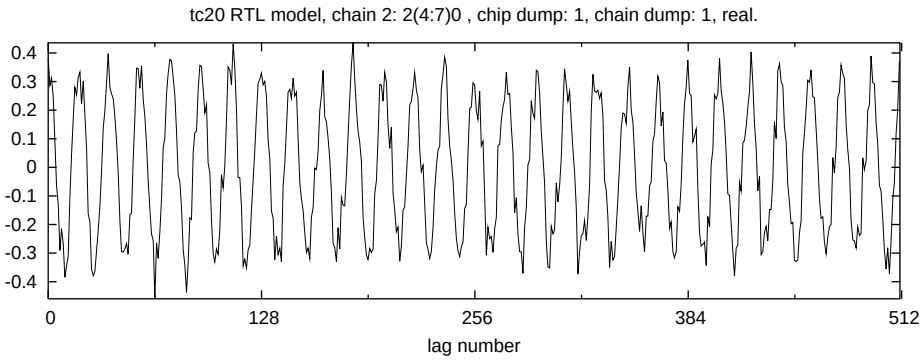

Illustration 2.7.75

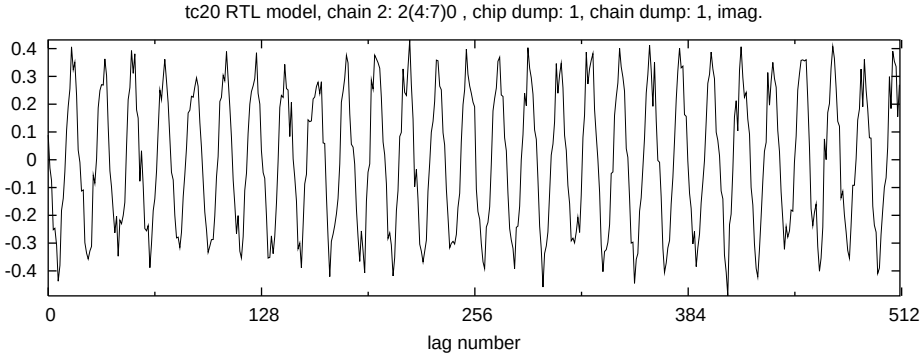

Illustration 2.7.76

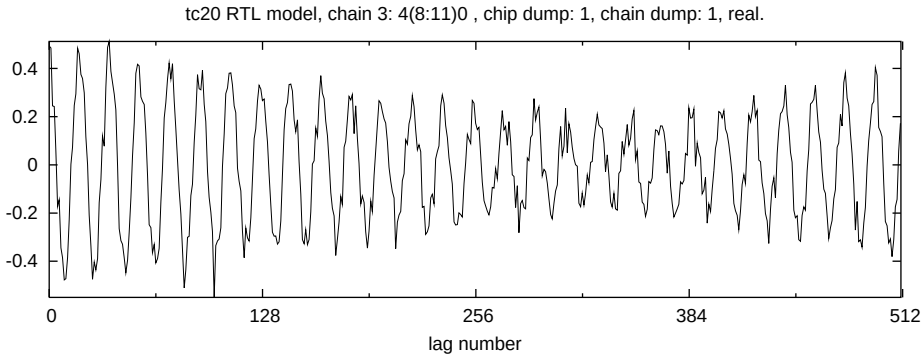

Illustration 2.7.77

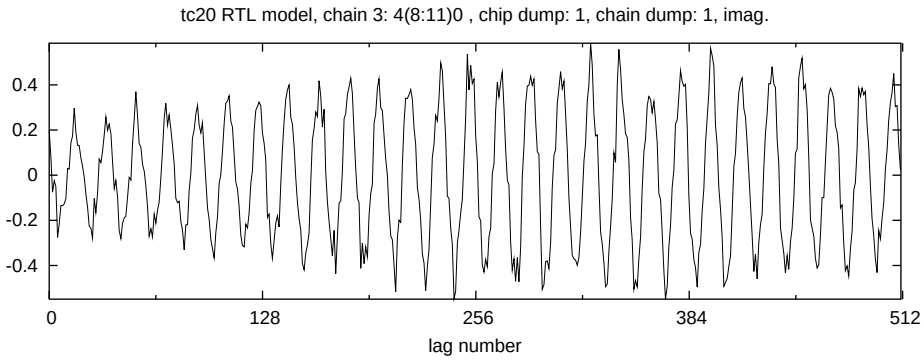

Illustration 2.7.78

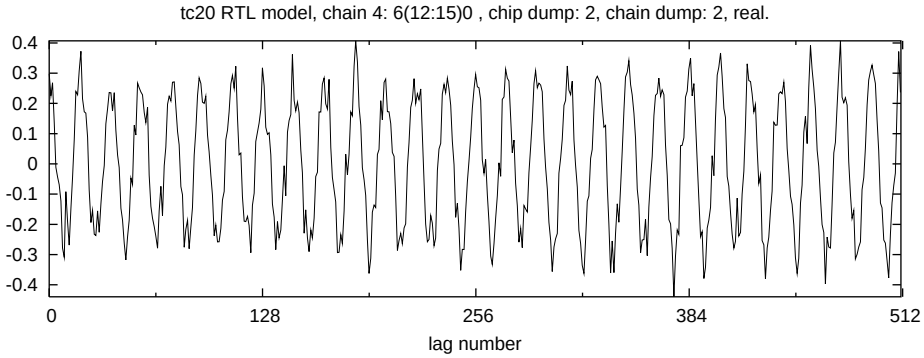

Illustration 2.7.79

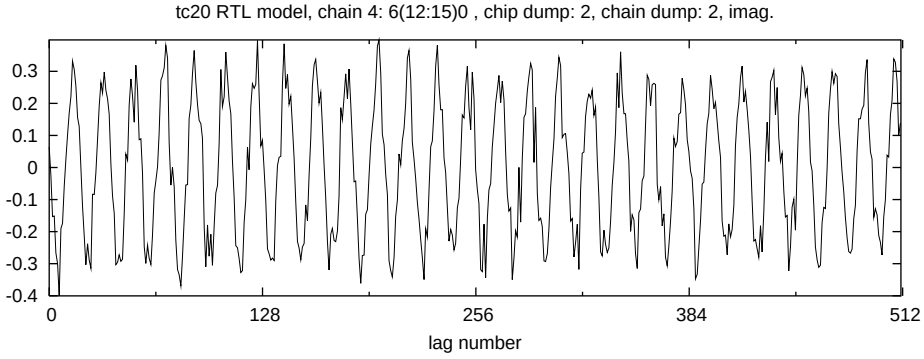

Illustration 2.7.80

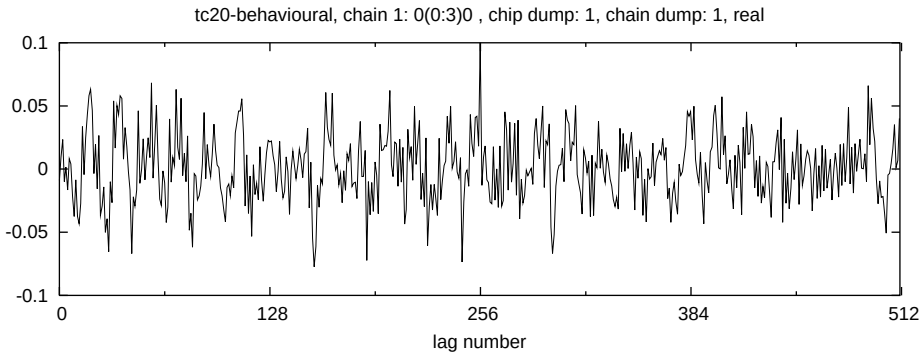

Illustration 2.7.81

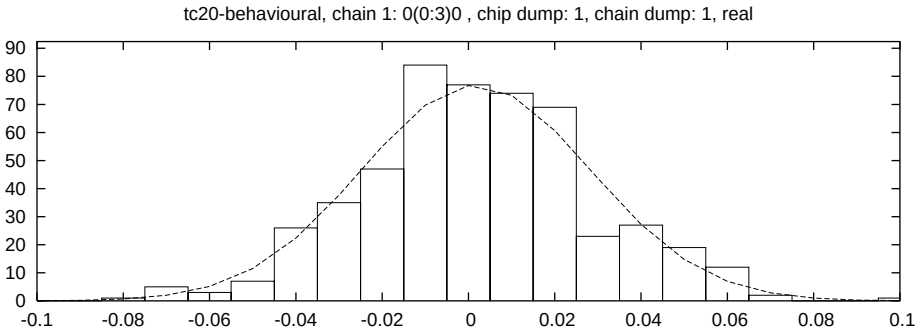

Illustration 2.7.82

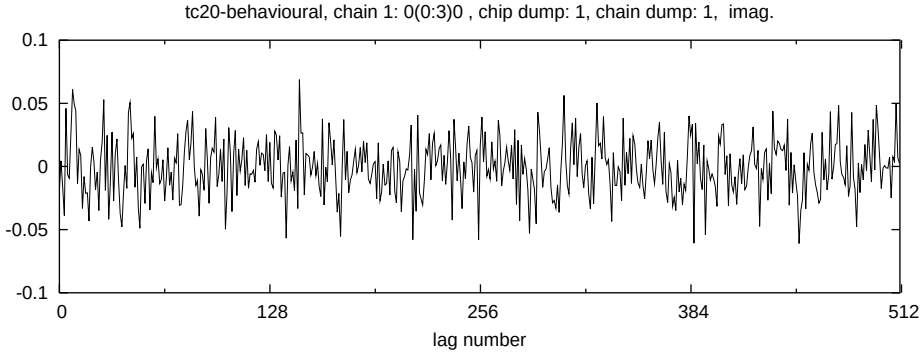

Illustration 2.7.83

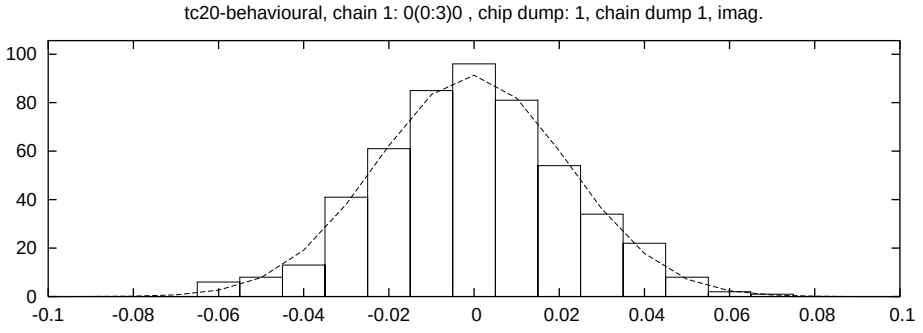

Illustration 2.7.84

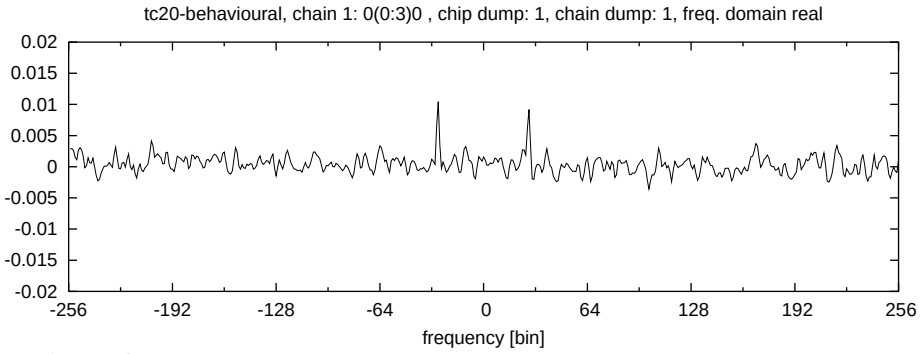

Illustration 2.7.85

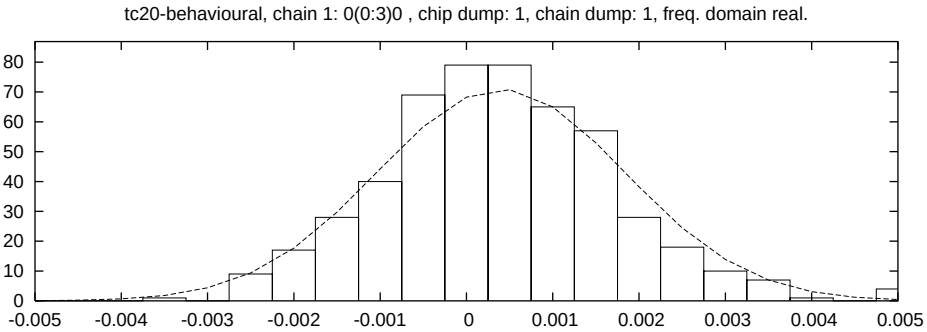

Illustration 2.7.86

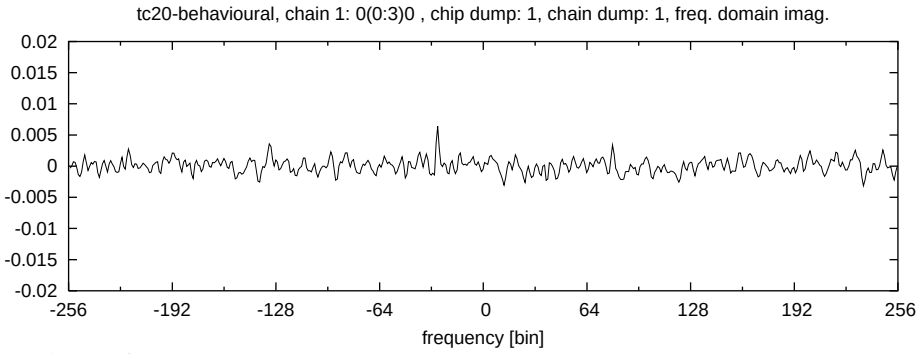

Illustration 2.7.87

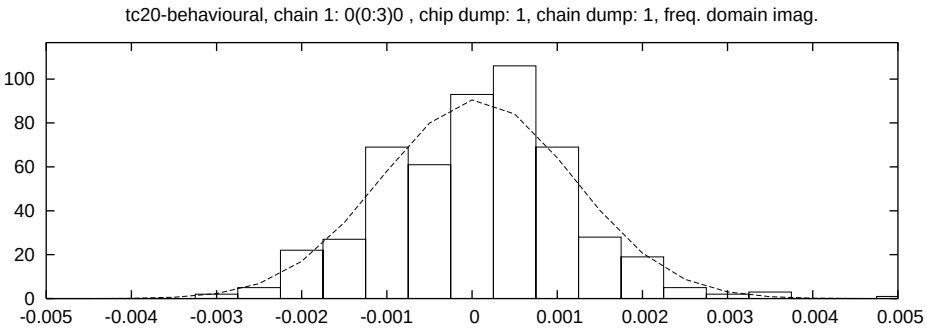

Illustration 2.7.88

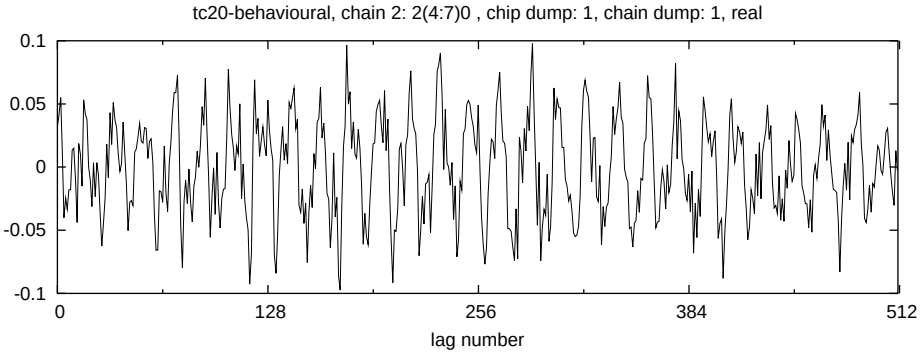

Illustration 2.7.89

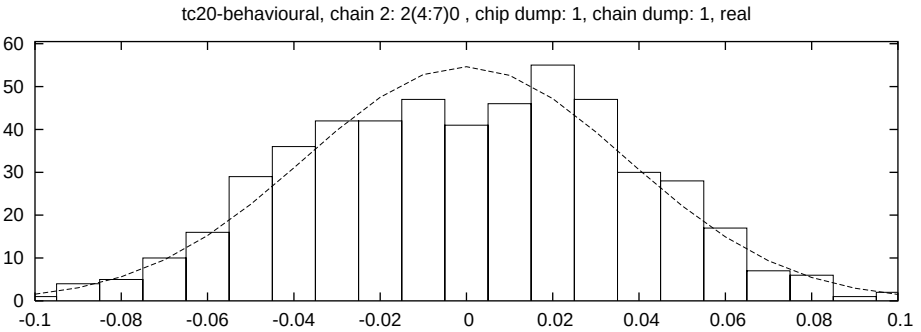

Illustration 2.7.90

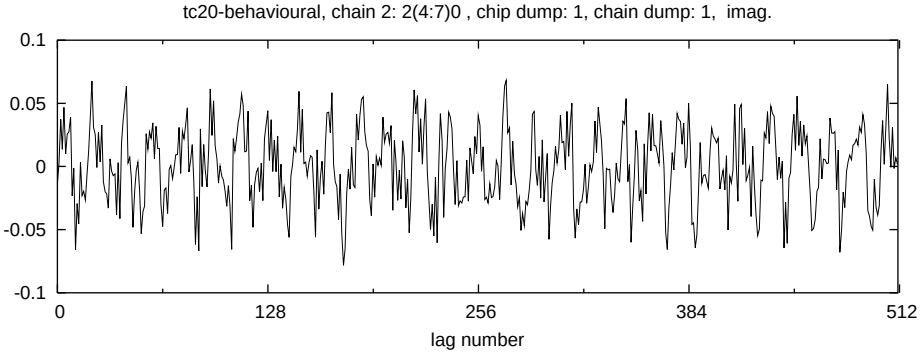

Illustration 2.7.91

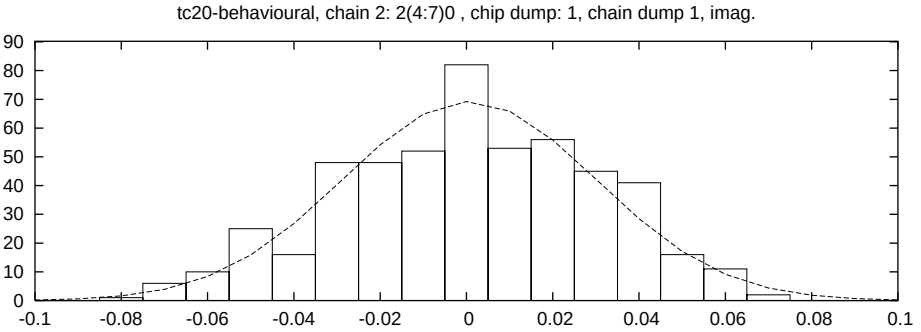

Illustration 2.7.92

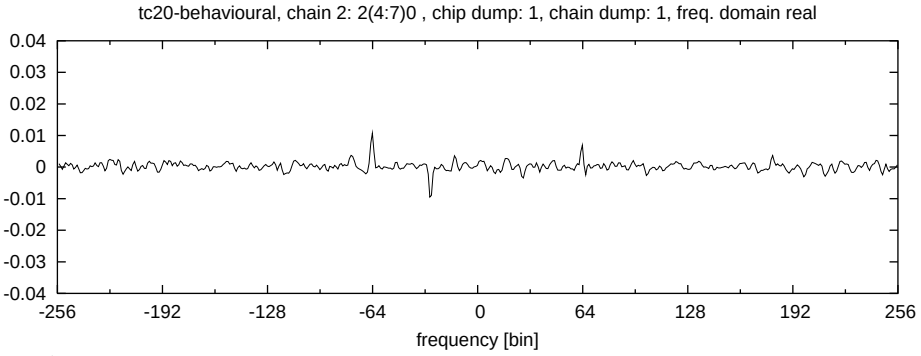

Illustration 2.7.93

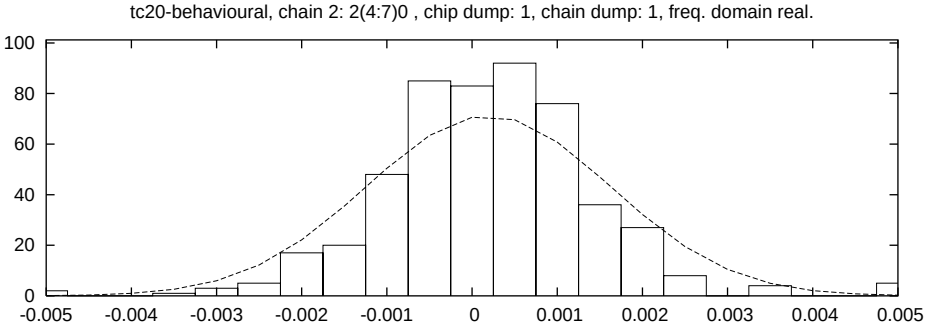

Illustration 2.7.94

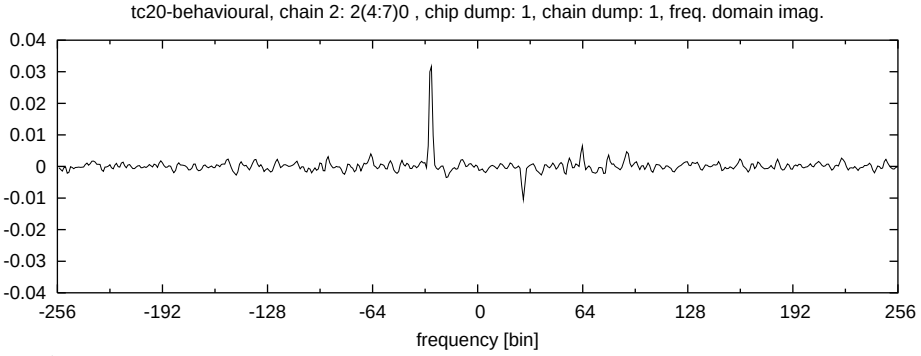

Illustration 2.7.95

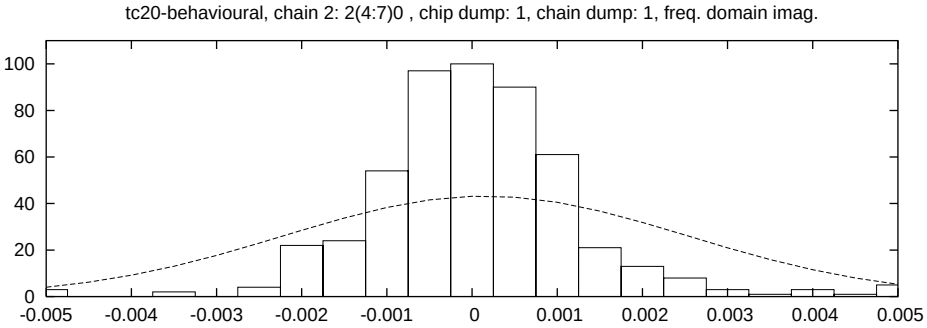

Illustration 2.7.96

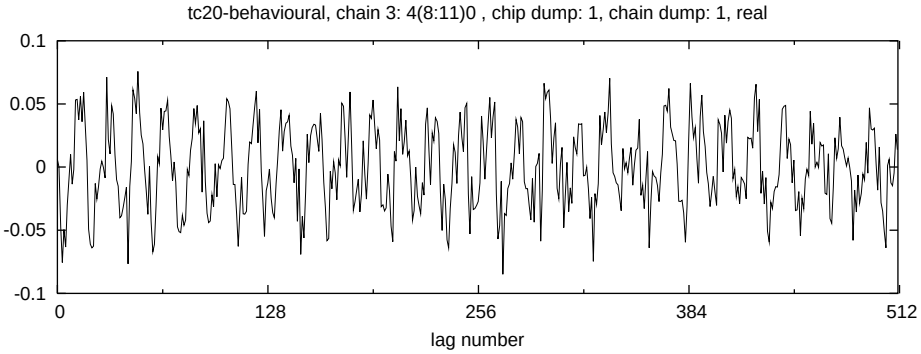

Illustration 2.7.97

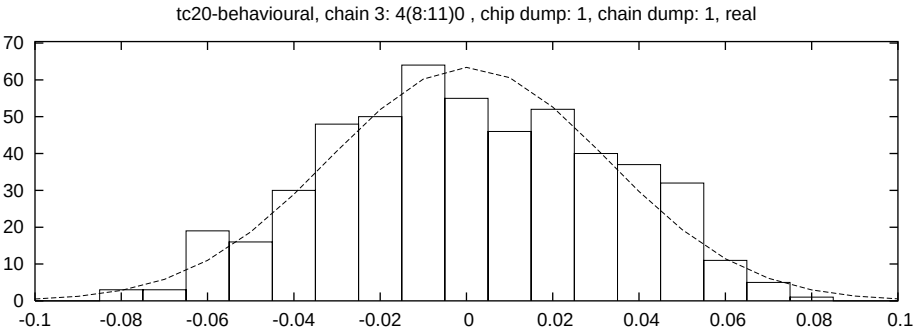

Illustration 2.7.98

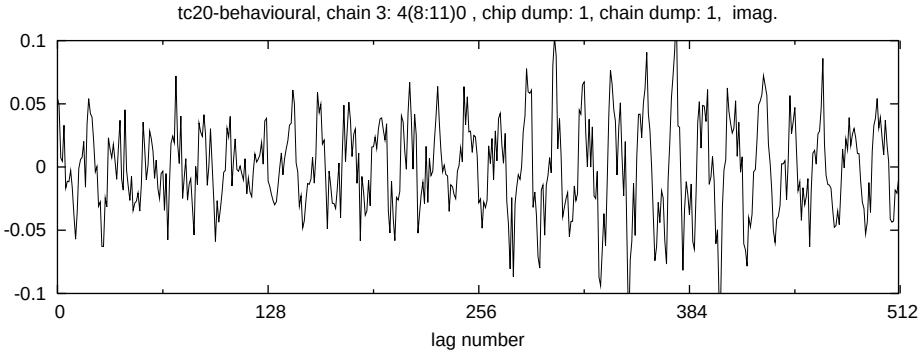

Illustration 2.7.99

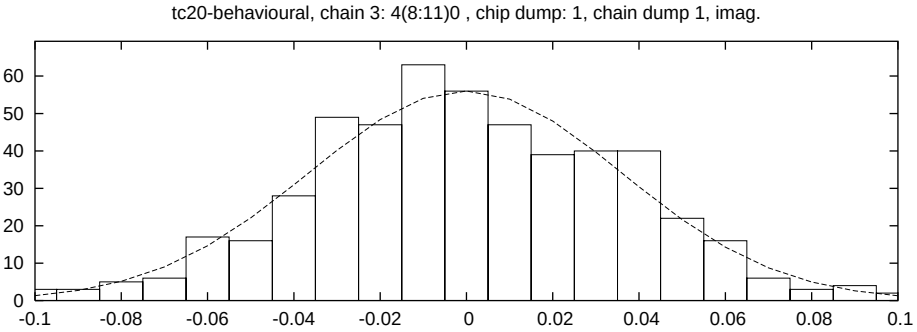

Illustration 2.7.100

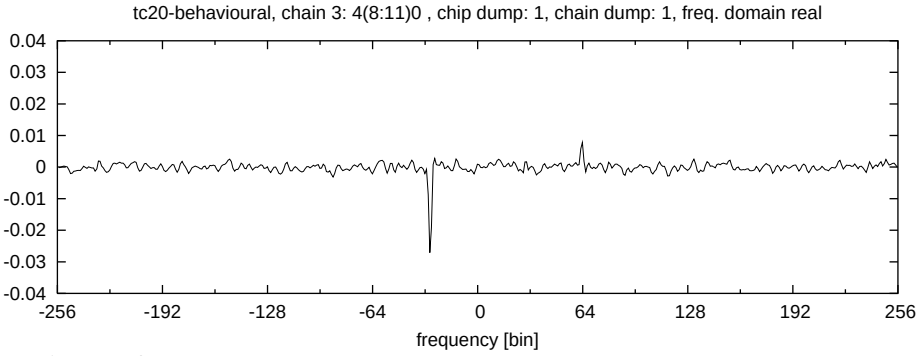

Illustration 2.7.101

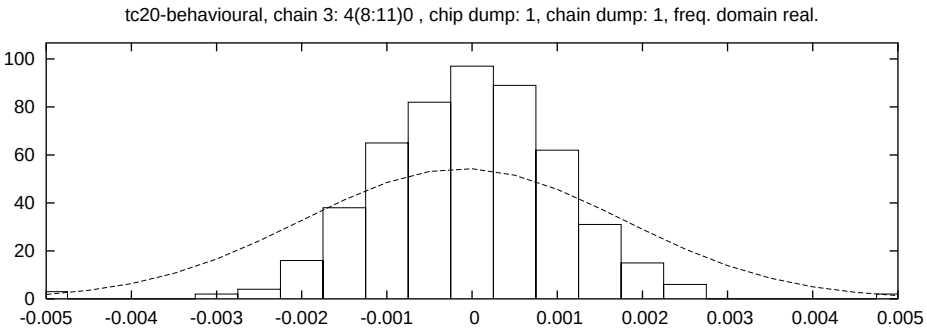

Illustration 2.7.102

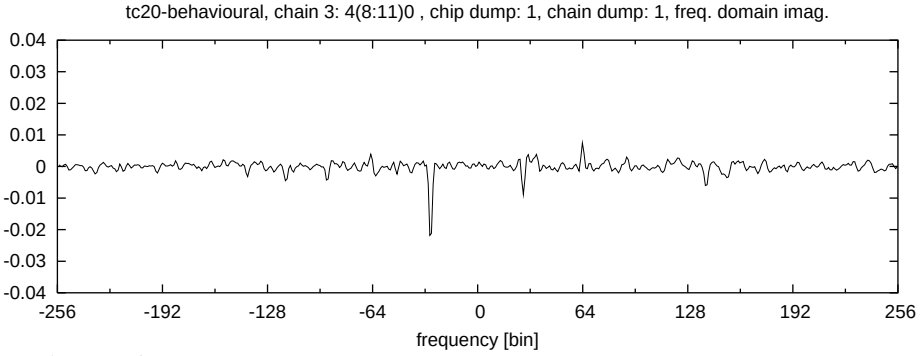

Illustration 2.7.103

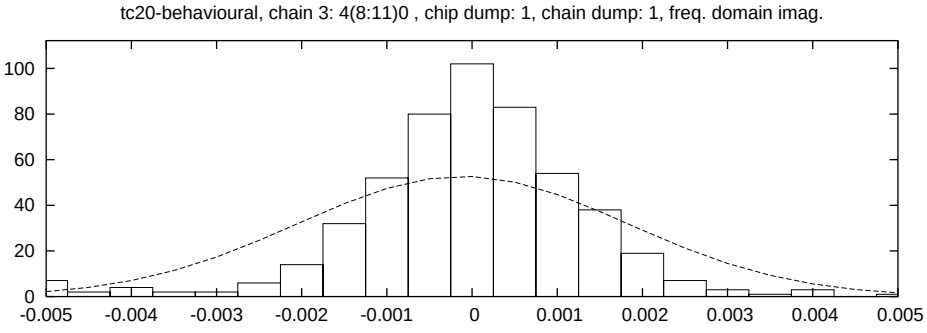

Illustration 2.7.104

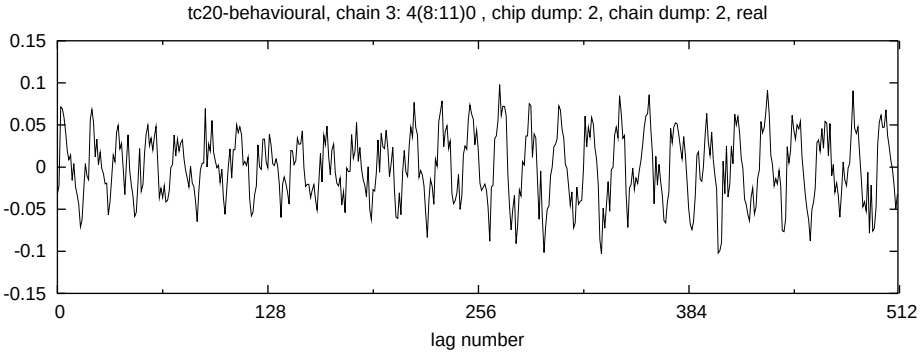

Illustration 2.7.105

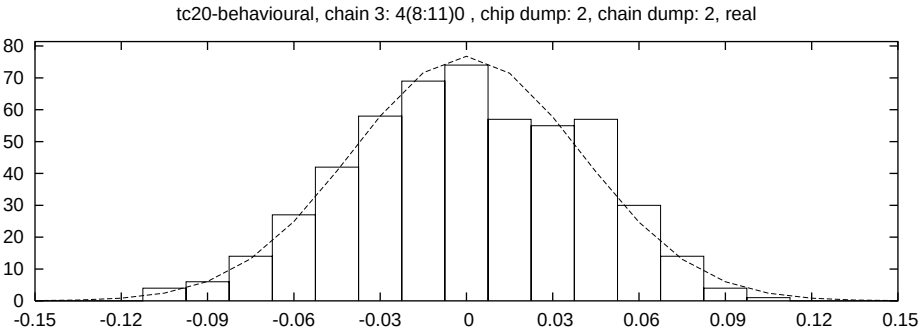

Illustration 2.7.106

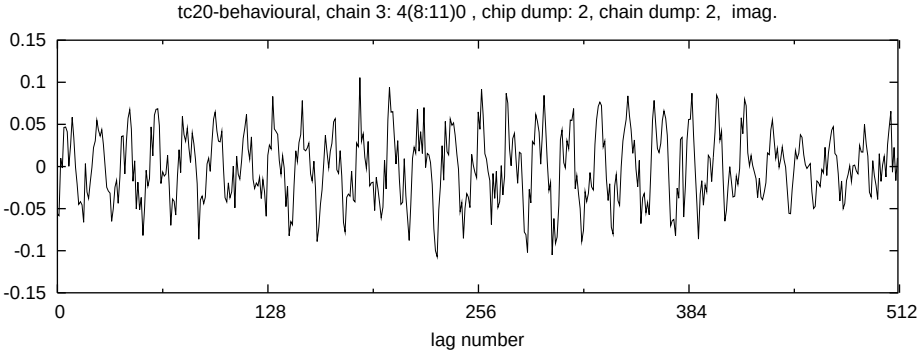

Illustration 2.7.107

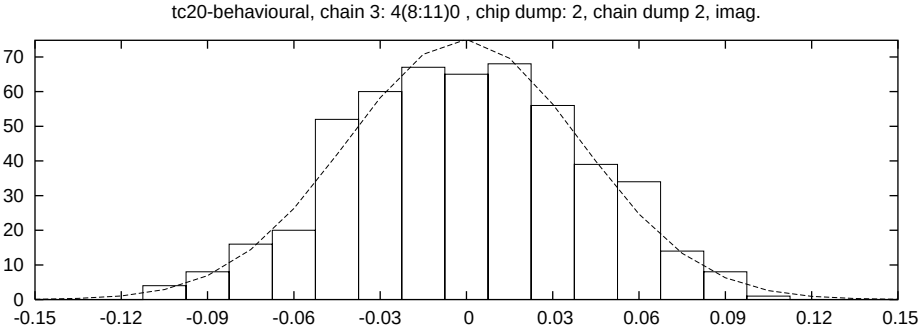

Illustration 2.7.108

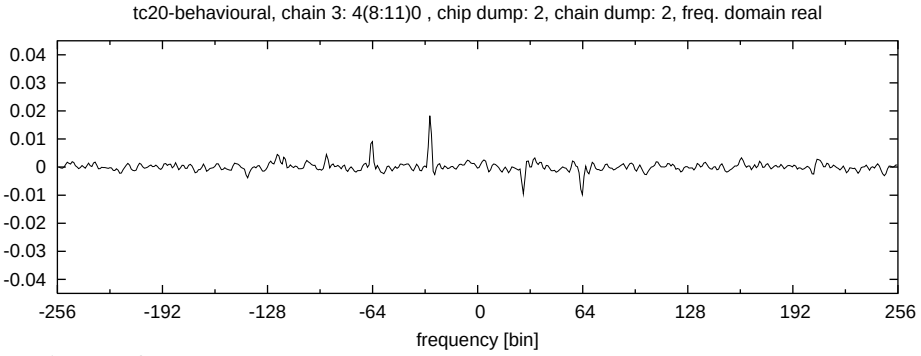

Illustration 2.7.109

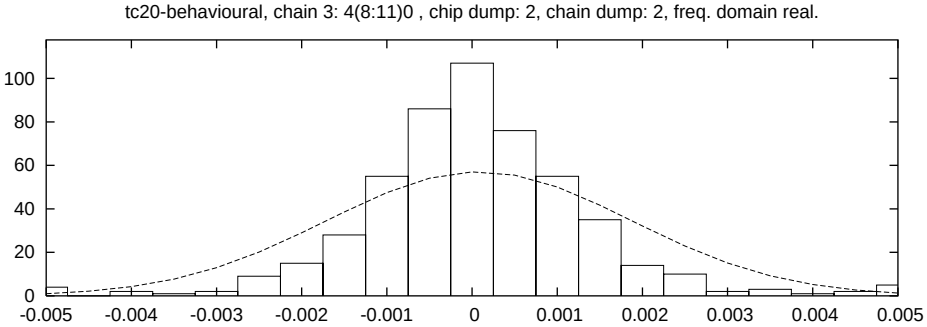

Illustration 2.7.110

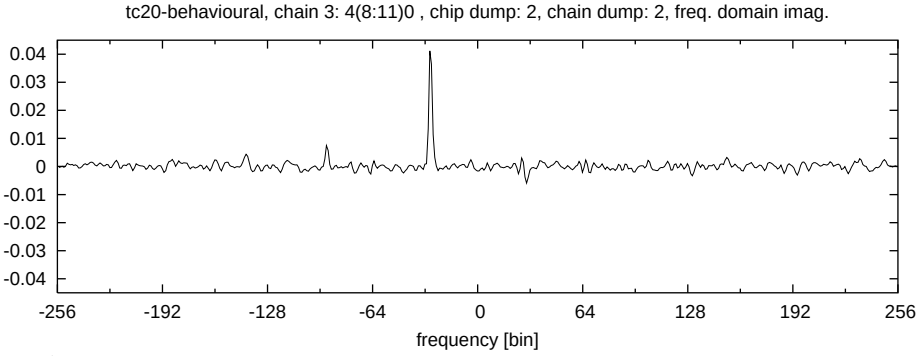

Illustration 2.7.111

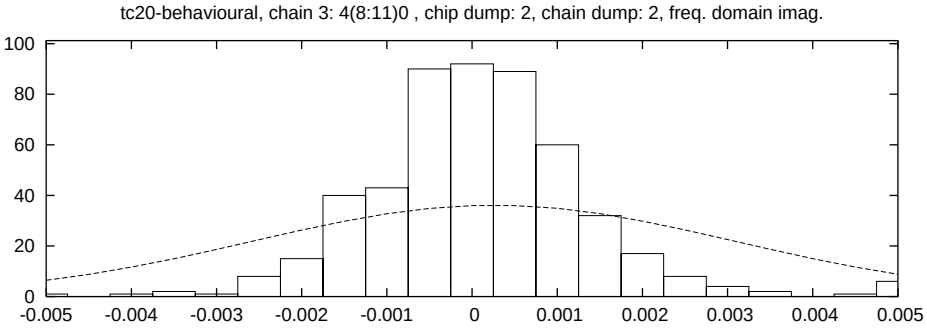

Illustration2.7.112

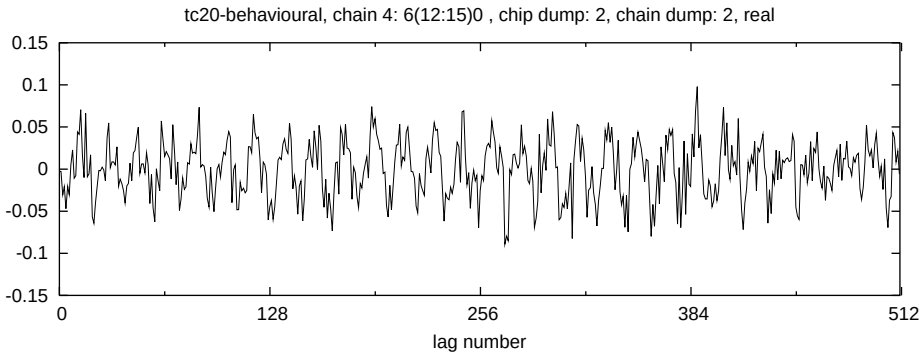

Illustration 2.7.113

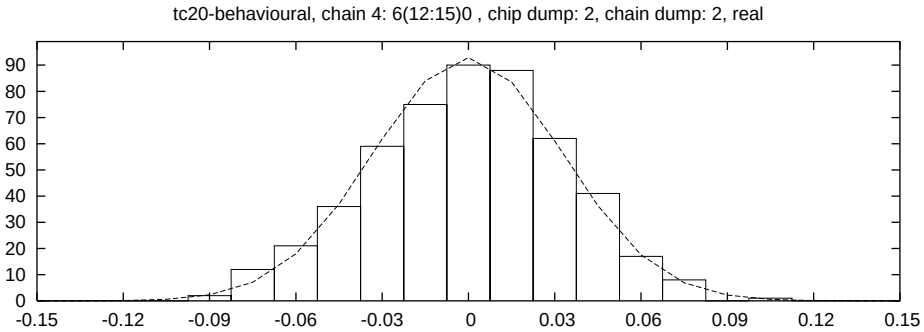

Illustration 2.7.114

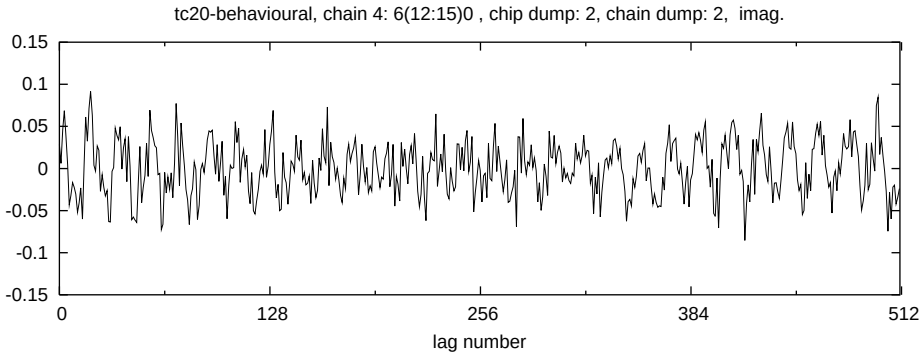

Illustration 2.7.115

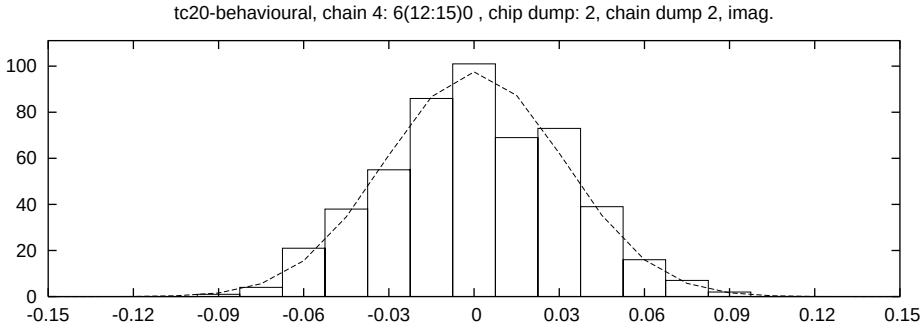

Illustration 2.7.116

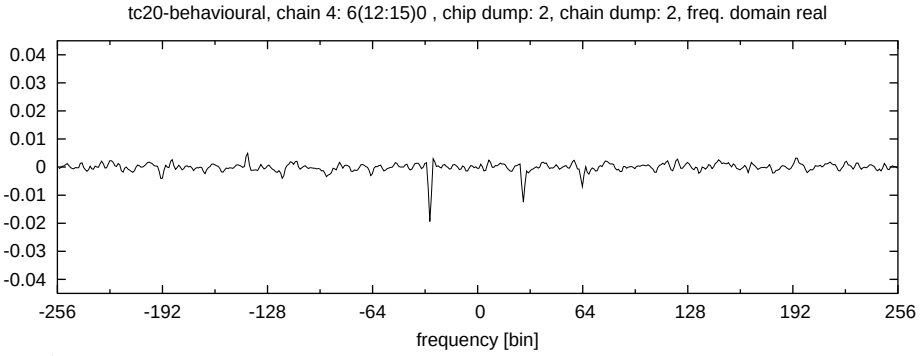

Illustration 2.7.117

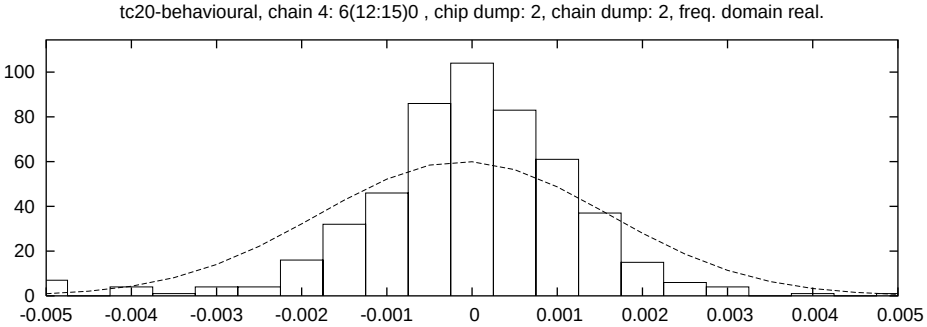

Illustration 2.7.118

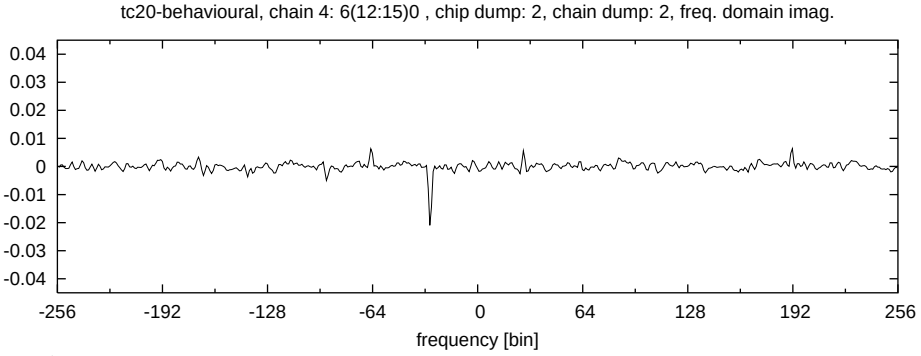

Illustration 2.7.119

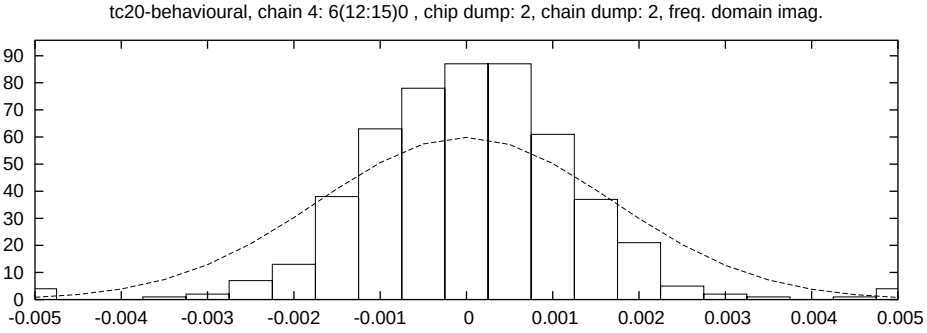

Illustration 2.7.120

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
00000101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
10000000  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=adj
00000001  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=pri
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000001  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=pri
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

01001000  CCQR_0_1: Q1: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q0: YSw1=sec YSw0=pri XSw1=pri XSw0=pri
01000100  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=sec XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
10101010  XSTATUS: X-SE_CLK: 10101010
11111110  YSTATUS: Y-SE_CLK: 11111110
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.7.13 content of file mcb_reg20.txt.

2.7.8 supplemental experiments

This section describes a set of experiments designed to demonstrate that the structure and spectral lines seen in the lag and frequency domain model difference plots of test case 20 (section 2.7.7) are explainable by known differences between the RTL and behavioural models (and between data applied to both)[3]. Rather than form the arithmetic difference between the RTL and behaviour model output as done in test case 20, the difference in output between runs of the behaviour model using two different test vector sets is examined with the objective of reproducing the former result without involving the RTL model. Output of the behavioural model using the test vector set DS4, of test case 20 is compared to that of special test vector set DS15. The latter test vectors more closely resemble the test vectors that would be applied to the corresponding RTL model by the test bench in that the **XYDVALID** bits are reset over the embedded station channel identification data at the beginning of each **SCHID** frame. Moreover, the test bench resets the **XYDVALID** bits for an interval at the start of data applied to the RTL model. The positions within the data stream of these reset **XYDVALID** bits are estimated and the corresponding bits reset in the special test vector set (DS15). Details of the data set creation are found in 3.25.4. The process used here bears some similarity to the bit exact tests described in section 2.3 . Valid data counts between individual tests are compared in Table 2.7.17.

RTL model using 12000 samples of DS4	***behavioural model using 12000 samples of DS4***	***behavioural model using 12000 samples of DS15***	***behavioural model using 48000 samples of DS4***	***behavioural model using 48000 samples of DS15***
10838	11107	10714	44247	44782

*Table 2.7.17 Valid data counts (**DVCOUNT**) for the various models.*

The plots of Illustration 2.7.121 to Illustration 2.7.128 closely resemble respective plots involving the RTL model, Illustration 2.7.97 to Illustration 2.7.104, implying that observed features within the lag and frequency domain difference plots are a consequence of the test procedure and known differences between models; they are not, in themselves, contrary to expectations. The normalized standard deviation of the lag coefficient difference between the behavioural model using test vector sets DS4 and DS15 is, for the real and imaginary coefficient parts, 1.59 and 1.63 respectively (obtained from Text 2.7.14). This compares well with values 1.61 and 1.82, of the corresponding quantities involving the RTL model tabulated in Table 2.7.14.

The plots in Illustration 2.7.129 to Illustration 2.7.140 were generated from the above experiment run with a 48000 sample dump interval (as opposed to 12000). Features in the lag and frequency domain difference plots are present but reduced in amplitude. The scale of the model difference plots are one half that of their 12000 sample counterparts.

Confidence in the above conclusion is enhanced by bit exact tests carried out on test case 30 which exhibits similar difference plots. See section 2.5.14 for details.

3 Note that these experiments do not eliminate the possibility of an unknown RTL-behavioural model difference contribution.

```
## generate special data vector set DS15 that closely resembles the
## vectors received by the RTL model.
##

octave:126> make_DS15
Data valid count in modified X data set: 45397
Data valid count in modified Y data set: 45592

## apply the DS15 vectors to the behavioural model.
## _test20x_ : 12000 sample single dump result using DS15
## _test20x4_: 48000 sample single dump result using DS15
## _test204_ : 48000 sample single dump result using DS4
## _test20_  : 12000 sample, 2 dump, result using DS4; copied from ../config
##

[smegal@castor bmod_experiments-051017]$ ./make_bdatax_tc20
Data valid count=10714
Data valid count=44247
Data valid count=44782
[smegal@castor bmod_experiments-051017]$


octave:56> tc20x_postsim
***--------------------------------------------------------------------------***
tc20x_postsim.m : V1.00 run time: Wed Oct 26 16:01:52 2005
Using postsim_lib: 1.40: 05-08-18

32 frames read for test case 20.
test case version: v3,05jun14
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
0(0:3)0 2(4:7)0 4(8:11)0 6(12:15)0


Initial (dump= 1) timestamp (word 0): 1431655765
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***

octave:57> tc20x_make_plots
Cannot open file CC0x0_test20_512lagsD0.dat.
Cannot open file CC0x0_test20_512lagsD1.dat.
Cannot open file CC2x0_test20_512lagsD0.dat.
Cannot open file CC2x0_test20_512lagsD1.dat.
Cannot open file CC6x0_test20_512lagsD0.dat.
Cannot open file CC6x0_test20_512lagsD1.dat.
Cannot open file CC0x0_test20x_512lagsD0.dat.
Cannot open file CC2x0_test20x_512lagsD0.dat.
Cannot open file CC6x0_test20x_512lagsD0.dat.

Writing plot files for chain 3, chip dump 1.

Normalized std. dev. of behavioural model difference:
real: 1.589040, imag: 1.625703

Normalized std. dev. of RTL model and x-behavioural difference:
real: 0.978180, imag: 1.016440

Data valid count for chain 3, chip dump 1: 10838
Cannot open file CC0x0_test204_512lagsD0.dat.
Cannot open file CC2x0_test204_512lagsD0.dat.
Cannot open file CC6x0_test204_512lagsD0.dat.
Cannot open file CC0x0_test20x4_512lagsD0.dat.
Cannot open file CC2x0_test20x4_512lagsD0.dat.
Cannot open file CC6x0_test20x4_512lagsD0.dat.

Writing 4x dump interval plot files for chain 3, chip dump 1.
```

Text 2.7.14 post simulation log of scripts make_DS15, make_bdatax_tc20, tc20x_postsim and tc20x_make_plots which run experiments on the behavioural model corresponding to test case 20. Only chain 3 (4(8:11)0) is considered for the most part, hence, the missing file messages are to be ignored.

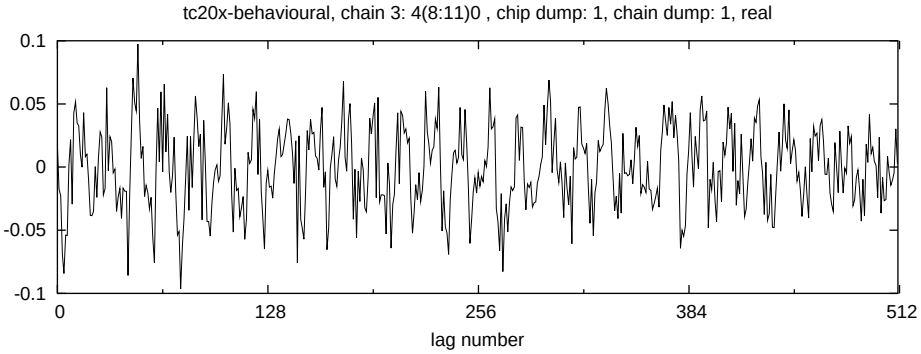

Illustration 2.7.121

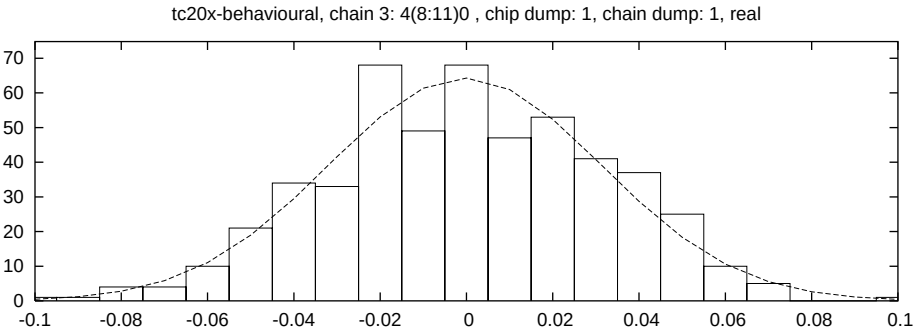

Illustration 2.7.122

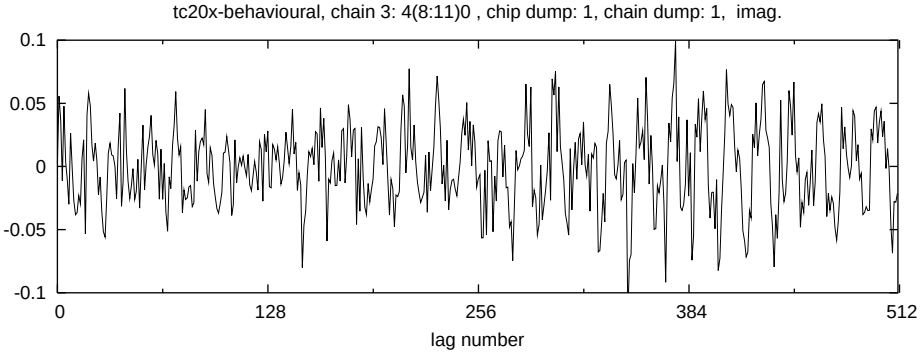

Illustration 2.7.123

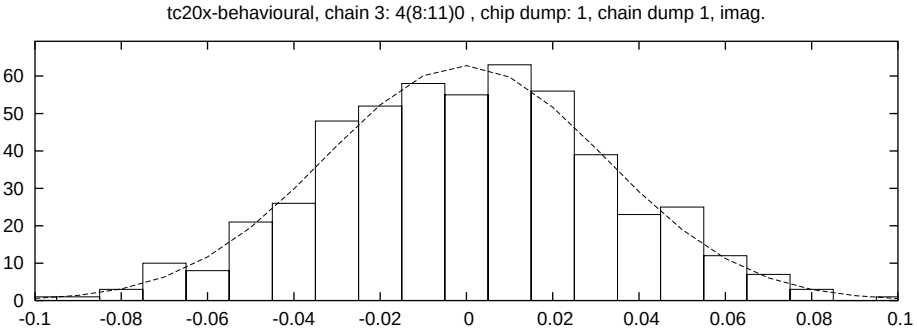

Illustration 2.7.124

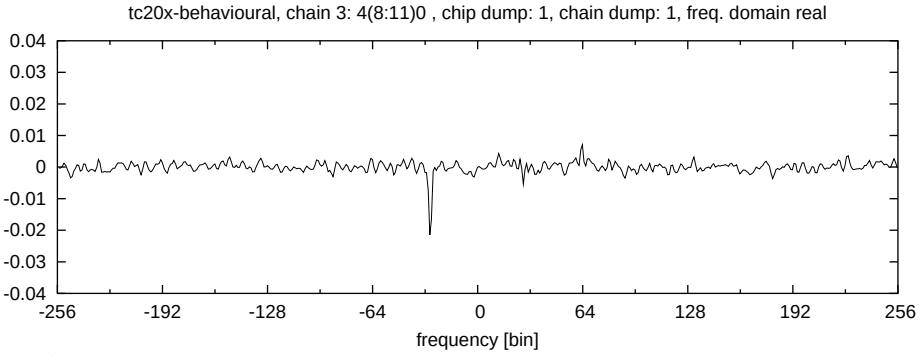

Illustration 2.7.125

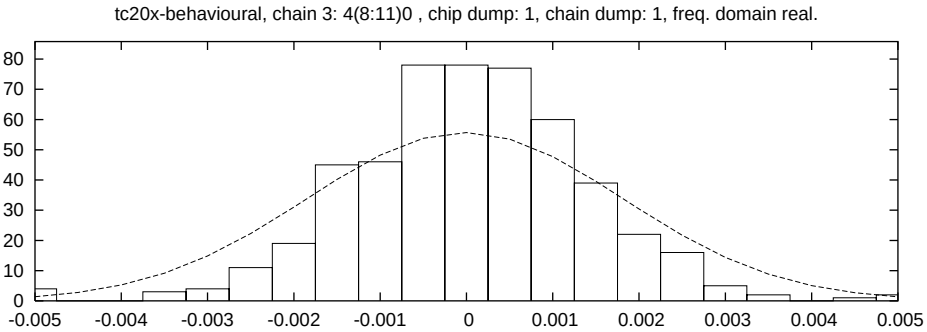

Illustration 2.7.126

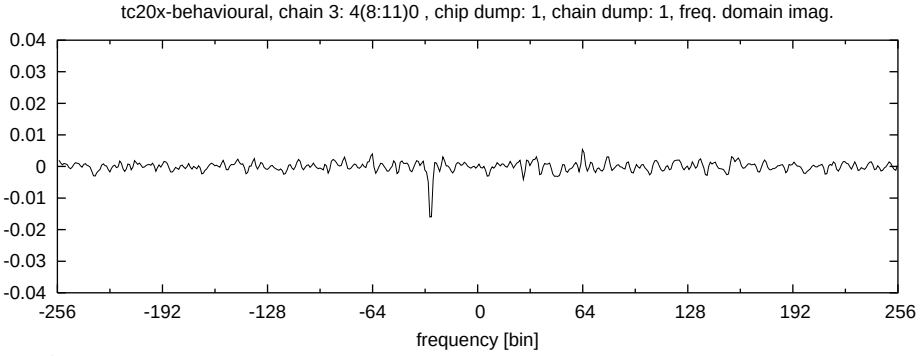

Illustration 2.7.127

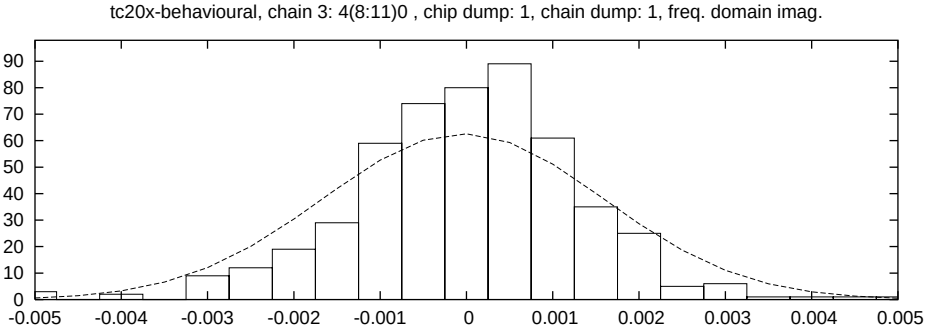

Illustration 2.7.128

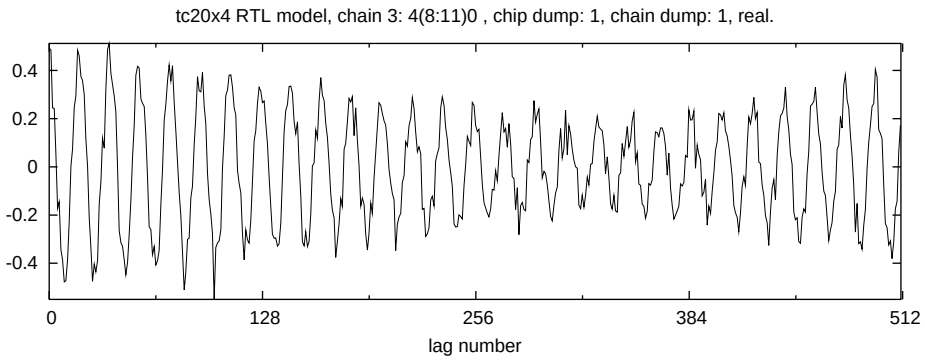

Illustration 2.7.129

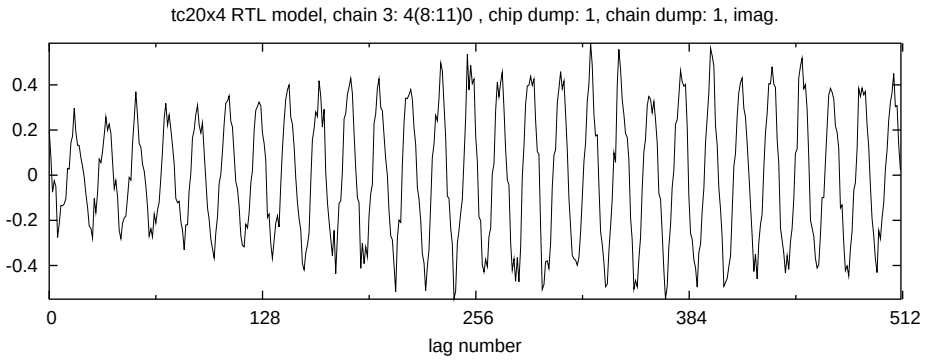

Illustration 2.7.130

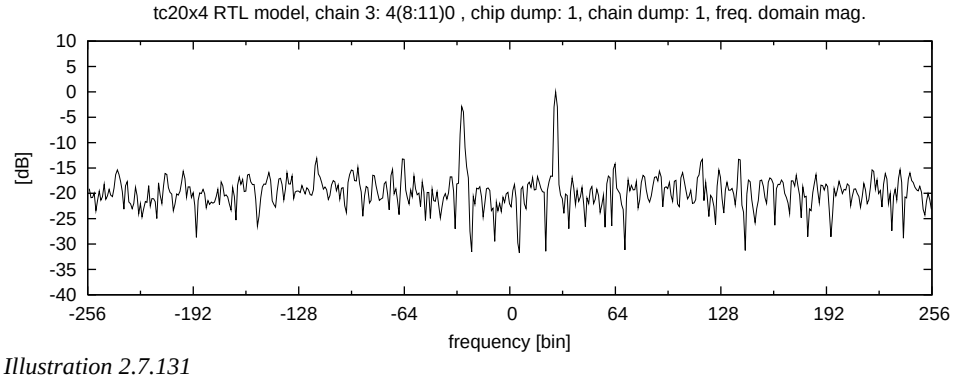

Illustration 2.7.131

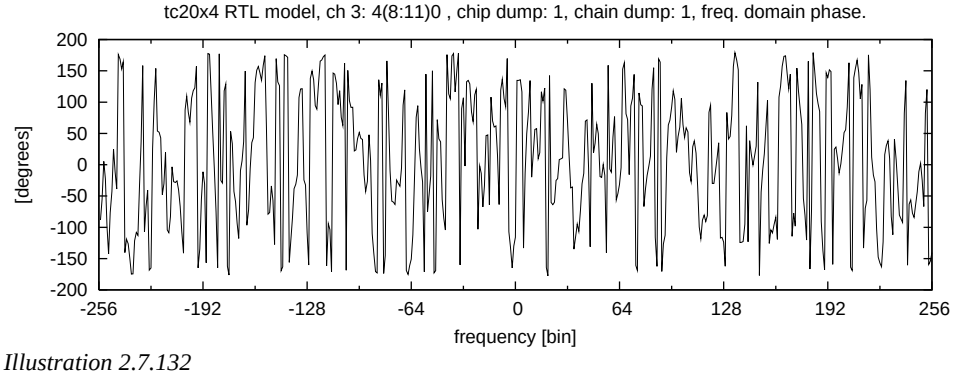

Illustration 2.7.132

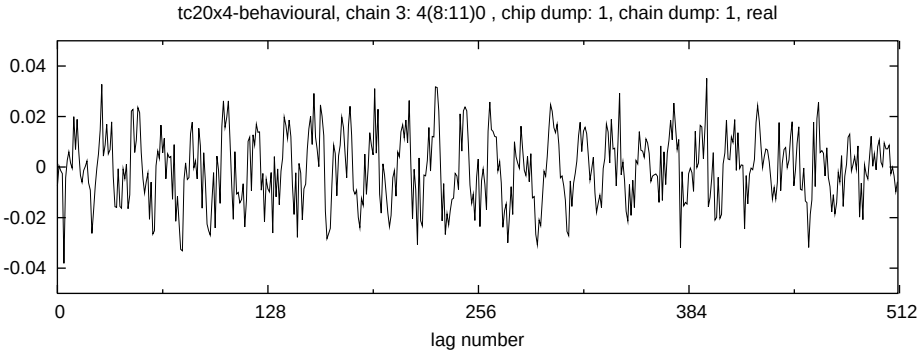

Illustration 2.7.133

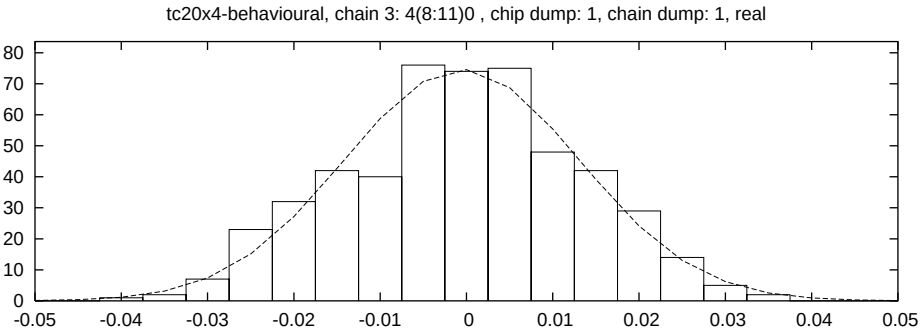

Illustration 2.7.134

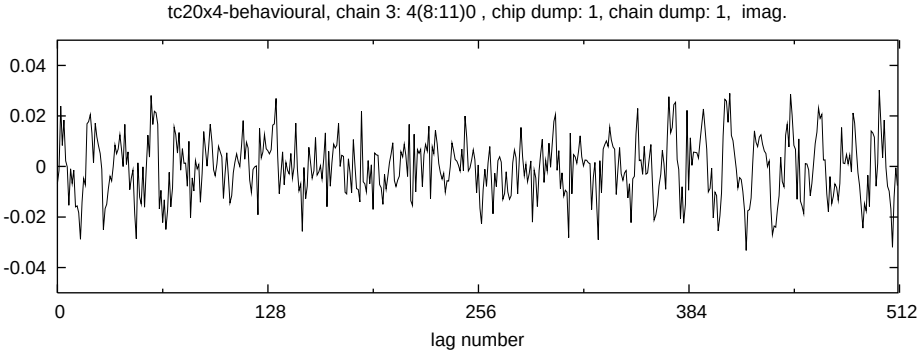

Illustration 2.7.135

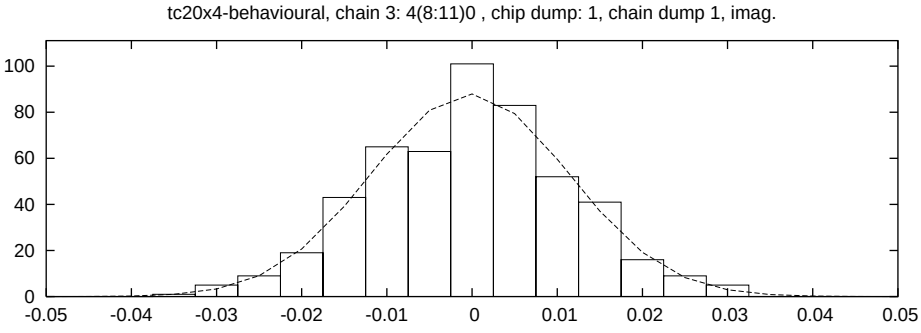

Illustration 2.7.136

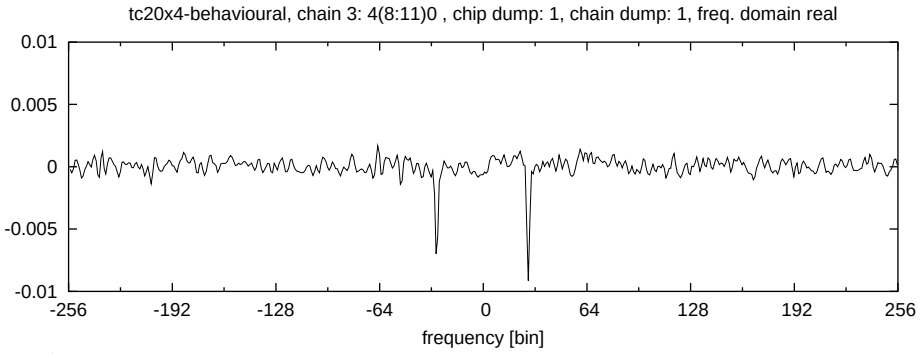

Illustration 2.7.137

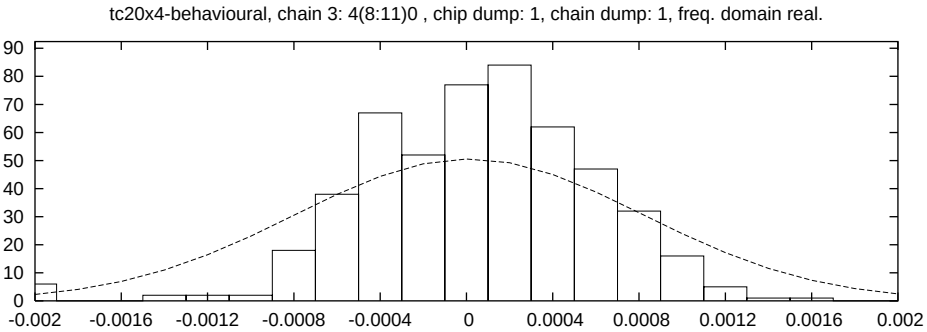

Illustration 2.7.138

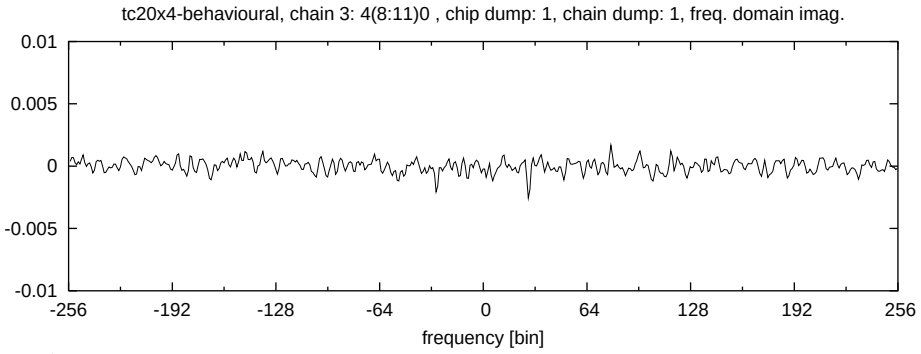

Illustration 2.7.139

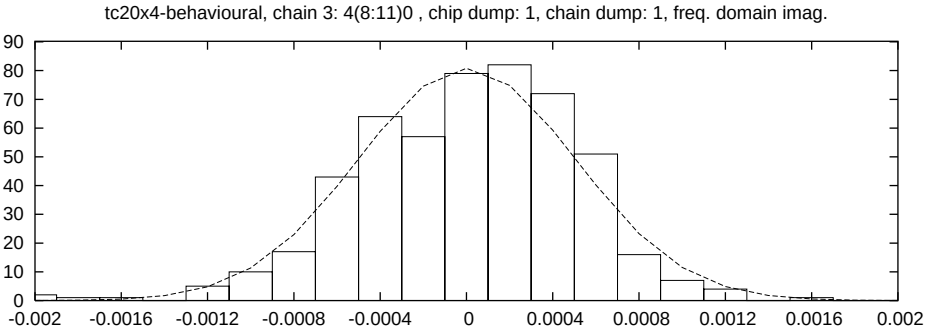

Illustration 2.7.140

2.7.9 test case 68

Synopsis: No extraordinary plot features or table entries are noted and no advisories or warnings regarding captured data or register content were logged. The phase plots of Illustration 2.7.141 to Illustration 2.7.144 are consistent with the absence of continuum correlation as expected and the plots of Illustration 2.7.145 to Illustration 2.7.148 demonstrate that continuum correlation is easily detectable with the short dump intervals used in this test case. The output generated by the cells of quad 0 compare on a bit exact basis to similarly configured cells of quad 4.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc68_postsim.m : V1.00 run time: Sun Dec 11 21:20:40 2005
Using postsim_lib: 1.40: 05-08-18

12 frames read for test case 68.
test case version: v1,05nov04
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 12.
6(0:0)0 6(1:1)1 7(2:2)0 7(3:3)1 0(8:8)0 0(9:9)1 1(10:10)0 1(11:11)1 6(12
:12)0 6(13:13)1 7(14:14)0 7(15:15)1


Initial (single dump) timestamp (word 0): 222333444

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***
octave:3> tc68_similarity_check

No differences exceeding the floating point precision limit of this
        machine were found between similar chains of test case 68.
```

Text 2.7.15: console log of tc68_postsim, tc68_similarity_check.

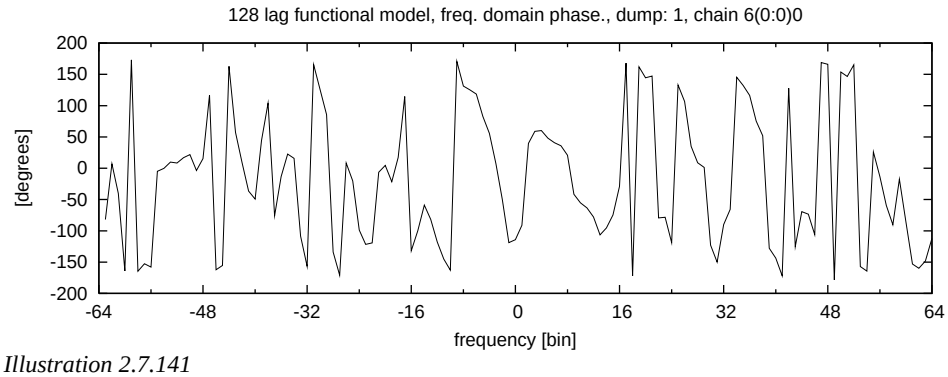

Illustration 2.7.141

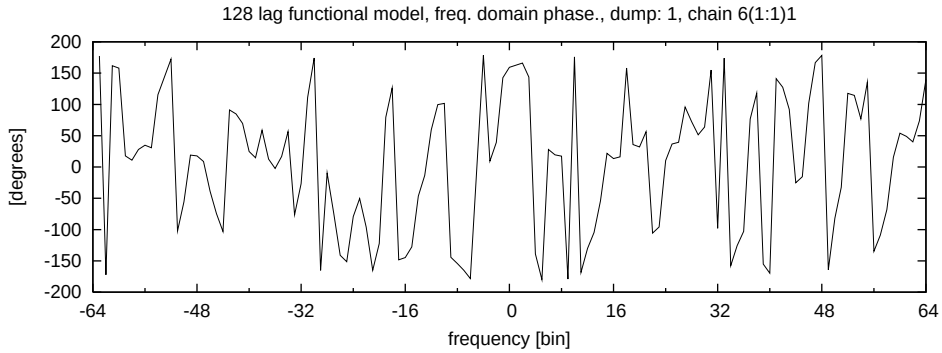

Illustration 2.7.142

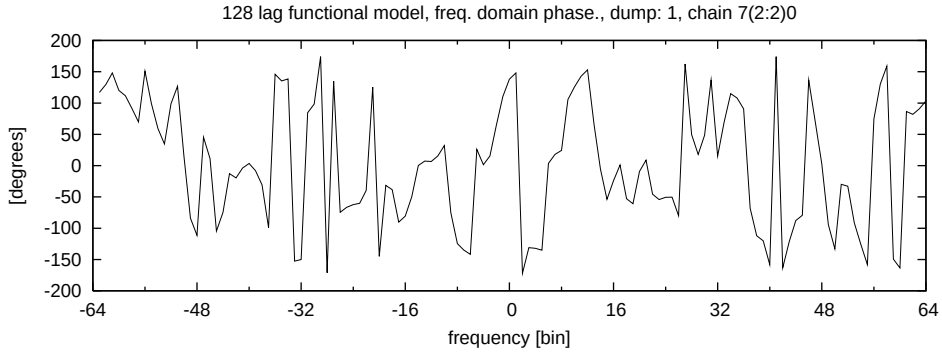

Illustration 2.7.143

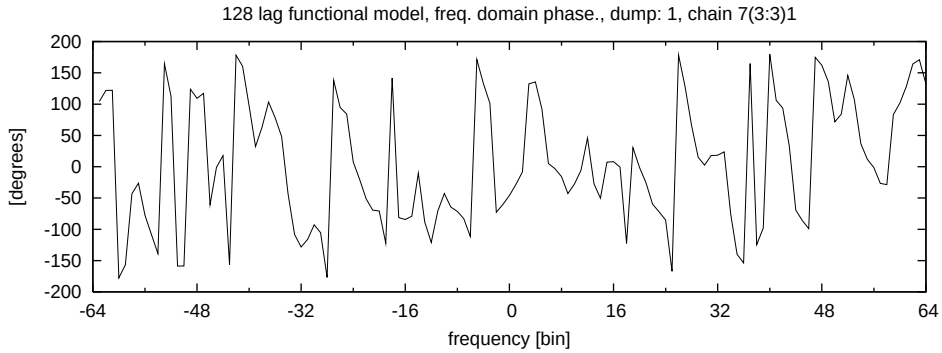

Illustration 2.7.144

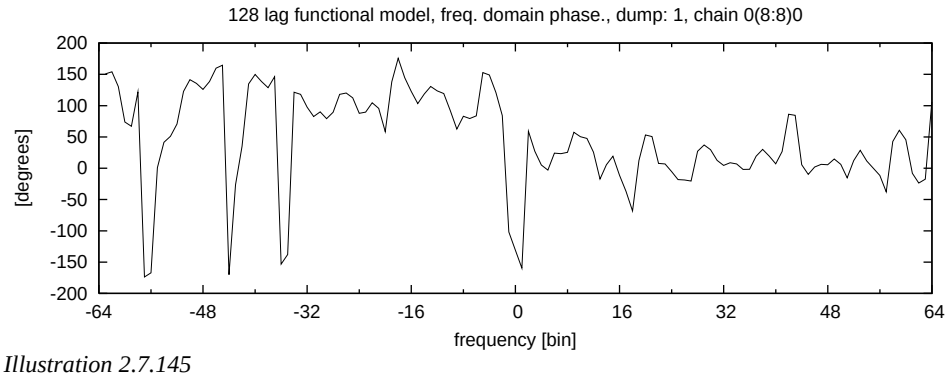

Illustration 2.7.145

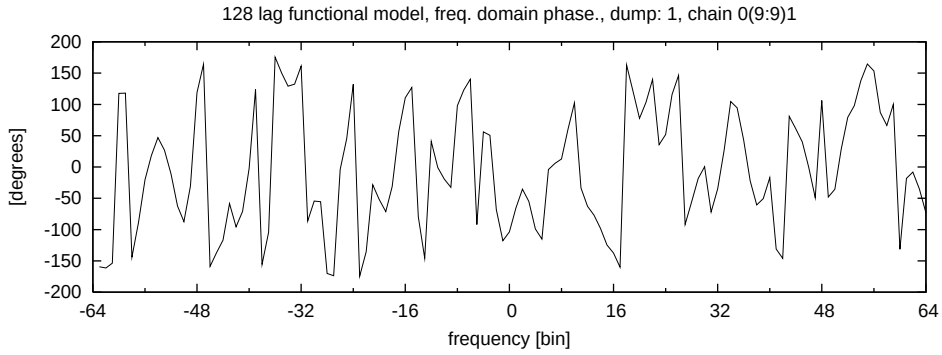

Illustration 2.7.146

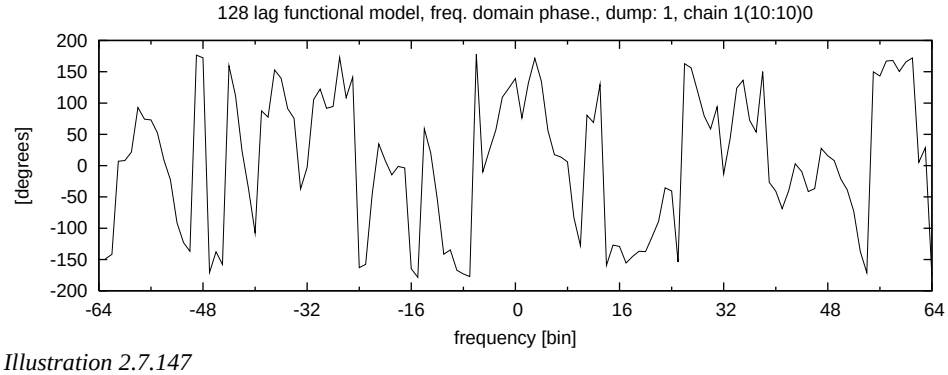

Illustration 2.7.147

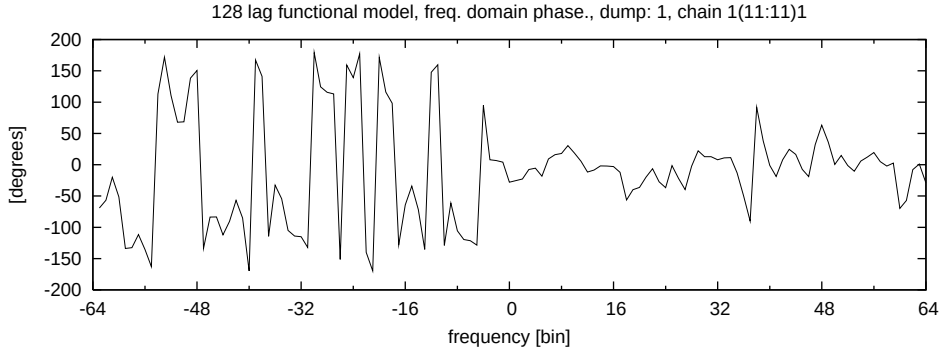

Illustration 2.7.148

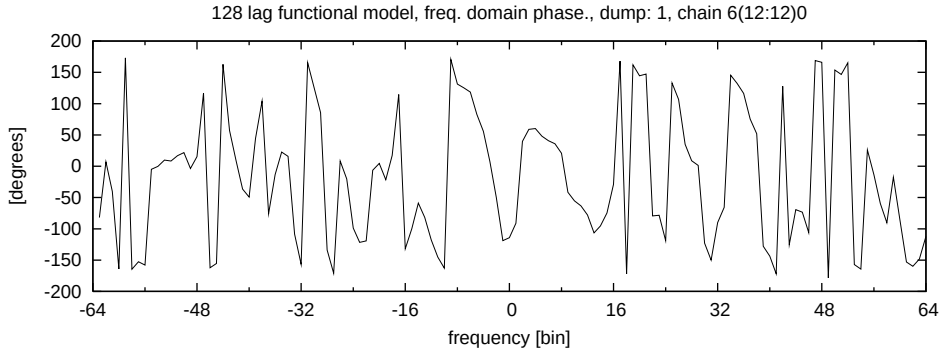

Illustration 2.7.149

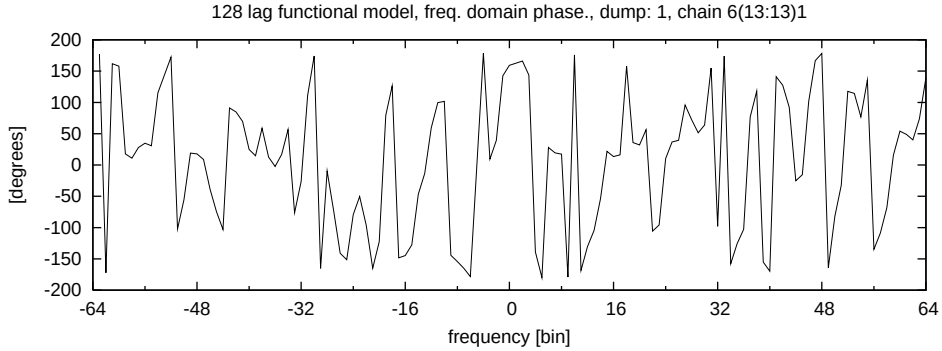

Illustration 2.7.150

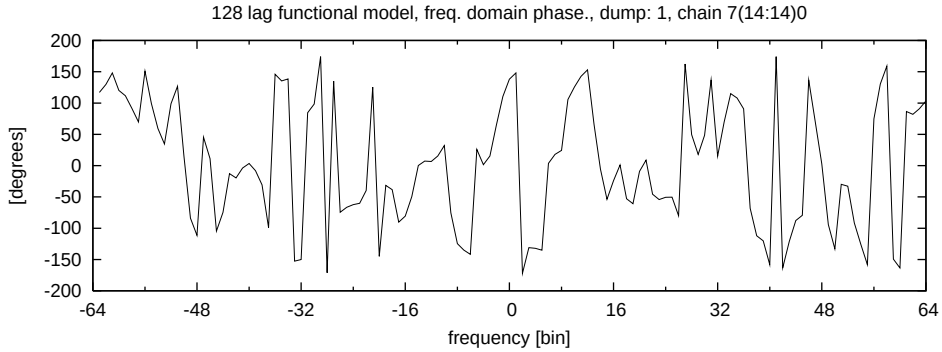

Illustration 2.7.151

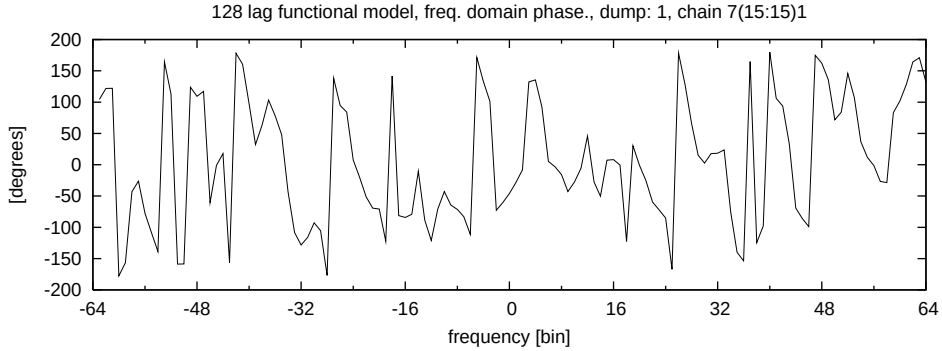

Illustration 2.7.152

```
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

01011001  CCCSCR_0_1:   CCC1: Y=pri X=pri   CCC0: Y=sec X=pri
01101010  CCCSCR_2_3:   CCC3: Y=pri X=sec   CCC2: Y=sec X=sec
11111111  CCCSCR_4_5:   CCC5: Y=n/c X=n/c   CCC4: Y=n/c X=n/c
11111111  CCCSCR_6_7:   CCC7: Y=n/c X=n/c   CCC6: Y=n/c X=n/c
01011001  CCCSCR_8_9:   CCC9: Y=pri X=pri   CCC8: Y=sec X=pri
01101010  CCCSCR_10_11: CCC11: Y=pri X=sec CCC10: Y=sec X=sec
01011001  CCCSCR_12_13: CCC13: Y=pri X=pri CCC12: Y=sec X=pri
01101010  CCCSCR_14_15: CCC15: Y=pri X=sec CCC14: Y=sec X=sec

00000011  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=sec XSw0=sec
11001111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00111100  XSTATUS: X-SE_CLK: 00111100
00111100  YSTATUS: Y-SE_CLK: 00111100
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.7.16: content of file mcb_reg68.txt.

2.8 extreme dump rate operation

Test cases: 07

2.8.1 organization

This section contains a single test case which validates operation at extreme (near minimum specification) dump rates. Validation is documented by the following:

(1) a post simulation synopsis;

(2) optional notes, if any, that detail outstanding features or conditions of the post simulation result;

(3) a *post simulation log* that documents script execution;

(4) the log generated by function *check_register_capture*;

(5) a table of dump overrun status bit states (**OVR**) associated with each cell for a given dump;

(6) a table of the bias associated with cells of each stream (four streams each dumping at a different rate);

(7) a table of bias for each cell in all dumps;

(8) content of the MCB register file that is provided by the test bench.

2.8.2 test case 07

Synopsis: Cells having long dump intervals (quads 2, 3) generated lag frames normally, without overrun, along side those experiencing extreme dump overrun (quads 0, 1) where several dumps occur in succession without generating frames. The RTL model did not enter a state from which it could not exit without external intervention (such as a reset), as expected. Dump overrun (**OVR**) bits in the output frame were set as expected in Table 2.8.1 and the overrun flag in the **MCSR** was set (Text 2.8.2 and Text 2.8.3). Advisories in the register and captured data log (Text 2.8.2) prompted the generation of a table of bias for all cells and dumps (Table 2.8.3).

Notes:

(1) A comprehensive bit exact test between cells of similar dump intervals and overrun bit states was not performed. However, a successful ad hoc bit exact comparison between a pair of similarly configured cells with overrun set and a second involving two cells with the overrun bit reset are included in the output log Text 2.8.1 .

(2) Advisories reported in the log generated by function *check_register_capture* is likely the result of a bug in the function since the bias tabulated in Table 2.8.3 does not support this log report.

```
***--------------------------------------------------------------------------***
tc07_postsim.m : V3.00 run time: Mon Dec 12 14:57:55 2005
Using postsim_lib: 1.40: 05-08-18

79 frames read for test case 07.
test case version: v4,05dec09
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
0(0:0)0  0(1:1)0  0(2:2)0  0(3:3)0
2(4:4)2  2(5:5)2  2(6:6)2  2(7:7)2
4(8:8)4  4(9:9)4  4(10:10)4  4(11:11)4
6(12:12)6  6(13:13)6  6(14:14)6  6(15:15)6


Initial (dump= 1) timestamp (word 0): 222333444
Missing timestamps (not increment by 1).

ADVISORIES encountered during register status and data capture checks;
                       details in log file.

Check OK: DESSR bits are set as expected.

Number of dumps: 37

Using list_dump_ovr: 1.10: 05-07-25
Requested number of dumps of cells dumping at the highest rate: 80
Requested number of dumps of cells dumping at the highest rate/10: 20
Requested number of dumps of cells dumping at the highest rate/50: 4
Requested number of dumps of cells dumping at the highest rate/100: 2

Write dump overrun tables.
***--------------------------------------------------------------------------***
octave:3> # demonstrate bit exact cell similarity (tc07_dump_overrun-f1.txt)
octave:3> dump07(13).TIMESTAMP_0
ans = 222333463
octave:4> dump07(13).CCchain(1).CCC
ans = 4
octave:5> dump07(13).CCchain(1).OVR
ans = 1
octave:6> find((dump07(13).CCchain(1).clag-dump07(13).CCchain(2).clag)> 10*eps)
ans = [](0x0)
octave:7> dump07(13).CCchain(3).CCC
ans = 8
octave:8> dump07(13).CCchain(3).OVR
ans = 0
octave:9> find((dump07(13).CCchain(3).clag-dump07(13).CCchain(4).clag)> 10*eps)
ans = [](0x0)
octave:10>
```

Text 2.8.1 console log for script tc07_postsim followed by ad hoc tests.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
                frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

ADVISORY from check_register_capture: integration dead time differs
        between streams; on cells reporting no overrun.

ADVISORY from check_register_capture: integration dead time differs
        between streams; on cells reporting overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

Check OK: Captured X,Y ID data (SID, SBID, BBID) are set
          to expected values.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                             template of expected states.
```

Text 2.8.2: test case 07; content of the register and capture log file.

timestamp | cell numb	*0 1 2 3*	*4 5 6 7*	*8 9 10 11*	*12 13 14 15*
222333444	0 0 0 0			
222333446	1 x x x			
222333447	x x x x	0 0 0 0		
222333448	x 1 x x			
222333449	x x 1 x			
222333450	x x x 1			
222333451	x x x x	0 x x x		
222333455	x x x x	x 1 1 1		
222333456	1 x x x			
222333458	x 1 x x			
222333459	x x 1 x	x x x x		
222333460	x x x 1			
222333463	x x x x	1 1 x x	0 0 0 0	
222333467	x x x x	x x 1 1		
222333471	1 x x x	x x x x		
222333472	x 1 x x			
222333474	x x 1 x			
222333475	x x x 1	x x x x		
222333479	x x x x	1 1 1 x		
222333481	1 x x x			
222333482	x 1 x x			
222333483	x x x x	x x x 1	0 0 0 0	0 0 0 0
222333484	x x 1 x			
222333485	x x x 1			
222333487	x x x x	1 1 x x		
222333491	x x x x	x x 1 1		
222333501	1 x x x			
222333502	x 1 x x			
222333503	x x 1 x	x x x x	0 0 0 0	
222333504	x x x 1			
222333507	x x x x	1 1 x x		
222333511	x x x x	x x 1 1		
222333516	1 x x x			
222333517	x 1 x x			
222333518	x x 1 x			
222333519	x x x 1	x x x x		
222333523	x x x x	1 1 1 x	0 0 0 0	0 0 0 0

Table 2.8.1: test case 07; dump overrun bit status where '1' and '0' designates set and reset respectively; 'x' indicates that no frame was generated for a particular cell and dump; timestamps for which no frames are generated are excluded.

stream	*applied dump*	*bias: !OVR*	*dump-bias: !OVR*	*bias: OVR*	*dump-bias: OVR*
0	500	224	276	496	4
2	2000	1724	276	1996	4
4	10000	9724	276	0	n-a
6	20000	19724	276	0	n-a

Table 2.8.2 test case 07; applied dump and bias.

timestamp | cell number	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
222333444	224	224	224	224												
222333446	**496**															
222333447					1724	1724	1724	1724								
222333448		**496**														
222333449			**496**													
222333450				**496**												
222333451					1724											
222333455						**1996**	**1996**	**1996**								
222333456	**496**															
222333458		**496**														
222333459			**496**													
222333460				**496**												
222333463					**1996**	**1996**			9724	9724	9724	9724				
222333467							**1996**	**1996**								
222333471	**496**															
222333472		**496**														
222333474			**496**													
222333475				**496**												
222333479					**1996**	**1996**	**1996**									
222333481	**496**															
222333482		**496**														
222333483								**1996**	9724	9724	9724	9724	19724	19724	19724	19724
222333484			**496**													
222333485				**496**												
222333487					**1996**	**1996**										
222333491							**1996**	**1996**								
222333501	**496**															
222333502		**496**														
222333503			**496**						9724	9724	9724	9724				
222333504				**496**												
222333507					**1996**	**1996**										
222333511							**1996**	**1996**								
222333516	**496**															
222333517		**496**														
222333518			**496**													
222333519				**496**												
222333523					**1996**	**1996**	**1996**		9724	9724	9724	9724	19724	19724	19724	19724

Table 2.8.3: test case 07; bias for each cell in all dumps; bold face indicates that the overrun bit (OVR) is set.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10001101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=1 PhEN=1 TVEN=0 CEN=1

10011001  CCCSCR_0_1:   CCC1: Y=sec X=pri   CCC0: Y=sec X=pri
10011001  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=sec X=pri
10011001  CCCSCR_4_5:   CCC5: Y=sec X=pri   CCC4: Y=sec X=pri
10011001  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=sec X=pri
10011001  CCCSCR_8_9:   CCC9: Y=sec X=pri   CCC8: Y=sec X=pri
10011001  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=sec X=pri
10011001  CCCSCR_12_13: CCC13: Y=sec X=pri CCC12: Y=sec X=pri
10011001  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=sec X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
10101010  XSTATUS: X-SE_CLK: 10101010
10101010  YSTATUS: Y-SE_CLK: 10101010
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.8.3 content of file mcb_reg07.txt.

2.9 overrun detection and recovery

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

2.9.1 organization

Tests of this section validate operation at dump rates that result in dump overruns for which the model operates in a well specified (according to [RFS-00]) manner. Validation is documented by the following:

(1) a post simulation synopsis;

(2) optional notes, if any, that detail outstanding features or conditions of the post simulation result;

(3) a *post simulation log* that documents script execution;

(4) the log generated by function *check_register_capture*, if warnings or advisories were issued;

(5) a table of dump overrun status bit states (**OVR**) associated with each cell for a given dump;

(6) a table of bias (expressed in number of samples) accumulated by cells associated with each stream;

(7) test case 24 only; statistical parameters of the difference between lag coefficients of identically configured chains;

(8) content of the MCB register file that is provided by the test bench.

The quantities of Table 2.9.3 are labelled based on the nomenclature presented in 2.2. Thus,

$\overline{\Re[f_1 - f_2]}$: is the mean of the real part of $[f_1 - f_2]$,

$\overline{\Im[f_1 - f_2]}$: is the mean of the imaginary part of $[f_1 - f_2]$,

$\sigma_{\Re[f_1 - f_2]}$: is the standard deviation of the real part of $[f_1 - f_2]$,

$\sigma_{\Im[f_1 - f_2]}$: is the standard deviation of the real part of $[f_1 - f_2]$

where the subscripts 1 and 2 refer to the first and second chain, respectively, of a pair.

2.9.2 test case 24

Synopsis: No advisories or warnings regarding captured data or register content were logged. The expected dump overrun bit (**OVR**) states were observed and are shown in Table 2.9.1. Table 2.9.3 shows low values of standard deviation and mean of the difference in lag coefficients between chains that are identically configured but having different overrun conditions; zero values in the case of identical overrun bit states.

Notes: Warnings in the console log Text 2.5.15 are issued from the function which concatenates cells based on status flags in the output frame and result from the absence of cells or mismatched OVR bit states within a chain of a given dump; these warnings are of no consequence within the context of this test case.

```
***-------------------------------------------------------------------------***
tc24_postsim.m : V2.00 run time: Sun Sep 25 14:58:32 2005
Using postsim_lib: 1.40: 05-08-18

133 frames read for test case 24.
test case version: v3,05apr25
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


WARNING from add_CCchain_dump:
        Lost chains on some dumps; number of concatenated cell
        chain starts does not equal the number of ends.

WARNING from add_CCchain_dump: OVR status bits
  differ between cells of some chains.
  data_bias= 0 indicates that data bias has been applied idependently
  to each cell of the chain.


Initial (dump= 1) timestamp (word 0): 222333444
Timestamps are sequential and increment by 1.

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Number of dumps: 10
Number of dumps of cells dumping at the highest rate: 10

Discarding isolated (unpaired) cells from each dump.

Concatenate chains and scale data regardless of OVR bit status.
Cells with mixed OVR status bits within a chain will have
the data bias applied independently to each cell.
A single DVCOUNT will be applied to the chain.


Write dump overrun tables.
***-------------------------------------------------------------------------***

octave:67> tc24_chain_diff_tables
Write table to file: tc24_chain_diff.txt.
```

Text 2.9.1 console log for scripts tc24_postsim, tc24_chain_diff_tables.

timestamp | cell number	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
222333444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
222333445	0	0	0	0	0	0	0	0	x	x	x	x	0	0	0	0
222333446	0	0	0	0	0	x	x	x	1	1	1	1	0	0	0	0
222333447	0	0	x	x	x	1	1	1	0	0	0	0	0	0	0	0
222333448	x	x	1	1	1	0	0	0	0	0	0	0	0	0	0	0
222333449	1	1	0	0	0	0	0	0	0	0	0	0	0	x	x	x
222333450	0	0	0	0	0	0	0	0	0	0	x	x	x	1	1	1
222333451	0	0	0	0	0	0	0	x	x	x	1	1	1	0	0	0
222333452	0	0	0	0	x	x	x	1	1	1	0	0	0	0	0	0
222333453	0	x	x	x	1	1	1	0	0	0	0	0	0	0	0	0

Table 2.9.1 test case 24; dump overrun bit status where '1' and '0' designates set and reset respectively; 'x' indicates that no frame was generated for a particular cell and dump.

stream	*applied dump*	*bias: !OVR*	*dump-bias: !OVR*	*bias: OVR*	*dump-bias: OVR*
0	8000	7724	276	7996	4
1	8000	7724	276	7996	4

Table 2.9.2 test case 24; applied dump and bias.

chain 1	*chain 2*	*timestamp*	*OVR chain1*	*OVR chain2*	$\Re\left[\overline{f_1 - f_2}\right]$	$\Im\left[\overline{f_1 - f_2}\right]$	$\sigma_{\Re[f_1 - f_2]}$	$\sigma_{\Im[f_1 - f_2]}$
0:1	8:9	222333444	0:0	0:0	0.0000	0.0000	0.0000	0.0000
2:3	10:11	222333444	0:0	0:0	0.0000	0.0000	0.0000	0.0000
4:5	12:13	222333444	0:0	0:0	0.0000	0.0000	0.0000	0.0000
6:7	14:15	222333444	0:0	0:0	0.0000	0.0000	0.0000	0.0000
4:5	12:13	222333445	0:0	0:0	0.0000	0.0000	0.0000	0.0000
6:7	14:15	222333445	0:0	0:0	0.0000	0.0000	0.0000	0.0000
0:1	8:9	222333446	0:0	1:1	0.0003	-0.0005	0.0151	0.0154
2:3	10:11	222333446	0:0	1:1	0.0000	0.0002	0.0116	0.0140
0:1	8:9	222333447	0:0	0:0	0.0000	0.0000	0.0000	0.0000
6:7	14:15	222333447	1:1	0:0	-0.0002	0.0001	0.0162	0.0140
2:3	10:11	222333448	1:1	0:0	0.0001	-0.0002	0.0139	0.0120
4:5	12:13	222333448	1:0	0:0	0.0012	0.0004	0.0131	0.0110
6:7	14:15	222333448	0:0	0:0	0.0000	0.0000	0.0000	0.0000
0:1	8:9	222333449	1:1	0:0	0.0002	0.0011	0.0141	0.0126
2:3	10:11	222333449	0:0	0:0	0.0000	0.0000	0.0000	0.0000
0:1	8:9	222333450	0:0	0:0	0.0000	0.0000	0.0000	0.0000
6:7	14:15	222333450	0:0	1:1	0.0001	-0.0016	0.0162	0.0122
2:3	10:11	222333451	0:0	1:1	0.0000	-0.0009	0.0141	0.0177
4:5	12:13	222333451	0:0	1:0	-0.0010	-0.0006	0.0106	0.0119
0:1	8:9	222333452	0:0	1:1	0.0002	0.0003	0.0115	0.0140
2:3	10:11	222333452	0:0	0:0	0.0000	0.0000	0.0000	0.0000
4:5	12:13	222333453	1:1	0:0	0.0004	-0.0000	0.0161	0.0176
6:7	14:15	222333453	1:0	0:0	0.0000	-0.0002	0.0135	0.0137

Table 2.9.3 test case 24; statistical parameters of the difference between lag coefficients of identically configured chains.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10001101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=1 PhEN=1 TVEN=0 CEN=1

01000010  CCCSCR_0_1:   CCC1: Y=pri X=adj   CCC0: Y=adj X=sec
10000001  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=pri
01000001  CCCSCR_4_5:   CCC5: Y=pri X=adj   CCC4: Y=adj X=pri
10000010  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=sec
01000010  CCCSCR_8_9:   CCC9: Y=pri X=adj   CCC8: Y=adj X=sec
10000001  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=pri
01000001  CCCSCR_12_13: CCC13: Y=pri X=adj CCC12: Y=adj X=pri
10000010  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=sec

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11110000  XSTATUS: X-SE_CLK: 11110000
11110000  YSTATUS: Y-SE_CLK: 11110000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.9.2 content of file mcb_reg24.txt.

2.9.3 test case 29

Synopsis: No advisories or warnings regarding captured data or register content were logged. The expected dump overrun bit (**OVR**) states were observed and are shown in Table 2.9.4 . Cells having identical overrun status and input streams passed a bit exact comparison (Text 2.9.3) as expected.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc29_postsim.m : V2.00 run time: Sun Sep 25 19:01:06 2005
Using postsim_lib: 1.40: 05-08-18

87 frames read for test case 29.
test case version: v2,04apr11
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
0(0:0)0  0(1:1)0  0(2:2)0  0(3:3)0
2(4:4)2  2(5:5)2  2(6:6)2  2(7:7)2
4(8:8)4  4(9:9)4  4(10:10)4  4(11:11)4
6(12:12)6  6(13:13)6  6(14:14)6  6(15:15)6


Initial (dump= 1) timestamp (word 0): 222333444
Missing timestamps (not increment by 1).

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Number of dumps: 15

Using list_dump_ovr: 1.10: 05-07-25
Requested number of dumps of cells dumping at the highest rate: 16
Requested number of dumps of cells dumping at the highest rate/2: 8
Requested number of dumps of cells dumping at the highest rate/4: 4
Requested number of dumps of cells dumping at the highest rate/8: 2

Write dump overrun tables.
***--------------------------------------------------------------------------***

octave:137> tc29_chain_similarity_check

No differences exceeding the floating point precision limit of this
machine were found between chains for any of the 15 dumps; all chains generated
identical lag data.
```

Text 2.9.3 console log for scripts, tc29_postsim, tc29_chain_similarity_check.

timestamp | cell number	***0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15***
222333444	0 0 0 0
222333445	0 0 0 0 0 0 0 0
222333446	0 0 0 0
222333447	x x x x 0 0 0 0 0 0 0 0
222333448	1 1 1 1
222333449	x x x x 0 0 0 0
222333450	1 1 1 x
222333451	x x x 1 0 0 0 0 0 0 0 0 0 0 0 0
222333452	1 1 1 0
222333453	x x x x 0 0 0 0
222333455	1 1 1 x x x x x 0 0 0 0
222333456	x x x 1
222333457	1 x x x 1 1 1 1
222333458	x 1 1 1
222333459	1 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0

Table 2.9.4 test case 29; dump overrun bit status where '1' and '0' designates set and reset respectively; 'x' indicates that no frame was generated for a particular cell and dump.

stream	***applied dump***	***bias: !OVR***	***dump-bias: !OVR***	***bias: OVR***	***dump-bias: OVR***
0	3000	2724	276	2996	4
2	6000	5724	276	5996	4
4	12000	11724	276	0	n-a
6	24000	23724	276	0	n-a

Table 2.9.5 test case 29; applied dump and bias.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10001101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=1 PhEN=1 TVEN=0 CEN=1

10011001  CCCSCR_0_1:   CCC1: Y=sec X=pri   CCC0: Y=sec X=pri
10011001  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=sec X=pri
10011001  CCCSCR_4_5:   CCC5: Y=sec X=pri   CCC4: Y=sec X=pri
10011001  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=sec X=pri
10011001  CCCSCR_8_9:   CCC9: Y=sec X=pri   CCC8: Y=sec X=pri
10011001  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=sec X=pri
10011001  CCCSCR_12_13: CCC13: Y=sec X=pri CCC12: Y=sec X=pri
10011001  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=sec X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.9.4 content of file mcb_reg29.txt

2.9.4 test case 31

Synopsis: No advisories or warnings regarding captured data or register content were logged. The expected dump overrun bit (**OVR**) states were observed and are shown in Table 2.9.6.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc31_postsim.m : V2.00 run time: Sun Oct 02 15:26:46 2005
Using postsim_lib: 1.40: 05-08-18

200 frames read for test case 31.
test case version: v3,05apr23
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
1(0:0)1  1(1:1)0  0(2:2)1  0(3:3)0
3(4:4)3  3(5:5)2  2(6:6)3  2(7:7)2
5(8:8)5  5(9:9)4  4(10:10)5  4(11:11)4
7(12:12)7  7(13:13)6  6(14:14)7  6(15:15)6


Initial (dump= 1) timestamp (word 0): 222333444
Missing timestamps (not increment by 1).

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Number of dumps: 31

Using list_dump_ovr: 1.10: 05-07-25
Requested number of dumps of cells dumping at the highest rate: 32
Requested number of dumps of cells dumping at the highest rate/2: 16
Requested number of dumps of cells dumping at the highest rate/4: 8
Requested number of dumps of cells dumping at the highest rate/8: 4

Write dump overrun tables.
***--------------------------------------------------------------------------***
```

Text 2.9.5: console log for tc31_postsim.txt.

timestamp | cell number	*0 1 2 3*	*4 5 6 7*	*8 9 10 11*	*12 13 14 15*
222333444	0 0 0 0			
222333445	0 0 0 0	0 0 0 0		
222333446	0 0 0 0			
222333447	0 0 0 0	0 0 0 0	0 0 0 0	
222333448	0 0 0 0			
222333449	x x x x	0 0 0 0		
222333450	1 1 1 1			
222333451	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
222333452	0 0 0 x			
222333453	x x x 1	0 0 0 0		
222333454	1 1 1 x			
222333455	0 x x 1	0 0 0 0	0 0 0 0	
222333456	x 1 1 0			
222333457	1 0 x x	0 0 0 0		
222333458	x x 1 1			
222333459	1 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0
222333461	x x x x	0 0 0 0		
222333462	1 1 1 1			
222333463	0 0 0 0	0 0 0 0	0 0 0 0	
222333464	0 0 x x			
222333465	x x 1 1	0 0 0 0		
222333466	1 1 0 0			
222333467	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
222333468	0 x x x			
222333469	x 1 1 1	0 0 0 x		
222333470	1 0 0 x			
222333471	0 0 x 1	0 0 0 1	0 0 0 0	
222333472	x x 1 0			
222333473	1 1 0 x	0 0 0 0		
222333474	0 x x 1			
222333475	0 1 1 0	0 0 0 0	0 0 0 0	0 0 0 0

Table 2.9.6: test case 31; dump overrun bit status where '1' and '0' designates set and reset respectively; 'x' indicates that no frame was generated for a particular cell and dump.

stream	*applied dump*	*bias:!OVR*	*dump-bias:!OVR*	*bias:OVR*	*dump-bias:OVR*
0	4000	3724	276	3996	4
1	4000	3724	276	3996	4
2	8000	7724	276	7996	4
3	8000	7724	276	0	n-a
4	16000	15724	276	0	n-a
5	16000	15724	276	0	n-a
6	32000	31724	276	0	n-a
7	32000	31724	276	0	n-a

Table 2.9.7: test case 31; applied dump and bias.

```
# ------------------------------------------------------------------------------
# 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.
#
# ------------------------------------------------------------------------------
10001101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=1 PhEN=1 TVEN=0 CEN=1

10100110  CCCSCR_0_1:   CCC1: Y=sec X=sec   CCC0: Y=pri X=sec
10010101  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=pri X=pri
10100110  CCCSCR_4_5:   CCC5: Y=sec X=sec   CCC4: Y=pri X=sec
10010101  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=pri X=pri
10100110  CCCSCR_8_9:   CCC9: Y=sec X=sec   CCC8: Y=pri X=sec
10010101  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=pri X=pri
10100110  CCCSCR_12_13: CCC13: Y=sec X=sec CCC12: Y=pri X=sec
10010101  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=pri X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.9.6: content of file mcb_reg31.txt.

2.9.5 test case 69

Synopsis: overrun bits for cells 0 to 3 dumping at extreme rates and cells 4 to 5 are set as expected; see Table 2.9.8. Statistical parameters of the behavioural - RTL model difference as defined in 2.5.1 are found in Table 2.9.10 to Table 2.9.12. The centre lag of cells 14 and 15 exhibit an accumulator overflow (see Text 2.9.7) which in turn set the **AOL** bit of the **MCSR**. This accumulator overflow was unexpected, hence the warning in the *check_register_capture* log of Text 2.9.8. A successful bit comparison between lag coefficients of similarly configured cells with identical overrun bit state is logged in Text 2.9.7.

Notes: the range of the accumulator was extended by adding 2^{23} to the centre lag coefficients of cells 14 and 15 before generating the behavioural-RTL model difference parameters.

```
***--------------------------------------------------------------------------***
tc69_postsim.m : V1.00 run time: Sun Dec 11 17:21:14 2005
Using postsim_lib: 1.40: 05-08-18

217 frames read for test case 69.
test case version: v1,05dec09
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
0(0:0)0  0(1:1)0  1(2:2)1  1(3:3)1
2(4:4)2  2(5:5)2  3(6:6)3  3(7:7)3
4(8:8)4  4(9:9)4  5(10:10)5  5(11:11)5
6(12:12)6  6(13:13)6  7(14:14)7  7(15:15)7

NOTE: Accumulator rollover at centre lag of chain 7 fixed by adding 2^23.

Initial (dump= 1) timestamp (word 0): 222333444
Missing timestamps (not increment by 1).

WARNINGS encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.10: 05-10-02
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics for chains 7 through 16; using name: tc69
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
octave:208> tc69_ACC_OV_scan

Bit ACC_OV set in ouput frame of cell 14, dump 82.
Bit ACC_OV set in ouput frame of cell 15, dump 82.

octave:209> tc69_make_dump_ovr_list

Number of dumps: 82

Using list_dump_ovr: 1.10: 05-07-25
Requested number of dumps of cells dumping at the highest rate: 128
Requested number of dumps of cells dumping at the highest rate/2: 64
Requested number of dumps of cells dumping at the highest rate/4: 32
Requested number of dumps of cells dumping at the highest rate/8: 16

octave:210> tc69_dump_overrun_tables
octave:211> tc69_chain_similarity_check


NOTICE: No differences exceeding the floating point precision
        limit of this machine were found between cells having similar
        OVR bit states and identical streams for any of the 82 dumps.
```

Text 2.9.7: console log for scripts tc69_postsim, tc69_ACC_OV_scan, tc69_make_dump_ovr_list, tc69_dump_overrun_tables, tc69_chain_similarity_check.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

Check OK: Captured X,Y ID data (SID, SBID, BBID) are set
          to expected values.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

WARNING from check_register_capture: MCSR.AOV differs
               from template.
```

Text 2.9.8: test case 69; check register and capture log; WARNING noted.

timestamp | cell number	*0 1*	*2 3*	*4 5*	*6 7*
222333444	0 0			
222333445	x x	0 0		
222333446	1 1			
222333447	x x	0 0	0 0	
222333448	1 x			
222333449	x 1	0 x		
222333451	x x	x 1	0 0	0 0
222333452	1 x			
222333453	x 1	1 x		
222333455	x x	x 1	0 0	
222333457	1 x	x x		
222333458	x 1			
222333459	x x	1 1	0 x	0 0
222333463	1 x	x x	x 1	
222333464	x 1			
222333465	x x	1 1		
222333467	1 x	x x	1 0	0 0
222333468	x 1			
222333469	x x	1 1		
222333471	x x	x x	0 0	
222333472	1 x			
222333473	x 1	1 x		
222333475	x x	x 1	0 0	0 0
222333476	1 x			
222333477	x 1	1 x		
222333479	x x	x 1	0 0	
222333483	1 x	x x	x x	0 0
222333484	x 1			
222333485	x x	1 1		
222333487	x x	x x	1 1	
222333488	1 x			
222333489	x 1	x x		
222333491	x x	1 1	0 0	0 0
222333492	1 x			
222333493	x 1	0 x		
222333495	x x	x 1	0 0	
222333498	1 x			
222333499	x 1	x x	x x	0 0
222333501	x x	1 1		
222333503	1 x	x x	1 1	
222333504	x 1			
222333505	x x	1 1		
222333507	1 x	x x	0 0	0 0
222333508	x 1			
222333509	x x	1 1		
222333511	x x	x x	0 0	

timestamp | cell number	*0 1*	*2 3*	*4 5*	*6 7*
222333515	x x	x x	x x	0 0
222333516	1 1			
222333517	x x	1 x		
222333519	x x	x 1	1 1	
222333521	1 1	x x		
222333523	x x	1 1	0 x	0 0
222333527	1 x	x x	x 1	
222333528	x 1			
222333529	x x	1 1		
222333531	1 x	x x	1 0	0 0
222333532	x 1			
222333533	x x	1 1		
222333535	x x	x x	0 0	
222333536	1 x			
222333537	x 1	1 x		
222333539	x x	x 1	0 0	0 0
222333540	1 1			
222333541	x x	1 x		
222333543	x x	x 1	0 0	
222333547	1 x	x x	x x	0 0
222333548	x 1			
222333549	x x	1 1		
222333551	x x	x x	1 1	
222333552	1 x			
222333553	x 1	1 x		
222333555	x x	x 1	0 0	0 0
222333556	1 x			
222333557	x 1	1 x		
222333559	x x	x 1	0 0	
222333562	1 x			
222333563	x 1	1 x	x x	0 0
222333565	x x	x 1		
222333567	1 x	x x	1 1	
222333568	x 1			
222333569	x x	1 1		
222333571	1 x	x x	0 0	0 0

Table 2.9.8: test case 69; table of dump overrun bit states.

stream	*dump*	*bias:!OVR*	*dump-bias:!OVR*	*bias:OVR*	*dump-bias:OVR*
0	1000	724	276	996	4
1	2000	1724	276	1996	4
2	4000	3724	276	3996	4
3	8000	7724	276	0	n-a
4	16000	15724	276	0	n-a
5	32000	31724	276	0	n-a
6	64000	63724	276	0	n-a
7	128000	127724	276	0	n-a

Table 2.9.9: test case 69; applied dump interval and bias.

chain	*dump*	$\overline{\Re[\,f-b\,]}$	$\overline{\Im[\,f-b\,]}$	*range* $\Re[\,f-b\,]$	*range* $\Im[\,f-b\,]$	$\sigma_{\Re[\,f-b\,]}$	$\sigma_{\Im[\,f-b\,]}$	$\frac{\sigma_{\Re[\,f-b\,]}}{\sigma_N}$	$\frac{\sigma_{\Im[\,f-b\,]}}{\sigma_N}$	*peak count*
7	56	8.23e-03	2.32e-03	0.1727	0.2172	0.0317	0.0353	1.29	1.44	1
7	62	-3.88e-03	-4.78e-03	0.1882	0.1778	0.0309	0.0332	1.26	1.35	2
7	66	2.00e-03	1.75e-03	0.1520	0.1923	0.0277	0.0361	1.13	1.47	0
7	72	-2.07e-04	-2.68e-03	0.1723	0.1904	0.0345	0.0369	1.41	1.51	0
7	77	1.03e-03	5.44e-03	0.1614	0.2338	0.0328	0.0431	1.34	1.76	0
7	82	7.36e-04	-3.71e-03	0.1653	0.1784	0.0368	0.0322	1.50	1.32	2
8	7	-5.22e-03	-2.53e-03	0.1789	0.2004	0.0388	0.0411	1.58	1.68	0
8	13	2.30e-03	4.42e-03	0.2028	0.1969	0.0389	0.0377	1.59	1.54	0
8	17	-9.36e-04	-2.41e-03	0.1687	0.1721	0.0327	0.0354	1.33	1.45	0
8	23	1.74e-03	5.77e-04	0.1778	0.1792	0.0378	0.0324	1.54	1.32	0
8	27	-4.59e-03	3.05e-03	0.1676	0.1737	0.0363	0.0302	1.48	1.23	0
8	33	-1.82e-03	-1.12e-03	0.1617	0.2037	0.0361	0.0347	1.47	1.42	1
8	38	1.64e-03	-2.55e-04	0.1725	0.2129	0.0360	0.0430	1.47	1.76	0
8	43	-1.51e-03	1.07e-03	0.2406	0.1544	0.0391	0.0334	1.60	1.36	1
8	47	-6.76e-04	4.44e-04	0.1778	0.1309	0.0314	0.0301	1.28	1.23	1
8	52	1.81e-04	-5.25e-04	0.1677	0.1815	0.0327	0.0338	1.34	1.38	0
8	56	8.23e-03	2.32e-03	0.1727	0.2172	0.0317	0.0353	1.29	1.44	1
8	62	-3.88e-03	-4.78e-03	0.1882	0.1778	0.0309	0.0332	1.26	1.35	2
8	66	2.00e-03	1.75e-03	0.1520	0.1923	0.0277	0.0361	1.13	1.47	0
8	72	-2.07e-04	-2.68e-03	0.1723	0.1904	0.0345	0.0369	1.41	1.51	0
8	77	1.03e-03	5.44e-03	0.1614	0.2338	0.0328	0.0431	1.34	1.76	0
8	82	7.36e-04	-3.71e-03	0.1653	0.1784	0.0368	0.0322	1.50	1.32	2
9	13	1.22e-04	3.59e-04	0.1262	0.0761	0.0218	0.0167	1.26	0.96	1
9	23	-1.82e-03	-6.76e-04	0.0636	0.0914	0.0157	0.0191	0.91	1.10	0
9	33	3.06e-03	6.08e-04	0.1111	0.0719	0.0212	0.0153	1.23	0.88	0
9	43	2.65e-05	-1.50e-03	0.0731	0.0793	0.0148	0.0152	0.86	0.88	0
9	52	-1.80e-04	-1.25e-04	0.0885	0.0784	0.0187	0.0161	1.08	0.93	0
9	62	-4.99e-04	7.62e-04	0.0864	0.0831	0.0163	0.0190	0.94	1.10	0
9	72	2.43e-04	1.27e-03	0.1019	0.0829	0.0164	0.0164	0.95	0.94	1
9	82	-1.44e-03	-1.45e-03	0.0834	0.0767	0.0154	0.0154	0.89	0.89	0
10	13	1.22e-04	3.59e-04	0.1262	0.0761	0.0218	0.0167	1.26	0.96	1
10	23	-1.82e-03	-6.76e-04	0.0636	0.0914	0.0157	0.0191	0.91	1.10	0

chain	*dump*	$\overline{\Re[f-b]}$	$\overline{\Im[f-b]}$	*range* $\Re[f-b]$	*range* $\Im[f-b]$	$\sigma_{\Re[f-b]}$	$\sigma_{\Im[f-b]}$	$\frac{\sigma_{\Re[f-b]}}{\sigma_N}$	$\frac{\sigma_{\Im[f-b]}}{\sigma_N}$	*peak count*
10	33	3.06e-03	6.08e-04	0.1111	0.0719	0.0212	0.0153	1.23	0.88	0
10	43	2.65e-05	-1.50e-03	0.0731	0.0793	0.0148	0.0152	0.86	0.88	0
10	52	-1.80e-04	-1.25e-04	0.0885	0.0784	0.0187	0.0161	1.08	0.93	0
10	62	-4.99e-04	7.62e-04	0.0864	0.0831	0.0163	0.0190	0.94	1.10	0
10	72	2.43e-04	1.27e-03	0.1019	0.0829	0.0164	0.0164	0.95	0.94	1
10	82	-1.44e-03	-1.45e-03	0.0834	0.0767	0.0154	0.0154	0.89	0.89	0
11	23	7.83e-05	-3.18e-04	0.0484	0.0474	0.0084	0.0087	0.69	0.71	1
11	43	-1.09e-04	1.10e-04	0.0463	0.0611	0.0098	0.0121	0.80	0.98	0
11	62	1.28e-03	-7.84e-04	0.0521	0.0649	0.0101	0.0118	0.82	0.96	1
11	82	-8.44e-04	5.25e-04	0.0532	0.0602	0.0104	0.0108	0.85	0.88	0
12	23	7.83e-05	-3.18e-04	0.0484	0.0474	0.0084	0.0087	0.69	0.71	1
12	43	-1.09e-04	1.10e-04	0.0463	0.0611	0.0098	0.0121	0.80	0.98	0
12	62	1.28e-03	-7.84e-04	0.0521	0.0649	0.0101	0.0118	0.82	0.96	1
12	82	-8.44e-04	5.25e-04	0.0532	0.0602	0.0104	0.0108	0.85	0.88	0
13	43	3.83e-04	-4.57e-04	0.0358	0.0294	0.0075	0.0060	0.87	0.69	0
13	82	9.74e-05	4.12e-04	0.0347	0.0350	0.0062	0.0063	0.72	0.73	2
14	43	3.83e-04	-4.57e-04	0.0358	0.0294	0.0075	0.0060	0.87	0.69	0
14	82	9.74e-05	4.12e-04	0.0347	0.0350	0.0062	0.0063	0.72	0.73	2
15	82	3.68e-04	6.02e-04	0.0296	0.0302	0.0050	0.0050	0.82	0.81	2
16	82	3.68e-04	6.02e-04	0.0296	0.0302	0.0050	0.0050	0.82	0.81	2

Table 2.9.10: test case 69; statistical parameters of the difference between RTL and behavioural model lag coefficients.

chain	*dump*	$\overline{\Re[\,f-f\,]}$	$\overline{\Im[\,f-f\,]}$	*range* $\Re[\,f-f\,]$	*range* $\Im[\,f-f\,]$	$\sigma_{\Re[\,f-f\,]}$	$\sigma_{\Im[\,f-f\,]}$	$\frac{\sigma_{\Re[\,f-f\,]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[\,f-f\,]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
7	7	2.26e-03	1.55e-03	0.4915	0.5066	0.0880	0.1113	0.91	1.01	null
7	13	-4.45e-03	-1.18e-04	0.4525	0.4603	0.0868	0.0959	0.81	1.04	null
7	17	5.99e-03	-2.79e-03	0.6070	0.4406	0.1086	0.0942	0.94	1.10	null
7	23	-9.56e-03	1.02e-02	0.5193	0.4535	0.1008	0.0807	1.01	1.05	null
7	27	1.12e-02	-1.51e-02	0.5288	0.4909	0.0963	0.0910	1.01	1.01	null
7	33	-1.47e-03	5.60e-03	0.5653	0.4714	0.1018	0.1015	0.99	1.06	null
7	38	2.09e-03	4.49e-03	0.5418	0.5041	0.1123	0.1010	1.19	1.05	null
7	43	-8.25e-04	-5.44e-03	0.5711	0.5455	0.1104	0.1086	1.10	1.02	null
7	47	-4.17e-03	2.35e-03	0.4489	0.3961	0.0944	0.0915	1.05	0.91	null
7	52	1.24e-02	5.19e-04	0.5250	0.4788	0.0993	0.0932	1.00	0.89	null
7	56	-8.32e-03	-7.07e-04	0.3828	0.4357	0.0949	0.0933	0.95	0.99	null
7	62	-2.66e-03	3.02e-03	0.5726	0.5316	0.1038	0.0965	1.03	1.17	null
7	66	1.23e-03	-3.55e-03	0.6790	0.4730	0.1319	0.0973	0.94	1.12	null
7	72	-4.46e-03	1.46e-03	0.5733	0.5272	0.1131	0.1064	0.92	1.02	null
7	77	7.12e-06	-4.17e-03	0.6300	0.5051	0.1112	0.0930	1.10	1.05	null
8	7	2.26e-03	1.55e-03	0.4915	0.5066	0.0880	0.1113	0.91	1.01	null
8	13	-4.45e-03	-1.18e-04	0.4525	0.4603	0.0868	0.0959	0.81	1.04	null
8	17	5.99e-03	-2.79e-03	0.6070	0.4406	0.1086	0.0942	0.94	1.10	null
8	23	-9.56e-03	1.02e-02	0.5193	0.4535	0.1008	0.0807	1.01	1.05	null
8	27	1.12e-02	-1.51e-02	0.5288	0.4909	0.0963	0.0910	1.01	1.01	null
8	33	-1.47e-03	5.60e-03	0.5653	0.4714	0.1018	0.1015	0.99	1.06	null
8	38	2.09e-03	4.49e-03	0.5418	0.5041	0.1123	0.1010	1.19	1.05	null
8	43	-8.25e-04	-5.44e-03	0.5711	0.5455	0.1104	0.1086	1.10	1.02	null
8	47	-4.17e-03	2.35e-03	0.4489	0.3961	0.0944	0.0915	1.05	0.91	null
8	52	1.24e-02	5.19e-04	0.5250	0.4788	0.0993	0.0932	1.00	0.89	null
8	56	-8.32e-03	-7.07e-04	0.3828	0.4357	0.0949	0.0933	0.95	0.99	null
8	62	-2.66e-03	3.02e-03	0.5726	0.5316	0.1038	0.0965	1.03	1.17	null
8	66	1.23e-03	-3.55e-03	0.6790	0.4730	0.1319	0.0973	0.94	1.12	null
8	72	-4.46e-03	1.46e-03	0.5733	0.5272	0.1131	0.1064	0.92	1.02	null
8	77	7.12e-06	-4.17e-03	0.6300	0.5051	0.1112	0.0930	1.10	1.05	null
9	13	-1.76e-03	-4.41e-03	0.2938	0.3012	0.0634	0.0613	0.93	1.02	null
9	23	1.90e-03	-4.04e-03	0.3576	0.3447	0.0678	0.0637	1.02	0.94	null
9	33	1.53e-03	1.64e-03	0.3827	0.3419	0.0741	0.0660	1.04	1.01	null
9	43	-8.01e-03	4.14e-03	0.3784	0.3287	0.0748	0.0686	1.04	0.95	null
9	52	-2.01e-03	-1.49e-03	0.3224	0.4353	0.0651	0.0729	1.04	1.01	null
9	62	4.79e-03	-1.51e-03	0.3660	0.3517	0.0621	0.0696	0.93	1.08	null
9	72	-1.62e-03	4.00e-03	0.3226	0.2880	0.0685	0.0612	1.00	1.00	null
10	13	-1.76e-03	-4.41e-03	0.2938	0.3012	0.0634	0.0613	0.93	1.02	null
10	23	1.90e-03	-4.04e-03	0.3576	0.3447	0.0678	0.0637	1.02	0.94	null
10	33	1.53e-03	1.64e-03	0.3827	0.3419	0.0741	0.0660	1.04	1.01	null
10	43	-8.01e-03	4.14e-03	0.3784	0.3287	0.0748	0.0686	1.04	0.95	null
10	52	-2.01e-03	-1.49e-03	0.3224	0.4353	0.0651	0.0729	1.04	1.01	null
10	62	4.79e-03	-1.51e-03	0.3660	0.3517	0.0621	0.0696	0.93	1.08	null
10	72	-1.62e-03	4.00e-03	0.3226	0.2880	0.0685	0.0612	1.00	1.00	null
11	23	1.77e-03	-2.25e-03	0.1996	0.2008	0.0407	0.0412	0.87	0.96	null

chain	*dump*	$\overline{\Re[f-f]}$	$\overline{\Im[f-f]}$	*range* $\Re[f-f]$	*range* $\Im[f-f]$	$\sigma_{\Re[f-f]}$	$\sigma_{\Im[f-f]}$	$\frac{\sigma_{\Re[f-f]}}{\sigma_{\Re[b-b]}}$	$\frac{\sigma_{\Im[f-f]}}{\sigma_{\Im[b-b]}}$	*co-dump peaks*
11	43	-8.80e-04	-2.01e-03	0.2597	0.1963	0.0505	0.0428	1.01	1.05	null
11	62	-2.24e-03	5.52e-03	0.2980	0.2600	0.0525	0.0485	1.02	1.07	null
12	23	1.77e-03	-2.25e-03	0.1996	0.2008	0.0407	0.0412	0.87	0.96	null
12	43	-8.80e-04	-2.01e-03	0.2597	0.1963	0.0505	0.0428	1.01	1.05	null
12	62	-2.24e-03	5.52e-03	0.2980	0.2600	0.0525	0.0485	1.02	1.07	null
13	43	-1.92e-03	-1.74e-03	0.1561	0.1639	0.0296	0.0325	1.03	0.98	null
14	43	-1.92e-03	-1.74e-03	0.1561	0.1639	0.0296	0.0325	1.03	0.98	null

Table 2.9.11: test case 69; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*
7	7	-7.78e-04	-1.76e-04	0.0396	0.0239	5.03e-03	3.58e-03	6.8	4.9	0
7	13	4.21e-04	2.28e-05	0.0327	0.0217	4.78e-03	3.59e-03	6.5	4.9	0
7	17	6.63e-04	5.56e-04	0.0230	0.0210	3.20e-03	3.50e-03	4.4	4.8	0
7	23	-4.23e-04	-3.60e-04	0.0245	0.0246	3.69e-03	4.22e-03	5.0	5.7	0
7	27	1.81e-04	7.35e-05	0.0255	0.0214	3.82e-03	3.50e-03	5.2	4.8	0
7	33	-3.14e-04	3.20e-04	0.0193	0.0227	3.70e-03	3.63e-03	5.0	4.9	0
7	38	-2.23e-04	-8.72e-04	0.0402	0.0182	4.81e-03	3.48e-03	6.5	4.7	0
7	43	-5.52e-04	8.94e-04	0.0383	0.0215	4.70e-03	3.31e-03	6.4	4.5	0
7	47	1.17e-03	3.93e-04	0.0149	0.0205	3.07e-03	3.52e-03	4.2	4.8	0
7	52	2.30e-04	6.72e-04	0.0208	0.0202	4.00e-03	3.17e-03	5.4	4.3	0
7	56	-5.01e-04	-8.68e-05	0.0251	0.0224	3.61e-03	3.41e-03	4.9	4.6	0
7	62	4.60e-04	-3.87e-04	0.0205	0.0224	3.47e-03	3.60e-03	4.7	4.9	0
7	66	-1.70e-04	-5.67e-05	0.0249	0.0161	3.91e-03	3.33e-03	5.3	4.5	0
7	72	7.28e-05	1.89e-04	0.0215	0.0176	4.05e-03	3.40e-03	5.5	4.6	0
7	77	2.58e-04	9.29e-07	0.0308	0.0176	4.68e-03	3.53e-03	6.4	4.8	0
7	82	4.65e-04	5.38e-04	0.0266	0.0174	4.19e-03	3.08e-03	5.7	4.2	0
8	7	-7.78e-04	-1.76e-04	0.0396	0.0239	5.03e-03	3.58e-03	6.8	4.9	0
8	13	4.21e-04	2.28e-05	0.0327	0.0217	4.78e-03	3.59e-03	6.5	4.9	0
8	17	6.63e-04	5.56e-04	0.0230	0.0210	3.20e-03	3.50e-03	4.4	4.8	0
8	23	-4.23e-04	-3.60e-04	0.0245	0.0246	3.69e-03	4.22e-03	5.0	5.7	0
8	27	1.81e-04	7.35e-05	0.0255	0.0214	3.82e-03	3.50e-03	5.2	4.8	0
8	33	-3.14e-04	3.20e-04	0.0193	0.0227	3.70e-03	3.63e-03	5.0	4.9	0
8	38	-2.23e-04	-8.72e-04	0.0402	0.0182	4.81e-03	3.48e-03	6.5	4.7	0
8	43	-5.52e-04	8.94e-04	0.0383	0.0215	4.70e-03	3.31e-03	6.4	4.5	0
8	47	1.17e-03	3.93e-04	0.0149	0.0205	3.07e-03	3.52e-03	4.2	4.8	0
8	52	2.30e-04	6.72e-04	0.0208	0.0202	4.00e-03	3.17e-03	5.4	4.3	0
8	56	-5.01e-04	-8.68e-05	0.0251	0.0224	3.61e-03	3.41e-03	4.9	4.6	0
8	62	4.60e-04	-3.87e-04	0.0205	0.0224	3.47e-03	3.60e-03	4.7	4.9	0
8	66	-1.70e-04	-5.67e-05	0.0249	0.0161	3.91e-03	3.33e-03	5.3	4.5	0
8	72	7.28e-05	1.89e-04	0.0215	0.0176	4.05e-03	3.40e-03	5.5	4.6	0
8	77	2.58e-04	9.29e-07	0.0308	0.0176	4.68e-03	3.53e-03	6.4	4.8	0
8	82	4.65e-04	5.38e-04	0.0266	0.0174	4.19e-03	3.08e-03	5.7	4.2	0
9	13	-7.19e-04	-1.89e-04	0.0152	0.0074	2.39e-03	1.53e-03	4.6	2.9	0
9	23	3.47e-04	-3.91e-04	0.0121	0.0119	1.98e-03	1.75e-03	3.8	3.4	0
9	33	-2.94e-04	4.75e-04	0.0118	0.0082	2.22e-03	1.45e-03	4.3	2.8	0
9	43	-1.58e-04	-7.77e-05	0.0095	0.0100	1.61e-03	1.64e-03	3.1	3.2	0
9	52	-1.81e-04	-1.96e-04	0.0166	0.0138	2.08e-03	1.82e-03	4.0	3.5	0
9	62	-2.77e-04	2.30e-04	0.0151	0.0116	2.10e-03	1.75e-03	4.0	3.4	0
9	72	1.70e-04	-9.61e-05	0.0103	0.0117	1.73e-03	1.63e-03	3.3	3.1	0
9	82	3.58e-04	-3.82e-07	0.0107	0.0083	1.87e-03	1.33e-03	3.6	2.6	0
10	13	-7.19e-04	-1.89e-04	0.0152	0.0074	2.39e-03	1.53e-03	4.6	2.9	0
10	23	3.47e-04	-3.91e-04	0.0121	0.0119	1.98e-03	1.75e-03	3.8	3.4	0
10	33	-2.94e-04	4.75e-04	0.0118	0.0082	2.22e-03	1.45e-03	4.3	2.8	0
10	43	-1.58e-04	-7.77e-05	0.0095	0.0100	1.61e-03	1.64e-03	3.1	3.2	0
10	52	-1.81e-04	-1.96e-04	0.0166	0.0138	2.08e-03	1.82e-03	4.0	3.5	0

chain	*dump*	$\overline{\Re[f^F - b^F]}$	$\overline{\Im[f^F - b^F]}$	*range* $\Re[f^F - b^F]$	*range* $\Im[f^F - b^F]$	$\sigma_{\Re[f^F - b^F]}$	$\sigma_{\Im[f^F - b^F]}$	$\frac{\sigma_{\Re[f^F - b^F]}}{\sigma_N}$	$\frac{\sigma_{\Im[f^F - b^F]}}{\sigma_N}$	*peak count*
10	62	-2.77e-04	2.30e-04	0.0151	0.0116	2.10e-03	1.75e-03	4.0	3.4	0
10	72	1.70e-04	-9.61e-05	0.0103	0.0117	1.73e-03	1.63e-03	3.3	3.1	0
10	82	3.58e-04	-3.82e-07	0.0107	0.0083	1.87e-03	1.33e-03	3.6	2.6	0
11	23	2.13e-04	-1.09e-05	0.0075	0.0044	1.09e-03	8.85e-04	3.0	2.4	0
11	43	1.25e-04	6.67e-05	0.0068	0.0046	1.26e-03	7.70e-04	3.4	2.1	0
11	62	1.79e-04	-6.46e-05	0.0119	0.0064	1.36e-03	9.17e-04	3.7	2.5	0
11	82	-2.28e-04	1.17e-04	0.0078	0.0066	1.23e-03	9.58e-04	3.4	2.6	0
12	23	2.13e-04	-1.09e-05	0.0075	0.0044	1.09e-03	8.85e-04	3.0	2.4	0
12	43	1.25e-04	6.67e-05	0.0068	0.0046	1.26e-03	7.70e-04	3.4	2.1	0
12	62	1.79e-04	-6.46e-05	0.0119	0.0064	1.36e-03	9.17e-04	3.7	2.5	0
12	82	-2.28e-04	1.17e-04	0.0078	0.0066	1.23e-03	9.58e-04	3.4	2.6	0
13	43	-1.37e-04	7.42e-07	0.0050	0.0031	8.71e-04	4.75e-04	3.4	1.8	0
13	82	3.61e-05	1.02e-04	0.0043	0.0030	6.93e-04	5.33e-04	2.7	2.1	0
14	43	-1.37e-04	7.42e-07	0.0050	0.0031	8.71e-04	4.75e-04	3.4	1.8	0
14	82	3.61e-05	1.02e-04	0.0043	0.0030	6.93e-04	5.33e-04	2.7	2.1	0
15	82	1.40e-05	1.48e-05	0.0034	0.0014	6.31e-04	3.07e-04	3.4	1.7	0
16	82	1.40e-05	1.48e-05	0.0034	0.0014	6.31e-04	3.07e-04	3.4	1.7	0

Table 2.9.12: test case 69; statistical parameters of the difference between RTL and behavioural model frequency domain coefficients.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10011101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=1 OVR=1 PhEN=1 TVEN=0 CEN=1

10011001  CCCSCR_0_1:   CCC1: Y=sec X=pri   CCC0: Y=sec X=pri
01100110  CCCSCR_2_3:   CCC3: Y=pri X=sec   CCC2: Y=pri X=sec
10011001  CCCSCR_4_5:   CCC5: Y=sec X=pri   CCC4: Y=sec X=pri
01100110  CCCSCR_6_7:   CCC7: Y=pri X=sec   CCC6: Y=pri X=sec
10011001  CCCSCR_8_9:   CCC9: Y=sec X=pri   CCC8: Y=sec X=pri
01100110  CCCSCR_10_11: CCC11: Y=pri X=sec CCC10: Y=pri X=sec
10011001  CCCSCR_12_13: CCC13: Y=sec X=pri CCC12: Y=sec X=pri
01100110  CCCSCR_14_15: CCC15: Y=pri X=sec CCC14: Y=pri X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.9.9: content of file mcb_reg69.txt.

2.9.6 test case 70

Synopsis: overrun bits are for cells 0 to 3 dumping at extreme rates and cells 4 to 5 are set as expected; see Table 2.9.13. The centre lag of cells 14 and 15 exhibit an accumulator overflow (see Text 2.9.10) which in turn set the **AOL** bit of the **MCSR**. This accumulator overflow was unexpected, hence the warning in the *check_register_capture* log of Text 2.9.11. A successful bit exact comparison between lag coefficients from similarly configured cells of test cases 69 and 70, having identical overrun bit state is logged in Text 2.9.10, hence, tables of statistical model difference parameters are not included. Statistical parameters of the behavioural - RTL model difference are found in Table 2.9.10 to Table 2.9.12 in the previous section.

Notes: the range of the accumulator was extended by adding 2^{23} to the centre lag coefficients of cells 14 and 15 before performing a bit exact comparison between the lag coefficients of test cases 69 and 70.

```
***--------------------------------------------------------------------------***
tc70_postsim.m : V1.00 run time: Sun Dec 11 18:57:14 2005
Using postsim_lib: 1.40: 05-08-18

217 frames read for test case 70.
test case version: v1,05dec09
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
0(0:0)0  0(1:1)0  1(2:2)1  1(3:3)1
2(4:4)2  2(5:5)2  3(6:6)3  3(7:7)3
4(8:8)4  4(9:9)4  5(10:10)5  5(11:11)5
6(12:12)6  6(13:13)6  7(14:14)7  7(15:15)7

WARNING from make_dump_index: Times stamp rollover detected.
 Assume rollover in ordering.
ans = 91
NOTE: Accumulator rollover at centre lag of chain 7 fixed by adding 2^23.

Initial (dump= 1) timestamp (word 0): 4294967295
Missing timestamps (not increment by 1).

WARNINGS encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.

Using model_diff_chains: 1.10: 05-10-02
Using make_stat_tables: 1.30: 05-08-04
Generating tables of model lag and frequency domain
difference statistics for chains 7 through 16; using name: tc70
Using peak detection threshold of 3.00 std deviation
 over lags and 10.00 std. dev. over the frequency domain
 for a given chain.

***--------------------------------------------------------------------------***
octave:239> tc70_ACC_OV_scan

Bit ACC_OV set in ouput frame of cell 14, dump 82.
Bit ACC_OV set in ouput frame of cell 15, dump 82.

octave:240> tc70_make_dump_ovr_list

Number of dumps: 82

Using list_dump_ovr: 1.10: 05-07-25
Requested number of dumps of cells dumping at the highest rate: 128
Requested number of dumps of cells dumping at the highest rate/2: 64
Requested number of dumps of cells dumping at the highest rate/4: 32
Requested number of dumps of cells dumping at the highest rate/8: 16

octave:241> tc70_dump_overrun_tables
octave:242> tc70_chain_similarity_check


NOTICE: No differences exceeding the floating point precision
        limit of this machine were found between cells having similar
        OVR bit states and identical streams for any of the 82 dumps.

octave:261> tc69_70_chain_compare


NOTICE: No differences exceeding the floating point precision
        limit of this machine were found between identically
        configured cells of test cases 69 and 70 having similar
        OVR bit states.
```

Text 2.9.10: console log for scripts tc70_postsim, tc70_ACC_OV_scan, tc70_make_dump_ovr_list, tc70_dump_overrun_tables, tc70_chain_similarity_check, tc69_70_chain_compare.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

Check OK: Captured X,Y ID data (SID, SBID, BBID) are set
          to expected values.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

WARNING from check_register_capture: MCSR.AOV differs
               from template.
```

Text 2.9.11: test case 70; check register and capture log; WARNING noted.

timestamp | cell number	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***
20	x	x	1	1				
22	1	x	x	x	1	0	0	0
23	x	1						
24	x	x	1	1				
26	x	x	x	x	0	0		
27	1	x						
28	x	1	1	x				
30	x	x	x	1	0	0	0	0
31	1	x						
32	x	1	1	x				
34	x	x	x	1	0	0		
38	1	x	x	x	x	x	0	0
39	x	1						
40	x	x	1	1				
42	x	x	x	x	1	1		
43	1	x						
44	x	1	x	x				
46	x	x	1	1	0	0	0	0
47	1	x						
48	x	1	0	x				
50	x	x	x	1	0	0		
53	1	x						
54	x	1	x	x	x	x	0	0
56	x	x	1	1				
58	1	x	x	x	1	1		
59	x	1						
60	x	x	1	1				
62	1	x	x	x	0	0	0	0
63	x	1						
64	x	x	1	1				
66	x	x	x	x	0	0		
70	x	x	x	x	x	x	0	0
71	1	1						
72	x	x	1	x				
74	x	x	x	1	1	1		
76	1	1	x	x				
78	x	x	1	1	0	x	0	0
82	1	x	x	x	x	1		
83	x	1						
84	x	x	1	1				
86	1	x	x	x	1	0	0	0
87	x	1						
88	x	x	1	1				
90	x	x	x	x	0	0		
91	1	x						
92	x	1	1	x				

timestamp | cell number	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
94	x	x	x	1	0	0	0	0
95	1	1						
96	x	x	1	x				
98	x	x	x	1	0	0		
102	1	x	x	x	x	x	0	0
103	x	1						
104	x	x	1	1				
106	x	x	x	x	1	1		
107	1	x						
108	x	1	1	x				
110	x	x	x	1	0	0	0	0
111	1	x						
112	x	1	1	x				
114	x	x	x	1	0	0		
117	1	x						
118	x	1	1	x	x	x	0	0
120	x	x	x	1				
122	1	x	x	x	1	1		
123	x	1						
124	x	x	1	1				
126	1	x	x	x	0	0	0	0

Table 2.9.13: test case 70; table of dump overrun bit states.

stream	*dump*	*bias:!OVR*	*dump-bias:!OVR*	*bias:OVR*	*dump-bias:OVR*
0	1000	724	276	996	4
1	2000	1724	276	1996	4
2	4000	3724	276	3996	4
3	8000	7724	276	0	n-a
4	16000	15724	276	0	n-a
5	32000	31724	276	0	n-a
6	64000	63724	276	0	n-a
7	128000	127724	276	0	n-a

Table 2.9.14: test case 70; applied dump interval and bias.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
00011101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=1 OVR=1 PhEN=1 TVEN=0 CEN=1

10011001  CCCSCR_0_1:   CCC1: Y=sec X=pri   CCC0: Y=sec X=pri
01100110  CCCSCR_2_3:   CCC3: Y=pri X=sec   CCC2: Y=pri X=sec
10011001  CCCSCR_4_5:   CCC5: Y=sec X=pri   CCC4: Y=sec X=pri
01100110  CCCSCR_6_7:   CCC7: Y=pri X=sec   CCC6: Y=pri X=sec
10011001  CCCSCR_8_9:   CCC9: Y=sec X=pri   CCC8: Y=sec X=pri
01100110  CCCSCR_10_11: CCC11: Y=pri X=sec CCC10: Y=pri X=sec
10011001  CCCSCR_12_13: CCC13: Y=sec X=pri CCC12: Y=sec X=pri
01100110  CCCSCR_14_15: CCC15: Y=pri X=sec CCC14: Y=pri X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.9.12: content of file mcb_reg70.txt.

2.10 on-line detection and reporting of errors

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

2.10.1 organization

Many of the test cases that validate *on line error detection and reporting* are also represented in other sections of this document. In such cases, reference is made to the section containing the post simulation logs and additional material that is specific to *on line error detection* will be added to this section. Some test cases support validation with only contents of the MCB register file.

2.10.2 test case 03

Synopsis: the **XY-SE_CLK** bits within the **XYSTATU**S register that correspond to odd numbered streams are set (as expected). The **SyncER** bit of the **MCSR** is reset (as expected).

Notes: post simulation log and MCB register file contents are found in section 2.5.3.

2.10.3 test case 04

Synopsis: **XY-SDATA**, **XY-PHASE** and **XY-SE_CLK** bits of the **XYSTATU**S register corresponding to all data streams are set (as expected) with exceptions noted below. The **SyncER** bit of the **MCSR** is set as expected.

Notes:

(1) post simulation log and MCB register file contents are found in section 2.5.4.

(1) **XSTATUS** and **YSTATUS** registers do not report errors on **X|Y-DVALID** in any of test cases 04, 06, 11 as these test cases did not invoke the test bench facility to specifically set an error condition on these lines.

(2) The **SE_CLK** warning in the log of check_register_capture Text 2.5.6 is generated because a **X|Y-STATUS** register bit is reset for a single **SE_CLK** (stream 6); unexpected since **SE_CLK** was enabled for all streams and all other **SE_CLK** status bits were set.

2.10.4 test case 05

Synopsis: all **XY-SE_CLK** bits within the **XYSTATU**S register with exception of the bits corresponding to data stream 1 are set (as expected). The **SyncER** bit of the **MCSR** is reset (as expected).

Notes: post simulation log and MCB register file contents are found in section 2.5.5.

2.10.5 test case 06

Synopsis: **XY-SDATA**, **XY-PHASE** and **XY-SE_CLK** bits of the **XYSTATUS** register corresponding to all data streams are set (as expected). The **SyncER** bit of the **MCSR** is set (as expected).

Notes:

(1) post simulation log and MCB register file contents are found in section 2.5.6.

(1) A single advisory in the log of *check_register_capture* (Text 2.5.11) was generated as a consequence of the **X|YSyncerr** bits of an output frame differing between cells common to a chain. This behaviour is not specified within the RFS document and is therefore not considered a validation failure.

(2) **XSTATUS** and **YSTATUS** registers do not report errors on **X|Y-DVALID** in any of test cases 04, 06, 11 as these test cases did not invoke the test bench facility to specifically set an error condition on these lines.

2.10.6 test case 11

Synopsis: **XY-SDATA**, **XY-PHASE** and **XY-SE_CLK** bits of the **XYSTATU**S register corresponding to all data streams are set (as expected) with exceptions noted below. The **SyncER** bit of the **MCSR** is set as expected.

Notes:

(1) post simulation log and MCB register file contents are found in section 2.5.9.

(1) **XSTATUS** and **YSTATUS** registers do not report errors on **X|Y-DVALID** in any of test cases 04, 06, 11 as these test cases did not invoke the test bench facility to specifically set an error condition on these lines. The template applied to *check_register_capture* did not reflect this knowledge a priori, hence, the log (Text 2.5.22) contains a warning regarding unexpected **XYDVALID** states.

(2) The **SE_CLK** warning in the log of check_register_capture Text 2.5.22 is generated because an **X|Y-STATUS** register bit is reset for a single **SE_CLK** (stream 6); unexpected since **SE_CLK** was enabled for all streams and all other status bits were set.

2.10.7 test case 12

Synopsis: **X-DUMP_EN** bits of the **XSTATUS** register corresponding to the even numbered streams are set (as expected) as reflected by the console log, Text 2.5.24 and contents of file mcb_reg12.txt, Text 2.5.25. The **SyncER** bit of the **MCSR** is set as expected.

Notes: post simulation log and MCB register file contents are found in section 2.5.10.

2.10.8 test case 13

Synopsis: **X|Y-SE_CLK** bits of the **X|YSTATUS** registers corresponding to data streams 4,5,6,7 are set (as expected). The **SyncER** bit of the **MCSR** is reset (as expected). The expected chains dumped and all passed a bit exact comparison with chain 1.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc13_postsim.m : V2.00 run time: Tue Sep 27 14:53:27 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 13.
test case version: v3,05apr22
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (single dump) timestamp (word 0): 4294967295

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***

octave:56> tc13_list_dumped_chains
(0:1)  (2:3)  (4:5)  (6:7)  (8:9)  (10:11)  (12:13)  (14:15)

octave:57> tc13_chain_similarity_check

No differences exceeding the floating point precision limit of this
machine were found between chains for any of the 1 dumps; all chains generated
identical lag data.
```

Text 2.10.1 console log for tc13_postsim, tc13_list_dumped_chains, tc13_chain_similarity_check.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
00000101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
10000001  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=pri
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
10000001  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=pri
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
10000001  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=pri
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
10000001  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=pri

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11110000  XSTATUS: X-SE_CLK: 11110000
11110000  YSTATUS: Y-SE_CLK: 11110000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.2 content of file mcb_reg13.txt.

2.10.9 test case 17

Synopsis: all **X-SE_CLK** bits of the **XSTATUS** register except that corresponding to data stream 1 are set (as expected); **Y-SE_CLK** bits of the **YSTATUS** register corresponding to even numbered data streams are set (as expected).

Notes: post simulation log and MCB register file contents are found in section 2.7.4.

2.10.10 test case 18

Synopsis: all **Y-SE_CLK** bits of the **YSTATUS** register except that corresponding to data stream 1 are set (as expected); **X-SE_CLK** bits of the **XSTATUS** register corresponding to even numbered data streams are set (as expected).

Notes: post simulation log and MCB register file contents are found in section 2.7.5.

2.10.11 test case 26

Synopsis: the **X-TIMESTAMP** bit of the **XSTATUS** register is set (as expected) as seen in the MCB register file, Text 2.10.3. The **SyncER** bit of the **MCSR** is set as expected.

Notes: the **Y-TIMESTAMP** is reset even though a synchronization error is specified within the test case file; expected as a consequence of test bench operation which applies the timestamp of the line selected by dump control to both *X* and *Y* timestamp inputs.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

01000010  CCCSCR_0_1:   CCC1: Y=pri X=adj   CCC0: Y=adj X=sec
10000001  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=pri
01000001  CCCSCR_4_5:   CCC5: Y=pri X=adj   CCC4: Y=adj X=pri
10000010  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=sec
01000010  CCCSCR_8_9:   CCC9: Y=pri X=adj   CCC8: Y=adj X=sec
10000001  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=pri
01000001  CCCSCR_12_13: CCC13: Y=pri X=adj CCC12: Y=adj X=pri
10000010  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=sec

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
11110000  XSTATUS: X-SE_CLK: 11110000
11110000  YSTATUS: Y-SE_CLK: 11110000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000001  XSTATUS: X-TIMESTAMP=1 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.3 content of file mcb_reg26.txt.

2.10.12 test case 27

Synopsis: frames are generated for all 8 chains but only chains associated with even numbered data streams contain valid lag data. **Y-SE_CLK** bits of the **YSTATUS** register corresponding to odd numbered data streams are set (as expected) and , as a consequence, the **SyncER** bit of the **MCSR** is set as expected.

Notes:

(1) Chains containing bad (NaN) lag data are listed in Text 2.10.4.

(2) The ad hoc test following line *octave:96>* lists numbers of cells that produce frames containing bad data ('x' or 'X' in the output frame). This list is consistent with the list of chains containing bad lag data and consistent with the reset **SE_CLK** stream numbers above.

(3) Division by zero warnings in Text 2.10.4 and the warnings in the log generated by the function *check_register_capture* (Text 2.10.5) are a consequence of the bad data discussed above and are of no concern in the context of this test case.

```
***--------------------------------------------------------------------------***
tc27_postsim.m : V2.00 run time: Tue Sep 27 15:27:35 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 27.
test case version: v2,05apr23

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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

warning: in /rigel1/smegal/EVLAC/corrchip/postsim_lib/scale_lag_data.m near line   62, column 7:

>>> clags = (CCchain.clag - data_bias * (1 + 1i)) / dvcount;


warning: division by zero
warning: division by zero
warning: division by zero
warning: division by zero

Initial (single dump) timestamp (word 0): 222333444

WARNINGS encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***

octave:86> tc27_list_dumped_chains
(0:1)  (2:3)  (4:5)  (6:7)  (8:9)  (10:11)  (12:13)  (14:15)

octave:87> tc27_find_NaN
chain (0:1) contain lag data
chain (2:3) all NaN
chain (4:5) contain lag data
chain (6:7) all NaN
chain (8:9) contain lag data
chain (10:11) all NaN
chain (12:13) contain lag data
chain (14:15) all NaN

octave:96>
for ii=1:8;printf("%d  ",frame27(dump27(1).index(dontcare_frames(ii))).CCC);endfor;
11  14  15  2  3  6  7  10
```

Text 2.10.4 console log for scripts tc27_postsim, tc27_list_dumped_chains, tc27_find_NaN, ad hoc tests.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 2.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 2.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 2.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 4.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 4.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 4.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 6.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 6.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 6.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 8.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 8.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 8.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                             template of expected states.
```

Text 2.10.5 test case 27; check register and capture log; WARNINGS noted.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
10101010  YSTATUS: Y-SE_CLK: 10101010
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.6: content of file mcb_reg27.txt.

stream	*applied dump*	*bias: !OVR*	*dump-bias: !OVR*	*bias: OVR*	*dump-bias: OVR*
0	10000	9724	276	0	n-a
1	10000	9724	276	0	n-a
2	10000	9724	276	0	n-a
3	10000	9724	276	0	n-a
4	10000	9724	276	0	n-a
5	10000	9724	276	0	n-a
6	10000	9724	276	0	n-a
7	10000	9724	276	0	n-a

Table 2.10.1 test case 27; applied dump interval and bias.

2.10.13 test case 35

Synopsis: frames are generated for all 4 chains but only chain 1, associated with the stream 0x0 product, contain valid lag data. **XY-SE_CLK** bits of the **XYSTATUS** register corresponding to odd numbered data streams are set (as expected). The **SyncER** bit of the **MCSR** is set (as expected) as a consequence of the reset **SE_CLK** stream 1 participation in products that result in frames (that is to say, involving streams 0 and 1).

Notes:

(1) Chains containing bad (NaN) lag data are listed in Text 2.10.7.

(2) The ad hoc test following line *octave:121>* (in Text 2.10.7) lists numbers of cells that produce frames containing bad data ('x' or 'X' in the output frame). This list is consistent with the list of chains containing bad lag data and consistent with chains involving reset **SE_CLK** stream 1.

(3) Division by zero warnings in Text 2.10.7 and the warnings in the log generated by the function *check_register_capture* (Text 2.10.8) are a consequence of the bad data discussed above and are of no concern in the context of this test case.

```
***--------------------------------------------------------------------------***
tc35_postsim.m : V2.00 run time: Tue Sep 27 17:03:00 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 35.
test case version: v2,05apr23

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
0(0:3)0  0(4:7)1  1(8:11)0  1(12:15)1

warning: in /rigel1/smegal/EVLAC/corrchip/postsim_lib/scale_lag_data.m near line 62, column 7:

>>> clags = (CCchain.clag - data_bias * (1 + 1i)) / dvcount;


warning: division by zero
warning: division by zero
warning: division by zero

Initial (single dump) timestamp (word 0): 4294967295

WARNINGS encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***

octave:118> tc35_list_dumped_chains
(0:3)  (4:7)  (8:11)  (12:15)

octave:119> tc35_find_NaN
chain (0:3) contain lag data
chain (4:7) all NaN
chain (8:11) all NaN
chain (12:15) all NaN

octave:120> printf("%d \n",columns(dontcare_frames));
12

octave:121> for ii=1:12;printf("%d  ",frame35(dump35(1).index(dontcare_frames(ii))).CCC);endfor;
11  12  13  14  15  4  5  6  7  8  9  10
```

Text 2.10.7 console log for scripts tc35_postsim, tc35_list_dumped_chains, tc35_find_NaN, ad hoc tests.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
                frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 2.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 2.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 2.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      differs between cells of chain 3.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) differs between cells of chain 3.

WARNING from check_register_capture: Captured X-Station baseband
      ID (BBID-X) differs between cells of chain 3.

WARNING from check_register_capture: Captured X-Station ID (SID-X)
      differs between cells of chain 4.

WARNING from check_register_capture: Captured Y-Station ID (SID-Y)
      differs between cells of chain 4.

WARNING from check_register_capture: Captured X-Station subband
      ID (SBID-X) differs between cells of chain 4.

WARNING from check_register_capture: Captured Y-Station subband
     ID (SBID-Y) differs between cells of chain 4.

WARNING from check_register_capture: Captured X-Station baseband
      ID (BBID-X) differs between cells of chain 4.

WARNING from check_register_capture: Captured Y-Station baseband
      ID (BBID-Y) differs between cells of chain 4.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
               template of expected states.
```

Text 2.10.8: tc35; check register and capture log; WARNINGS noted.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
01000101  MCSR: X/Y_DS=0 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
10101010  XSTATUS: X-SE_CLK: 10101010
10101010  YSTATUS: Y-SE_CLK: 10101010
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.9 content of file mcb_reg35.txt.

stream	*applied dump*	*bias: !OVR*	*dump-bias: !OVR*	*bias: OVR*	*dump-bias: OVR*
0	10000	9724	276	0	n-a
1	10000	9724	276	0	n-a

Table 2.10.2 test case 35; applied dump interval and bias.

2.10.14 test case 36

Synopsis: the **Y-TIMESTAMP** bit of the **YSTATUS** register is set (as expected). The **SyncER** bit of the **MCSR** is set as expected. **Y-DUMP_EN** bits of the **YSTATUS** register corresponding to the odd numbered streams are set (as expected) as reflected by the console log, Text 2.10.10 and contents of file mcb_reg36.txt, Text 2.10.11.

Notes: n-a

```
***-------------------------------------------------------------------------***
tc36_postsim.m : V2.00 run time: Mon Sep 26 17:31:40 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 36.
test case version: v3,05apr36
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (single dump) timestamp (word 0): 4294967295

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.
***-------------------------------------------------------------------------***
```

Text 2.10.10 console log for script tc36_postsim.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
01000101  MCSR: X/Y_DS=0 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000001  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=sec
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
10101010  YSTATUS: Y-DUMP_EN: 10101010
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000001  YSTATUS: Y-TIMESTAMP=1 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.11 content of file mcb_reg36.txt.

2.10.15 test case 37

Synopsis: **XY-DUMP_SYNC** bits of the **XYSTATUS** registers are set (as expected).

Notes: n-a

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000100  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=1
00000100  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=1

11111111  DESSR_0_7: 11111111
11111111  DESSR_8_15: 11111111
```

Text 2.10.12 content of file mcb_reg37.txt.

2.10.16 test case 46

Synopsis: **X-DUMP_EN** bits of the **XSTATUS** register corresponding to the odd numbered streams are set (as expected) as reflected by the console log, Text 2.15.3 and contents of file mcb_reg46.txt, Text 2.15.4. The **SyncER** bit of the **MCSR** is set as expected.

Notes: post simulation log and mcb_reg46 file contents are found in section 2.15.4.

2.10.17 test case 47

Synopsis: **Y-DUMP_EN** bits of the **YSTATUS** register corresponding to the even numbered streams are set (as expected) as reflected by the console log, Text 2.15.5 and contents of file mcb_reg47.txt, Text 2.15.6.

Notes: post simulation log and mcb_reg47 file contents are found in section 2.15.5.

2.10.18 test case 50

Synopsis: **XY-SDATA** and **XY-PHASE** bit states of the **XYSTATUS** registers are expected; checked by the script *tc50_check_status_bits* and inspection of file *mcb_reg50.txt* (Text 2.10.17). Captured data differs from the template and test case source file as a consequence of reset data valid lines. The script *tc50_check_captured_data* accounts for the missing data lines and validates the captured data.

Notes: warnings in the *check_register_capture* log, Text 2.10.14 to Text 2.10.16 result from generally reset valid data lines.

```
***--------------------------------------------------------------------------***
tc50_postsim.m : V2.00 run time: Tue Sep 27 10:09:56 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 50.
test case version: v1,05apr24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
1(0:0)1  1(1:1)0  0(2:2)1  0(3:3)0
3(4:4)3  3(5:5)2  2(6:6)3  2(7:7)2
5(8:8)5  5(9:9)4  4(10:10)5  4(11:11)4
7(12:12)7  7(13:13)6  6(14:14)7  6(15:15)6


Initial (single dump) timestamp (word 0): 222333444

WARNINGS encountered during register status and data capture checks;
                       details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***
octave:29> tc50_check_captured_data

Check OK: Expected data are captured on streams with reset data lines.
octave:30> tc50_check_status_bits

 Check OK: X|Y-DATA and X|Y-PHASE STATUS register bits are set
as expected.
```

Text 2.10.13 console log for scripts tc50_postsim, tc50_check_captured_data, tc50_check_status_bits.

stream	***dump***	***bias: !OVR***	***dump-bias: !OVR***	***bias: OVR***	***dump-bias: OVR***
0	5000	4724	276	0	n-a
1	5000	4724	276	0	n-a
2	5000	4724	276	0	n-a
3	5000	4724	276	0	n-a
4	5000	4724	276	0	n-a
5	5000	4724	276	0	n-a
6	5000	4724	276	0	n-a
7	5000	4724	276	0	n-a

Table 2.10.3 test case 50; applied dump interval and bias.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:1 and Y:1.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:1 and Y:1.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:1 and Y:0.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:1 and Y:0.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:0 and Y:1.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:0 and Y:0.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:3 and Y:3.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:3 and Y:3.
WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:3 and Y:2.
```

Text 2.10.14 test case 50; check register and captured data log; WARNINGS noted; part (1).

```
WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:3 and Y:2.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:2 and Y:3.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:2 and Y:3.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:2 and Y:2.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:2 and Y:2.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:5 and Y:5.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:5 and Y:5.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:5 and Y:4.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:5 and Y:4.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:4 and Y:5.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:4 and Y:5.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:4 and Y:4.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:4 and Y:4.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:7 and Y:7.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:7 and Y:7.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:7 and Y:6.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:6 and Y:7.
```

Text 2.10.15 test case50; check register and captured data log; WARNINGS noted; part (2).

```
WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:6 and Y:7.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:6 and Y:6.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                            template of expected states.
```

Text 2.10.16: test case50; check register and captured data log; WARNINGS noted; part (3).

```
# ------------------------------------------------------------------------------
# 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:
#
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10100110  CCCSCR_0_1:   CCC1: Y=sec X=sec   CCC0: Y=pri X=sec
10010101  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=pri X=pri
10100110  CCCSCR_4_5:   CCC5: Y=sec X=sec   CCC4: Y=pri X=sec
10010101  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=pri X=pri
10100110  CCCSCR_8_9:   CCC9: Y=sec X=sec   CCC8: Y=pri X=sec
10010101  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=pri X=pri
10100110  CCCSCR_12_13: CCC13: Y=sec X=sec CCC12: Y=pri X=sec
10010101  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=pri X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

01010101  XSTATUS: X-SDATA1: 0101, X-SDATA0: 0101
10101010  YSTATUS: Y-SDATA1: 1010, Y-SDATA0: 1010
01010101  XSTATUS: X-SDATA3: 0101, X-SDATA2: 0101
10101010  YSTATUS: Y-SDATA3: 1010, Y-SDATA2: 1010
01010101  XSTATUS: X-SDATA5: 0101, X-SDATA4: 0101
10101010  YSTATUS: Y-SDATA5: 1010, Y-SDATA4: 1010
01010101  XSTATUS: X-SDATA7: 0101, X-SDATA6: 0101
10101010  YSTATUS: Y-SDATA7: 1010, Y-SDATA6: 1010
01010101  XSTATUS: X-PHASE1: 0101, X-PHASE0: 0101
10101010  YSTATUS: Y-PHASE1: 1010, Y-PHASE0: 1010
01010101  XSTATUS: X-PHASE3: 0101, X-PHASE2: 0101
10101010  YSTATUS: Y-PHASE3: 1010, Y-PHASE2: 1010
01010101  XSTATUS: X-PHASE5: 0101, X-PHASE4: 0101
10101010  YSTATUS: Y-PHASE5: 1010, Y-PHASE4: 1010
01010101  XSTATUS: X-PHASE7: 0101, X-PHASE6: 0101
10101010  YSTATUS: Y-PHASE7: 1010, Y-PHASE6: 1010
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.17 content of file mcb_reg50.txt.

2.10.19 test case 51

Synopsis: **XY-SDATA** and **XY-PHASE** bit states of the **XYSTATUS** registers are expected; checked by the script *tc51_check_status_bits* and inspection of file *mcb_reg51.txt* (Text 2.10.22). Captured data differs from the template and test case source file as a consequence of generally reset data valid lines. The script *tc51_check_captured_data* accounts for the missing data lines and validates the captured data.

Notes: warnings in the *check_register_capture* log, Text 2.10.19 to Text 2.10.21 result from generally reset data valid lines.

```
***--------------------------------------------------------------------------***
tc51_postsim.m : V2.00 run time: Tue Sep 27 10:14:57 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 51.
test case version: v1,05apr24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
1(0:0)1  1(1:1)0  0(2:2)1  0(3:3)0
3(4:4)3  3(5:5)2  2(6:6)3  2(7:7)2
5(8:8)5  5(9:9)4  4(10:10)5  4(11:11)4
7(12:12)7  7(13:13)6  6(14:14)7  6(15:15)6


Initial (single dump) timestamp (word 0): 222333444

WARNINGS encountered during register status and data capture checks;
                       details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***
octave:34> tc51_check_captured_data

Check OK: Expected data are captured on streams with reset data lines.
octave:35> tc51_check_status_bits

 Check OK: X|Y-DATA and X|Y-PHASE STATUS register bits are set
as expected.
```

Text 2.10.18 console log for scripts tc51_postsim, tc51_check_captured_data, tc51_check_status_bits.

stream	*applied dump*	*bias: !OVR*	*dump-bias: !OVR*	*bias: OVR*	*dump-bias: OVR*
0	5000	4724	276	0	n-a
1	5000	4724	276	0	n-a
2	5000	4724	276	0	n-a
3	5000	4724	276	0	n-a
4	5000	4724	276	0	n-a
5	5000	4724	276	0	n-a
6	5000	4724	276	0	n-a
7	5000	4724	276	0	n-a

Table 2.10.4 test case 51; applied dump interval and bias.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:1 and Y:1.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:1 and Y:1.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:1 and Y:0.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:0 and Y:1.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:0 and Y:1.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:0 and Y:0.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:3 and Y:3.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:3 and Y:3.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:3 and Y:2.
```

Text 2.10.19 test case 51; check register and captured data log; WARNINGS noted; part (1).

```
WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:3 and Y:2.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:2 and Y:3.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:2 and Y:3.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:2 and Y:2.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:2 and Y:2.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:5 and Y:5.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:5 and Y:4.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:5 and Y:4.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:4 and Y:5.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:4 and Y:5.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:4 and Y:4.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:4 and Y:4.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:7 and Y:7.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:7 and Y:7.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:7 and Y:6.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:6 and Y:7.

WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:6 and Y:7.

WARNING from check_register_capture: Captured Station subband IDs
       (SBID) differ from expected values.
       Check chains associated with data streams X:6 and Y:6.
```

Text 2.10.20 test case 51; check register and captured data log; WARNINGS noted; part (2).

```
WARNING from check_register_capture: Captured Station baseband IDs
       (BBID) differ from expected values.
       Check chains associated with data streams X:6 and Y:6.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                            template of expected states.
```

Text 2.10.21: test case 51; check register and captured data log; WARNINGS noted; part (3).

```
# ------------------------------------------------------------------------------
# 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:
#
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10100110  CCCSCR_0_1:   CCC1: Y=sec X=sec   CCC0: Y=pri X=sec
10010101  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=pri X=pri
10100110  CCCSCR_4_5:   CCC5: Y=sec X=sec   CCC4: Y=pri X=sec
10010101  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=pri X=pri
10100110  CCCSCR_8_9:   CCC9: Y=sec X=sec   CCC8: Y=pri X=sec
10010101  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=pri X=pri
10100110  CCCSCR_12_13: CCC13: Y=sec X=sec CCC12: Y=pri X=sec
10010101  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=pri X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

10101010  XSTATUS: X-SDATA1: 1010, X-SDATA0: 1010
01010101  YSTATUS: Y-SDATA1: 0101, Y-SDATA0: 0101
10101010  XSTATUS: X-SDATA3: 1010, X-SDATA2: 1010
01010101  YSTATUS: Y-SDATA3: 0101, Y-SDATA2: 0101
10101010  XSTATUS: X-SDATA5: 1010, X-SDATA4: 1010
01010101  YSTATUS: Y-SDATA5: 0101, Y-SDATA4: 0101
10101010  XSTATUS: X-SDATA7: 1010, X-SDATA6: 1010
01010101  YSTATUS: Y-SDATA7: 0101, Y-SDATA6: 0101
10101010  XSTATUS: X-PHASE1: 1010, X-PHASE0: 1010
01010101  YSTATUS: Y-PHASE1: 0101, Y-PHASE0: 0101
10101010  XSTATUS: X-PHASE3: 1010, X-PHASE2: 1010
01010101  YSTATUS: Y-PHASE3: 0101, Y-PHASE2: 0101
10101010  XSTATUS: X-PHASE5: 1010, X-PHASE4: 1010
01010101  YSTATUS: Y-PHASE5: 0101, Y-PHASE4: 0101
10101010  XSTATUS: X-PHASE7: 1010, X-PHASE6: 1010
01010101  YSTATUS: Y-PHASE7: 0101, Y-PHASE6: 0101
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.22 content of file mcb_reg51.txt.

2.10.20 test case 52

Synopsis: the station channel ID synchronization bits, **XY-SCHID_SYNC** of the **XYSTATUS** registers are both set (as expected). The **SyncER** bit of the **MCSR** is set (as expected).

Notes: n-a

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000110  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=1 X-DUMP_SYNCs=1
00000110  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=1 Y-DUMP_SYNCs=1

11111111  DESSR_0_7: 11111111
11111111  DESSR_8_15: 11111111
```

Text 2.10.23 content of file mcb_reg52.txt.

2.10.21 test case 53

Synopsis: the station channel ID synchronization bits, **XY-SCHID_SYNC** of the **XYSTATUS** registers are both set (as expected). The **SyncER** bit of the **MCSR** is set (as expected).

Notes: n-a

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000110  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=1 X-DUMP_SYNCs=1
00000110  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=1 Y-DUMP_SYNCs=1

11111111  DESSR_0_7: 11111111
11111111  DESSR_8_15: 11111111
```

Text 2.10.24 content of file mcb_reg53.txt.

2.10.22 test case 65

Synopsis: The **SyncER** bit of the **MCSR** and **XYSTATUS** register bits are set as expected.

Notes: n-a

```
# ------------------------------------------------------------------------------
# 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:
#
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10100110  CCCSCR_0_1:   CCC1: Y=sec X=sec   CCC0: Y=pri X=sec
10010101  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=pri X=pri
10100110  CCCSCR_4_5:   CCC5: Y=sec X=sec   CCC4: Y=pri X=sec
10010101  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=pri X=pri
10100110  CCCSCR_8_9:   CCC9: Y=sec X=sec   CCC8: Y=pri X=sec
10010101  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=pri X=pri
10100110  CCCSCR_12_13: CCC13: Y=sec X=sec CCC12: Y=pri X=sec
10010101  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=pri X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00001111  XSTATUS: X-SDATA1: 0000, X-SDATA0: 1111
11110000  YSTATUS: Y-SDATA1: 1111, Y-SDATA0: 0000
00001111  XSTATUS: X-SDATA3: 0000, X-SDATA2: 1111
11110000  YSTATUS: Y-SDATA3: 1111, Y-SDATA2: 0000
00001111  XSTATUS: X-SDATA5: 0000, X-SDATA4: 1111
11110000  YSTATUS: Y-SDATA5: 1111, Y-SDATA4: 0000
00001111  XSTATUS: X-SDATA7: 0000, X-SDATA6: 1111
11110000  YSTATUS: Y-SDATA7: 1111, Y-SDATA6: 0000
00001111  XSTATUS: X-PHASE1: 0000, X-PHASE0: 1111
11110000  YSTATUS: Y-PHASE1: 1111, Y-PHASE0: 0000
00001111  XSTATUS: X-PHASE3: 0000, X-PHASE2: 1111
11110000  YSTATUS: Y-PHASE3: 1111, Y-PHASE2: 0000
00001111  XSTATUS: X-PHASE5: 0000, X-PHASE4: 1111
11110000  YSTATUS: Y-PHASE5: 1111, Y-PHASE4: 0000
00001111  XSTATUS: X-PHASE7: 0000, X-PHASE6: 1111
11110000  YSTATUS: Y-PHASE7: 1111, Y-PHASE6: 0000
01010101  XSTATUS: X-DVALID: 01010101
10101010  YSTATUS: Y-DVALID: 10101010
01010101  XSTATUS: X-SE_CLK: 01010101
10101010  YSTATUS: Y-SE_CLK: 10101010
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.25: content of file mcb_reg65.txt.

2.10.23 test case 66

Synopsis: The **SyncER** bit of the **MCSR** and **XYSTATUS** register bits are set as expected.

Notes: n-a

```
# ------------------------------------------------------------------------------
# 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:
#
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10100110  CCCSCR_0_1:   CCC1: Y=sec X=sec   CCC0: Y=pri X=sec
10010101  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=pri X=pri
10100110  CCCSCR_4_5:   CCC5: Y=sec X=sec   CCC4: Y=pri X=sec
10010101  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=pri X=pri
10100110  CCCSCR_8_9:   CCC9: Y=sec X=sec   CCC8: Y=pri X=sec
10010101  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=pri X=pri
10100110  CCCSCR_12_13: CCC13: Y=sec X=sec CCC12: Y=pri X=sec
10010101  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=pri X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

11110000  XSTATUS: X-SDATA1: 1111, X-SDATA0: 0000
00001111  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 1111
11110000  XSTATUS: X-SDATA3: 1111, X-SDATA2: 0000
00001111  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 1111
11110000  XSTATUS: X-SDATA5: 1111, X-SDATA4: 0000
00001111  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 1111
11110000  XSTATUS: X-SDATA7: 1111, X-SDATA6: 0000
00001111  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 1111
11110000  XSTATUS: X-PHASE1: 1111, X-PHASE0: 0000
00001111  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 1111
11110000  XSTATUS: X-PHASE3: 1111, X-PHASE2: 0000
00001111  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 1111
11110000  XSTATUS: X-PHASE5: 1111, X-PHASE4: 0000
00001111  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 1111
11110000  XSTATUS: X-PHASE7: 1111, X-PHASE6: 0000
00001111  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 1111
10101010  XSTATUS: X-DVALID: 10101010
01010101  YSTATUS: Y-DVALID: 01010101
10101010  XSTATUS: X-SE_CLK: 10101010
01010101  YSTATUS: Y-SE_CLK: 01010101
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.10.26: content of file mcb_reg66.txt.

2.11 X-Y dump enable offset detection

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

2.11.1 organization

Many of the test cases that validate *X-Y dump enable offset detection* are also represented in other sections of this document and in such cases, reference is made to the section containing the post simulation logs . Additional material that is specific to *X-Y dump enable offset detection* will be added to this section. Some test cases support validation with only contents of the MCB register file.

2.11.2 test case 03

Synopsis: all **DESSR** register bits are set (as expected) and the **DUMP_SYNC** bit in *both* the **XSTATUS** and **YSTATUS** registers is set.

Notes: post simulation log and MCB register file contents are found in section 2.5.3.

2.11.3 test case 05

Synopsis: all **DESSR** register bits are set (as expected) and the **DUMP_SYNC** bit in *both* the **XSTATUS** and **YSTATUS** registers is set.

Notes: post simulation log and MCB register file contents are found in section 2.5.5.

2.11.4 test case 11

Synopsis: all **DESSR** register bits are set (as expected) and the **DUMP_SYNC** bit in *both* the **XSTATUS** and **YSTATUS** registers is set.

Notes: post simulation log and MCB register file contents are found in section 2.5.9.

2.11.5 test case 37

Synopsis: all bits of register **DESSR** are set (as expected) and the **DUMP_SYNC** bit in *both* the **XSTATUS** and **YSTATUS** registers is set (as expected).

Notes: MCB register file contents are found in section 2.10.15.

2.11.6 test case 48

Synopsis: the state of the register **DESSR** bits are as expected; an alternating set-reset bit pattern, the complement of which occurs in test case 49, section 2.11.7. The **XY-DUMP_SYNC** bit of the **XYSTATUS** register is reset (as expected).

Notes:

(1) The dump interval is too short for frame building and readout, hence, frames associated with cells dumping at the higher rate are not represented in the second dump. Since only two dumps occur, the dump overrun bit (**OVR**) remains reset.

(2) Warnings in the check register and capture log, Text 2.11.2 are a consequence of the unusual stimulation; short dump interval as described in the above point (1) and different dump interval between X and Y. The captured data tabulated in Table 2.11.1 is consistent with expectations and point (1) above, despite entries in Text 2.11.2, hence, these warnings are of no concern. A detailed investigation into the mechanism that produced these warnings has not been undertaken.

```
***--------------------------------------------------------------------------***
tc48_postsim.m : V2.00 run time: Wed Sep 28 17:21:34 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 48.
test case version: v3,05aug07
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Input configuration from MCSR regsister settings:
0(0:0)0 0(1:1)1 1(2:2)1 1(3:3)0 0(4:4)0 0(5:5)1 1(6:6)1 1(7:7)0 0(8:8)0
0(9:9)1 1(10:10)1 1(11:11)0 0(12:12)0 0(13:13)1 1(14:14)1 1(15:15)0


WARNINGS encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***

octave:113> tc48_captured_table
octave:114> tc48_data_ID_capture
```

Text 2.11.1 console log for script tc48_postsim, tc48_captured_table, tc48_data_ID_capture.

dump 1	***cell number***	**(0)**	**(1)**	**(4)**	**(5)**	**(8)**	**(9)**	**(12)**	**(13)**
dump 1	***Phase bin***	55666	55666	55666	55666	55666	55666	55666	55666
dump 1	***cmmd***	010	010	010	010	010	010	010	010
dump 1	***SBID_X***	0	0	0	0	0	0	0	0
dump 1	***SBID_Y***	0	1	0	1	0	1	0	1
dump 1	***BBID_X***	0	0	0	0	0	0	0	0
dump 1	***BBID_Y***	0	1	0	1	0	1	0	1
dump 2	***cell number***	**(2)**	**(3)**	**(6)**	**(7)**	**(10)**	**(11)**	**(14)**	**(15)**
dump 2	***Phase bin***	55666	55666	55666	55666	55666	55666	55666	55666
dump 2	***cmmd***	010	010	010	010	010	010	010	010
dump 2	***SBID_X***	1	1	1	1	1	1	1	1
dump 2	***SBID_Y***	1	0	1	0	1	0	1	0
dump 2	***BBID_X***	1	1	1	1	1	1	1	1
dump 2	***BBID_Y***	1	0	1	0	1	0	1	0

Table 2.11.1 test case 48; captured data generated by script tc48_captured_table.

stream	***dump***	***bias: !OVR***	***dump-bias: !OVR***	***bias: OVR***	***dump-bias: OVR***
0	500	224	276	0	n-a
1	1000	724	276	0	n-a

Table 2.11.2 test case 48; applied dump and bias.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

WARNING from check_register_capture: Captured Y-Station ID
       (SID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 0.

WARNING from check_register_capture: Captured Y-Station ID
       (SID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

WARNING from check_register_capture: Captured Y-Station subband ID
       (SBID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

WARNING from check_register_capture: Captured Y-Station baseband ID
       (BBID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

WARNING from check_register_capture: phase bin
       differs between chains with common input data stream.
       Check chains associated with Y data stream 0.

WARNING from check_register_capture: dump command
       differs between chains with common input data stream.
       Check chains associated with Y data stream 0.

WARNING from check_register_capture: phase bin
       differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

WARNING from check_register_capture: dump command
       differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

Check OK: MCSR and status registers agree with the
                             template of expected states.
```

Text 2.11.2 test case 48; check register and capture log; WARNINGS noted.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

01011001  CCCSCR_0_1:   CCC1: Y=pri X=pri   CCC0: Y=sec X=pri
10100110  CCCSCR_2_3:   CCC3: Y=sec X=sec   CCC2: Y=pri X=sec
01011001  CCCSCR_4_5:   CCC5: Y=pri X=pri   CCC4: Y=sec X=pri
10100110  CCCSCR_6_7:   CCC7: Y=sec X=sec   CCC6: Y=pri X=sec
01011001  CCCSCR_8_9:   CCC9: Y=pri X=pri   CCC8: Y=sec X=pri
10100110  CCCSCR_10_11: CCC11: Y=sec X=sec CCC10: Y=pri X=sec
01011001  CCCSCR_12_13: CCC13: Y=pri X=pri CCC12: Y=sec X=pri
10100110  CCCSCR_14_15: CCC15: Y=sec X=sec CCC14: Y=pri X=sec

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

10101010  DESSR_0_7: 10101010
10101010  DESSR_8_15: 10101010
```

Text 2.11.3 content of file mcb_reg48.txt.

2.11.7 test case 49

Synopsis: the state of the register **DESSR** bits are as expected; an alternating set-reset bit pattern, the complement of which occurs in test case 48, section 2.11.6. The **XY-DUMP_SYNC** bit of the **XYSTATUS** register is reset (as expected).

Notes:

(3) The dump interval is too short for frame building and readout, hence, frames associated with cells dumping at the higher rate are not represented in the second dump. Since only two dumps occur, the dump overrun bit (**OVR**) remains reset.

(4) Warnings in the check register and capture log, Text 2.11.5 are a consequence of the unusual stimulation; short dump interval as described in the above point (1) and different dump interval between X and Y. The captured data tabulated in Table 2.11.3 is consistent with expectations and point (1) above, despite entries in Text 2.11.5, hence, these warnings are of no concern. A detailed investigation into the mechanism that produced these warnings has not been undertaken.

```
***-------------------------------------------------------------------------***
tc49_postsim.m : V2.00 run time: Wed Sep 28 18:11:13 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 49.
test case version: v3,05aug07
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Input configuration from MCSR regsister settings:
1(0:0)0 1(1:1)1 0(2:2)1 0(3:3)0 1(4:4)0 1(5:5)1 0(6:6)1 0(7:7)0 1(8:8)0
1(9:9)1 0(10:10)1 0(11:11)0 1(12:12)0 1(13:13)1 0(14:14)1 0(15:15)0


WARNINGS encountered during register status and data capture checks;
                        details in log file.

Check OK: DESSR bits are set as expected.
***-------------------------------------------------------------------------***
octave:118> tc49_captured_table
octave:119> tc49_data_ID_capture
```

Text 2.11.4 console log of tc49_postsim, tc49_captured_table, tc49_data_ID_capture.

dump 1	***cell number***	**(2)**	**(3)**	**(6)**	**(7)**	**(10)**	**(11)**	**(14)**	**(15)**
dump 1	***Phase bin***	55666	55666	55666	55666	55666	55666	55666	55666
dump 1	***cmmd***	010	010	010	010	010	010	010	010
dump 1	***SBID_X***	0	0	0	0	0	0	0	0
dump 1	***SBID_Y***	1	0	1	0	1	0	1	0
dump 1	***BBID_X***	1	1	1	1	1	1	1	1
dump 1	***BBID_Y***	2	1	2	1	2	1	2	1
dump 2	***cell number***	**(0)**	**(1)**	**(4)**	**(5)**	**(8)**	**(9)**	**(12)**	**(13)**
dump 2	***Phase bin***	55666	55666	55666	55666	55666	55666	55666	55666
dump 2	***cmmd***	010	010	010	010	010	010	010	010
dump 2	***SBID_X***	1	1	1	1	1	1	1	1
dump 2	***SBID_Y***	0	1	0	1	0	1	0	1
dump 2	***BBID_X***	2	2	2	2	2	2	2	2
dump 2	***BBID_Y***	1	2	1	2	1	2	1	2

Table 2.11.3 test case 49; captured data generated by script tc49_captured_table.

stream	***applied dump***	***bias: !OVR***	***dump-bias: !OVR***	***bias: OVR***	***dump-bias: OVR***
0	500	224	276	0	n-a
1	1000	724	276	0	n-a

Table 2.11.4 test case 49; applied dump and bias.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

WARNING from check_register_capture: Captured Y-Station ID
       (SID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 0.

WARNING from check_register_capture: Captured Y-Station ID
       (SID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

WARNING from check_register_capture: Captured Y-Station subband ID
       (SBID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

WARNING from check_register_capture: Captured Y-Station baseband ID
       (BBID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 0.

WARNING from check_register_capture: Captured Y-Station baseband ID
       (BBID-Y) differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

WARNING from check_register_capture: phase bin
       differs between chains with common input data stream.
       Check chains associated with Y data stream 0.

WARNING from check_register_capture: dump command
       differs between chains with common input data stream.
       Check chains associated with Y data stream 0.

WARNING from check_register_capture: phase bin
       differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

WARNING from check_register_capture: dump command
       differs between chains with common input data stream.
       Check chains associated with Y data stream 1.

Check OK: MCSR and status registers agree with the
                            template of expected states.
```

Text 2.11.5 test case 49; check register and capture log; WARNINGS noted.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

01101010  CCCSCR_0_1:   CCC1: Y=pri X=sec   CCC0: Y=sec X=sec
10010101  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=pri X=pri
01101010  CCCSCR_4_5:   CCC5: Y=pri X=sec   CCC4: Y=sec X=sec
10010101  CCCSCR_6_7:   CCC7: Y=sec X=pri   CCC6: Y=pri X=pri
01101010  CCCSCR_8_9:   CCC9: Y=pri X=sec   CCC8: Y=sec X=sec
10010101  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=pri X=pri
01101010  CCCSCR_12_13: CCC13: Y=pri X=sec CCC12: Y=sec X=sec
10010101  CCCSCR_14_15: CCC15: Y=sec X=pri CCC14: Y=pri X=pri

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

01010101  DESSR_0_7: 01010101
01010101  DESSR_8_15: 01010101
```

Text 2.11.6 content of file mcb_reg49.txt.

2.12 off-line detection and reporting of synchronization errors

Test cases: 38, 39, 40

2.12.1 organization

Tests of this section are validated by inspection of the **MSCR** register file contents of Text 2.12.1 to Text 2.12.3.

2.12.2 test case 38

Summary: all bits (excluding **XY-SCHID_SYNC** and **XY-DUMP_SYNC)**[4] of the **XYSTATUS** register are reset as expected; the **SyncER** and **TvER** bits of the **MCSR** are reset as expected.

Notes: n-a

4 this statement is not meant to imply that **XY-SCHID_SYNC** and **XY-DUMP_SYNC** are set but that they are excluded from the group of bits with explicitly expected states.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
00000111  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=1 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
11111111  DESSR_8_15: 11111111
```

Text 2.12.1 content of the the file mcb_reg38.txt.

2.12.3 test case 39

Summary: all bits (excluding **XY-SCHID_SYNC** and **XY-DUMP_SYNC)**[5] of register **XYSTATUS** are set as expected; the **SyncER** bit of the **MCSR** is reset, as expected; the **TvER** bit of the **MCSR** is set, as expected.

Notes: n-a

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
00100111  MCSR: X/Y_DS=0 SyncER=0 TvER=1 AOV=0 OVR=0 PhEN=1 TVEN=1 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

11111111  XSTATUS: X-SDATA1: 1111, X-SDATA0: 1111
11111111  YSTATUS: Y-SDATA1: 1111, Y-SDATA0: 1111
11111111  XSTATUS: X-SDATA3: 1111, X-SDATA2: 1111
11111111  YSTATUS: Y-SDATA3: 1111, Y-SDATA2: 1111
11111111  XSTATUS: X-SDATA5: 1111, X-SDATA4: 1111
11111111  YSTATUS: Y-SDATA5: 1111, Y-SDATA4: 1111
11111111  XSTATUS: X-SDATA7: 1111, X-SDATA6: 1111
11111111  YSTATUS: Y-SDATA7: 1111, Y-SDATA6: 1111
11111111  XSTATUS: X-PHASE1: 1111, X-PHASE0: 1111
11111111  YSTATUS: Y-PHASE1: 1111, Y-PHASE0: 1111
11111111  XSTATUS: X-PHASE3: 1111, X-PHASE2: 1111
11111111  YSTATUS: Y-PHASE3: 1111, Y-PHASE2: 1111
11111111  XSTATUS: X-PHASE5: 1111, X-PHASE4: 1111
11111111  YSTATUS: Y-PHASE5: 1111, Y-PHASE4: 1111
11111111  XSTATUS: X-PHASE7: 1111, X-PHASE6: 1111
11111111  YSTATUS: Y-PHASE7: 1111, Y-PHASE6: 1111
11111111  XSTATUS: X-DVALID: 11111111
11111111  YSTATUS: Y-DVALID: 11111111
11111111  XSTATUS: X-SE_CLK: 11111111
11111111  YSTATUS: Y-SE_CLK: 11111111
11111111  XSTATUS: X-DUMP_EN: 11111111
11111111  YSTATUS: Y-DUMP_EN: 11111111
00000001  XSTATUS: X-TIMESTAMP=1 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000001  YSTATUS: Y-TIMESTAMP=1 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.12.2 content of file mcb_reg39.txt.

5 **XY-SCHID_SYNC** and **XY-DUMP_SYNC** are excluded from the group of bits with explicitly expected states.

2.12.4 test case 40

Summary: many but not all bits (excluding **XY-SCHID_SYNC** and **XY-DUMP_SYNC**,)[6] of register **XYSTATUS** are set; the **SyncER** bit of the **MCSR** is reset, as expected; the **TvER** bit of the **MCSR** is set, as expected.

Notes: a reduced (over test case 39) applied error rate resulted in some bits of the **XYSTATUS** register remaining reset. This result is reasonable to expect although there was no prediction of specific bit states.

```
# ------------------------------------------------------------------------------
# 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
#
# ------------------------------------------------------------------------------
00100111  MCSR: X/Y_DS=0 SyncER=0 TvER=1 AOV=0 OVR=0 PhEN=1 TVEN=1 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
01000000  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=adj
00000010  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=sec
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000010  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=sec
01000000  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

11111010  XSTATUS: X-SDATA1: 1111, X-SDATA0: 1010
11101111  YSTATUS: Y-SDATA1: 1110, Y-SDATA0: 1111
11111111  XSTATUS: X-SDATA3: 1111, X-SDATA2: 1111
11111111  YSTATUS: Y-SDATA3: 1111, Y-SDATA2: 1111
11111111  XSTATUS: X-SDATA5: 1111, X-SDATA4: 1111
11111111  YSTATUS: Y-SDATA5: 1111, Y-SDATA4: 1111
11111111  XSTATUS: X-SDATA7: 1111, X-SDATA6: 1111
11111111  YSTATUS: Y-SDATA7: 1111, Y-SDATA6: 1111
11111111  XSTATUS: X-PHASE1: 1111, X-PHASE0: 1111
11111111  YSTATUS: Y-PHASE1: 1111, Y-PHASE0: 1111
11111111  XSTATUS: X-PHASE3: 1111, X-PHASE2: 1111
11111111  YSTATUS: Y-PHASE3: 1111, Y-PHASE2: 1111
01110000  XSTATUS: X-PHASE5: 0111, X-PHASE4: 0000
01110100  YSTATUS: Y-PHASE5: 0111, Y-PHASE4: 0100
00111111  XSTATUS: X-PHASE7: 0011, X-PHASE6: 1111
01111111  YSTATUS: Y-PHASE7: 0111, Y-PHASE6: 1111
11111111  XSTATUS: X-DVALID: 11111111
11111111  YSTATUS: Y-DVALID: 11111111
11111111  XSTATUS: X-SE_CLK: 11111111
11111111  YSTATUS: Y-SE_CLK: 11111111
11111111  XSTATUS: X-DUMP_EN: 11111111
11111111  YSTATUS: Y-DUMP_EN: 11111111
00000001  XSTATUS: X-TIMESTAMP=1 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000001  YSTATUS: Y-TIMESTAMP=1 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.12.3: content of file mcb_reg40.txt.

6 **XY-SCHID_SYNC** and **XY-DUMP_SYNC** are excluded from the group of bits with explicitly expected states.

2.12.5 MCSR status flags

The post simulation script of most test cases calls the function *check_register_capture* which checks, among other things, for both expected and consistent states of the **MCSR** status bits. The *expected* states are encompassed by a template (defined in the post simulation script) of **MCSR** bit states which is built from the test case descriptions within the test plan document ([TVP-02]). Pass or failure of a template match is reported in the log as described in 2.5.1 . *Consistency* of **MCSR** bits **AOV**, **OVR** and **SyncER** with **XYSTATUS** register contents and the status bits contained within each output frame is checked and the result reported in the log. A warning is also logged if **TvER** is set while the test vector generation bit, **TVEN** is disabled.

2.13 correlation hold off

Test cases: 42

2.13.1 test case 42

Synopsis: the accumulated bias associated with a given stream number, *n*, is *2n* less than the bias associated with stream 0, as expected. The bias is tabulated in Table 2.13.1.

Notes: the advisory in the log of function *check_register_capture* is expected as a consequence of the increasing hold off with stream number.

```
***--------------------------------------------------------------------------***
tc42_postsim.m : V2.00 run time: Fri Sep 30 16:49:32 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 42.
test case version: v2,05apr26
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (single dump) timestamp (word 0): 4294967295

ADVISORIES encountered during register status and data capture checks;
                       details in log file.

Check OK: DESSR bits are set as expected.

Writing dump capture table.
Writing data ID capture table.
***--------------------------------------------------------------------------***
```

Text 2.13.1 console log for script tc42_postsim.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

ADVISORY from check_register_capture: integration dead time differs
        between streams; on cells reporting no overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

Check OK: Captured X,Y ID data (SID, SBID, BBID) are set
          to expected values.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                             template of expected states.
```

Text 2.13.2 test case 42; check register and captured data log; ADVISORIES noted.

stream	*applied dump*	*bias:!OVR*	*applied dump-bias:!OVR*	*bias:OVR*	*dump-bias:OVR*
0	10000	9420	580	0	n-a
1	10000	9418	582	0	n-a
2	10000	9416	584	0	n-a
3	10000	9414	586	0	n-a
4	10000	9412	588	0	n-a
5	10000	9410	590	0	n-a
6	10000	9408	592	0	n-a
7	10000	9406	594	0	n-a

Table 2.13.1 test case 42; applied dump intervals and bias.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
00000101  MCSR: X/Y_DS=0 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.13.3 content of file mcb_reg42.txt.

2.14 frame abort

Test cases: 43

2.14.1 test case 43

Synopsis: alternate dumps of a given chain are discarded as expected. A list of chains that dump for each timestamp are included in the post simulation log of Table 2.14.1.

Notes: dumps containing the string, "00000000 ***ABORTING FRAME***" in the test bench output file, lag_frames43.txt were expunged with a text editor before application of the post simulation script. This string marks aborted frames.

```
***--------------------------------------------------------------------------***
tc43_postsim.m : V2.00 run time: Fri Sep 30 09:59:55 2005
Using postsim_lib: 1.40: 05-08-18

44 frames read for test case 43.
test case version: v2,05apr24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 4.
0(0:3)0  2(4:7)2  4(8:11)4  6(12:15)6


Initial (dump= 1) timestamp (word 0): 222333445
Missing timestamps (not increment by 1).

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.
***--------------------------------------------------------------------------***

octave:4> tc43_print_timestamps
222333445: 0(0:3)0  2(4:7)2
222333447: 0(0:3)0  2(4:7)2  4(8:11)4
222333449: 0(0:3)0  2(4:7)2
222333451: 0(0:3)0  2(4:7)2  4(8:11)4  6(12:15)6
```

Text 2.14.1 console output of script tc43_postsim, tc43_print_timestamps.

stream	***dump***	***bias:!OVR***	***dump-bias:!OVR***	***bias:OVR***	***dump-bias:OVR***
0	10000	9724	276	0	n-a
2	10000	9724	276	0	n-a
4	20000	19724	276	0	n-a
6	40000	39724	276	0	n-a

Table 2.14.1 test case 43; applied dump interval and bias.

```
$ ------------------------------------------------------------------------------
# TEST 43:
# ver: v2,05apr24
# Synopsis:   CONCAT CELLS IN QUADS, 4 INPUTS, 4 DUMP RATES ON X, FRAME ABORT
# Description:
#              Purpose of Test:      check frame abort
#              Concatenation:        0(0:3)0, 2(4:7)2, 4(8:11)4, 6(12:15)6
#              Num of Dumps-input:   8-0,8-2,4-4,2-6
#              Dump Control:         X
#              Num samples/dump-inp: 10000-0, 10000-2, 20000-4, 40000-6
#              Distinct samp/dump:   10000-0, 10000-2, 20000-4, 40000-6
#              Active SE clocks:     0, 2, 4, 6
#              Modulo se_clk gen:    none
#              Compare result with:  check for frame abort first and every other dump
#              Additional details:   frame abort applied
#
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
10000000  CCCSCR_2_3:   CCC3: Y=sec X=adj   CCC2: Y=adj X=adj
00000001  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=pri
10000000  CCCSCR_6_7:   CCC7: Y=sec X=adj   CCC6: Y=adj X=adj
00000001  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=pri
10000000  CCCSCR_10_11: CCC11: Y=sec X=adj CCC10: Y=adj X=adj
00000001  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=pri
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
10101010  XSTATUS: X-SE_CLK: 10101010
10101010  YSTATUS: Y-SE_CLK: 10101010
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.14.2 content of file mcb_reg43.txt.

2.15 dump command capture

Test cases: 41, 42, 46, 47

2.15.1 organization

The following elements are generated for each test case, unless noted otherwise:

(1) a console log of post simulation scripts,

(2) the register capture log if any warnings or advisories are issued,

(3) a table of data captured from the dump control line,

(4) content of the MCB register file.

2.15.2 test case 41

Synopsis: expected data, presented in Table 2.15.1, was captured from dump control.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc41_postsim.m : V2.00 run time: Fri Sep 30 18:08:40 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 41.
test case version: v2,05apr26
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (single dump) timestamp (word 0): 222333444

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Writing dump capture table.
Writing data ID capture table.
***--------------------------------------------------------------------------***
```

Text 2.15.1 console log for post simulation scripts, tc41_postsim.

chain	***cmmd***	***Phase_bin***	***RECIRC_BLK_X***	***RECIRC_BLK_Y***
0(0:1)0	000	21840	80	0
1(2:3)1	001	21841	81	0
2(4:5)2	010	21842	82	0
3(6:7)3	011	21843	83	0
4(8:9)4	110	21844	84	0
5(10:11)5	101	21845	85	0
6(12:13)6	011	21846	86	0
7(14:15)7	111	21847	87	0

Table 2.15.1 test case 41; data captured from dump control lines; dump control X.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.15.2 content of file mcb_reg41.txt.

2.15.3 test case 42

Synopsis: expected data, presented in Table 2.15.2, was captured from dump control.

Notes: see section 2.13 for the post simulation log, register capture log and mcb_reg42.txt file contents.

chain	***cmmd***	***Phase_bin***	***RECIRC_BLK_X***	***RECIRC_BLK_Y***
0(0:1)0	011	43690	0	80
1(2:3)1	101	43691	0	81
2(4:5)2	110	43692	0	82
3(6:7)3	010	43693	0	83
4(8:9)4	111	43694	0	84
5(10:11)5	010	43695	0	85
6(12:13)6	001	43696	0	86
7(14:15)7	000	43697	0	87

Table 2.15.2 test case 42; data captured from dump control lines; dump control Y.

2.15.4 test case 46

Synopsis: expected data, presented in Table 2.15.3, was captured from dump control.

Notes: n-a

```
****--------------------------------------------------------------------------***
tc46_postsim.m : V2.00 run time: Mon Sep 26 18:37:54 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 46.
test case version: v2,05apr24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (single dump) timestamp (word 0): 222333444

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Writing dump capture table.
Writing data ID capture table.
***--------------------------------------------------------------------------***
octave:12> mcb46.X_DUMP_EN'
ans =

  0  1  0  1  0  1  0  1

octave:13> mcb46.Y_DUMP_EN'
ans =

  0  0  0  0  0  0  0  0
```

Text 2.15.3 console log for script tc46_postsim and ad hoc tests.

chain	*cmmd*	*Phase_bin*	*RECIRC_BLK_X*	*RECIRC_BLK_Y*
0(0:1)0	000	21840	170	0
1(2:3)1	001	21841	171	0
2(4:5)2	010	21842	172	0
3(6:7)3	011	21843	173	0
4(8:9)4	010	21844	174	0
5(10:11)5	010	21845	175	0
6(12:13)6	011	21846	176	0
7(14:15)7	011	21847	177	0

Table 2.15.3 test case 46; data captured from dump control lines; dump control X.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
11000101  MCSR: X/Y_DS=1 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
10101010  XSTATUS: X-DUMP_EN: 10101010
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.15.4 content of file mcb_reg46.txt.

2.15.5 test case 47

Synopsis: expected data, presented in Table 2.15.4, was captured from dump control.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc47_postsim.m : V2.00 run time: Mon Sep 26 18:39:20 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 47.
test case version: v2,05apr24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 8.
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


Initial (single dump) timestamp (word 0): 4294967295

Check OK: no inconsistencies reported in captured data
          or register contents; no unexpected register reports;
          details in log file.

Check OK: DESSR bits are set as expected.

Writing dump capture table.
Writing data ID capture table.
***--------------------------------------------------------------------------***
octave:17> mcb47.X_DUMP_EN'
ans =

  0  0  0  0  0  0  0  0

octave:18> mcb47.Y_DUMP_EN'
ans =

  1  0  1  0  1  0  1  0
```

Text 2.15.5 console log for script tc47_postsim and ad hoc tests.

chain	***cmmd***	***Phase_bin***	***RECIRC_BLK_X***	***RECIRC_BLK_Y***
0(0:1)0	000	21840	0	170
1(2:3)1	001	21841	0	171
2(4:5)2	010	21842	0	172
3(6:7)3	011	21843	0	173
4(8:9)4	010	21844	0	174
5(10:11)5	010	21845	0	175
6(12:13)6	011	21846	0	176
7(14:15)7	011	21847	0	177

Table 2.15.4 test case 47; data captured from dump control lines; dump control Y.

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
01000101  MCSR: X/Y_DS=0 SyncER=1 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

10000001  CCCSCR_0_1:   CCC1: Y=sec X=adj   CCC0: Y=adj X=pri
01000010  CCCSCR_2_3:   CCC3: Y=pri X=adj   CCC2: Y=adj X=sec
10000001  CCCSCR_4_5:   CCC5: Y=sec X=adj   CCC4: Y=adj X=pri
01000010  CCCSCR_6_7:   CCC7: Y=pri X=adj   CCC6: Y=adj X=sec
10000001  CCCSCR_8_9:   CCC9: Y=sec X=adj   CCC8: Y=adj X=pri
01000010  CCCSCR_10_11: CCC11: Y=pri X=adj CCC10: Y=adj X=sec
10000001  CCCSCR_12_13: CCC13: Y=sec X=adj CCC12: Y=adj X=pri
01000010  CCCSCR_14_15: CCC15: Y=pri X=adj CCC14: Y=adj X=sec

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
01010101  YSTATUS: Y-DUMP_EN: 01010101
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.15.6 content of file mcb_reg47.txt.

2.16 timestamp capture

Test cases: 17, 18, 19, 20

2.16.1 organization

Post simulation script logs and other related items pertaining to test cases 17 through 20 are found in sections 2.7.4 through 2.7.7, respectively. Tables of captured timestamps are presented in this section for each of the four test cases.

The timestamp increment between dumps is checked within the post simulation script of *all* multiple dump test cases and a report written to the respective console logs. Refer to the section containing the log for a particular test case.

2.16.2 test case 17

Synopsis: expected timestamps are presented in Table 2.16.1.

Notes: see section 2.7.4 for post simulation script logs.

dump	*dump control*	*TIMESTAMP word1 (hexadecimal)*	*TIMESTAMP word0 (hexadecimal)*
1	X	1431655765 (55555555)	2863311530 (AAAAAAAA)
2	X	1431655765 (55555555)	2863311531 (AAAAAAAB)

Table 2.16.1 test case 17; captured timestamp.

2.16.3 test case 18

Synopsis: expected timestamps are presented in Table 2.16.2.

Notes: see section 2.7.5 for post simulation script logs.

dump	*dump control*	*TIMESTAMP word1 (hexadecimal)*	*TIMESTAMP word0 (hexadecimal)*
1	X	2863311530 (AAAAAAAA)	1431655765 (55555555)
2	X	2863311530 (AAAAAAAA)	1431655766 (55555556)

Table 2.16.2 test case 18; captured timestamp.

2.16.4 test case 19

Synopsis: expected timestamps are presented in Table 2.16.3.

Notes: see section 2.7.6 for post simulation script logs.

dump	*dump control*	*TIMESTAMP word1 (hexadecimal)*	*TIMESTAMP word0 (hexadecimal)*
1	Y	1431655765 (55555555)	2863311530 (AAAAAAAA)
2	Y	1431655765 (55555555)	2863311531 (AAAAAAAB)

Table 2.16.3 test case 19; captured timestamp.

2.16.5 test case 20

Synopsis: expected timestamps are presented in Table 2.16.4.

Notes: see section 2.7.7 for post simulation script logs.

dump	*dump control*	*TIMESTAMP word1 (hexadecimal)*	*TIMESTAMP word0 (hexadecimal)*
1	Y	2863311530 (AAAAAAAA)	1431655765 (55555555)
2	Y	2863311530 (AAAAAAAA)	1431655766 (55555556)

Table 2.16.4 test case 20; captured timestamp.

2.17 data stream ID capture

Test cases: 17, 18, 19, 20

2.17.1 organization

Post simulation script logs and other related items pertaining to test cases 17 through 20 are found in sections 2.7.4 through 2.7.7, respectively. Expected **SID-XY**, **SBID-XY** and **BBID-XY** are generated within the function *check_register_capture* from the supplied template values (defined in the post simulation script) which are built from the test case descriptions within the test plan document ([TVP-02]). These expected values are compared against those captured and a warning issued in the register capture log if the comparison fails. This test is performed for most test cases, including those of this section, making explicit tabulation of these captured data redundant. However, tables of these data are included in this section as an additional check.

2.17.2 test case 17

Synopsis: expected data stream ID data are presented in Table 2.17.1.

Notes: see section 2.7.4 for post simulation script logs.

chain	***SID_X***	***SID_Y***	***SBID_X***	***SBID_Y***	***BBID_X***	***BBID_Y***
1(0:1)1	85	170	22	11	2	2
1(4:5)3	85	170	22	13	2	4
1(8:9)5	85	170	22	15	2	6
1(12:13)7	85	170	22	17	2	0

Table 2.17.1 test case 17; captured stream ID data.

2.17.3 test case 18

Synopsis: expected data stream ID data are presented in Table 2.17.2.

Notes: see section 2.7.5 for post simulation script logs.

chain	***SID_X***	***SID_Y***	***SBID_X***	***SBID_Y***	***BBID_X***	***BBID_Y***
1(0:1)1	85	170	22	11	3	3
3(4:5)1	85	170	24	11	5	3
5(8:9)1	85	170	26	11	7	3
7(12:13)1	85	170	28	11	1	3

Table 2.17.2 test case 18; captured stream ID data.

2.17.4 test case 19

Synopsis: expected data stream ID data are presented in Table 2.17.3.

Notes: see section 2.7.6 for post simulation script logs.

chain	***SID_X***	***SID_Y***	***SBID_X***	***SBID_Y***	***BBID_X***	***BBID_Y***
0(0:1)0	170	85	10	21	3	3
0(4:5)2	170	85	10	23	3	5
0(8:9)4	170	85	10	25	3	7
0(12:13)6	170	85	10	27	3	1

Table 2.17.3 test case 19; captured stream ID data.

2.17.5 test case 20

Synopsis: expected data stream ID data are presented in Table 2.17.4.

Notes: see section 2.7.7 for post simulation script logs.

chain	*SID_X*	*SID_Y*	*SBID_X*	*SBID_Y*	*BBID_X*	*BBID_Y*
0(0:1)0	170	85	10	21	4	4
2(4:5)0	170	85	12	21	6	4
4(8:9)0	170	85	14	21	0	4
6(12:13)0	170	85	16	21	2	4

Table 2.17.4 test case 20; captured stream ID data.

2.18 enable phase rotation

Test cases: 44

2.18.1 test case 44

Synopsis: The 128 output lag coefficients are real (see the ad hoc check in the console log of Text 2.18.1) and the lag coefficient plot of Illustration 2.18.1 shows no apparent structure, as expected. However, this test is incomplete since an oversight in this version (v2,05apr24) of the test case source file generally resets all input streams with exception of stream 0 (involving chains 2 and 4).

Notes: the error in Text 2.18.1 regarding unexpected **DESSR** states is a consequence of the reset input streams. The data similarity checks between like streams succeeds for chains 2 and 4, as expected. This check will also succeed for other chains since, in this test, their associated data streams are generally reset, resulting in only a bias accumulation.

```
***--------------------------------------------------------------------------***
tc44_postsim.m : V2.00 run time: Sat Oct 01 14:46:56 2005
Using postsim_lib: 1.40: 05-08-18

2 frames read for test case 44.
test case version: v2,05apr24
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).

Chip configuration obtained from MCSR regsister settings.
List chains from 1 to 16.
1(0:0)1  0(1:1)0  1(2:2)1  0(3:3)0
2(4:4)2  2(5:5)2  3(6:6)3  3(7:7)3
5(8:8)5  5(9:9)5  4(10:10)4  4(11:11)4
6(12:12)6  7(13:13)7  6(14:14)6  7(15:15)7


Initial (single dump) timestamp (word 0): 222333444

WARNING: DESSR bits are set NOT as expected.
***--------------------------------------------------------------------------***

octave:71> rows(find(real(dump44(1).CCchain(1).clag)))
ans = 128
octave:72> rows(find(imag(dump44(1).CCchain(1).clag)))
ans = 0
octave:73> 40000-dump44(1).CCchain(1).DATA_BIAS/64
ans = 276

octave:74> tc44_similarity_check

No differences exceeding the floating point precision limit of this
        machine were found between similar chains of test case 44.
```

Text 2.18.1 console log for scripy tc44_postsim, tc44_similarity_check.

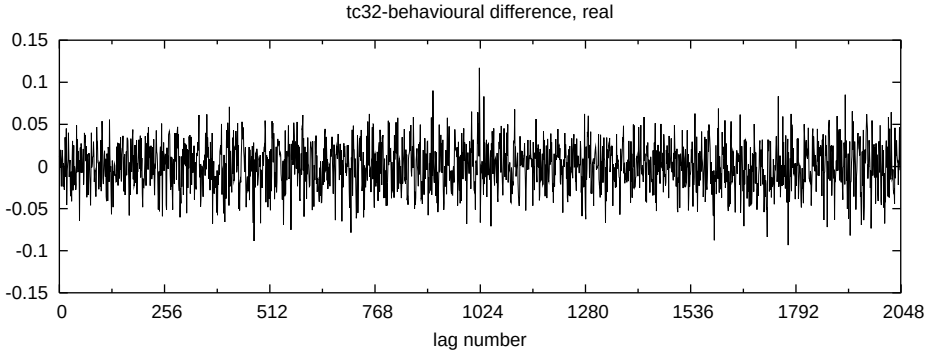

Illustration 2.18.1 Phase rotation disabled; no correlation evident.

```
# ------------------------------------------------------------------------------
# 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)
#
# ------------------------------------------------------------------------------
10000001  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=0 TVEN=0 CEN=1

10010110  CCCSCR_0_1:   CCC1: Y=sec X=pri   CCC0: Y=pri X=sec
10010110  CCCSCR_2_3:   CCC3: Y=sec X=pri   CCC2: Y=pri X=sec
10011001  CCCSCR_4_5:   CCC5: Y=sec X=pri   CCC4: Y=sec X=pri
01100110  CCCSCR_6_7:   CCC7: Y=pri X=sec   CCC6: Y=pri X=sec
01100110  CCCSCR_8_9:   CCC9: Y=pri X=sec   CCC8: Y=pri X=sec
10011001  CCCSCR_10_11: CCC11: Y=sec X=pri CCC10: Y=sec X=pri
01101001  CCCSCR_12_13: CCC13: Y=pri X=sec CCC12: Y=sec X=pri
01101001  CCCSCR_14_15: CCC15: Y=pri X=sec CCC14: Y=sec X=pri

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
00000000  CCQR_2_3: Q3: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

11110101  DESSR_0_7: 11110101
11111111  DESSR_8_15: 11111111
```

Text 2.18.2 content of file mcb_reg44.txt.

2.19 enable correlation

Test cases: 45

2.19.1 test case 45

Synopsis: output lag coefficients are not generated (see lag frame file contents, Text 2.19.1).

Notes: n-a

```
# ------------------------------------------------------------------------------
# TEST 45:
# ver: v2,05jun14
# 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: 1000-0
#              Distinct samp/dump:   1000-0
#              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:
#
# ------------------------------------------------------------------------------
```

Text 2.19.1 complete content of the file lag_frames45.txt; output lag coefficients were not generated.

```
# ------------------------------------------------------------------------------
# TEST 45:
# ver: v2,05jun14
# 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: 1000-0
#              Distinct samp/dump:   1000-0
#              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:
#
# ------------------------------------------------------------------------------
10000100  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=0

00000001  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=pri
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
10000000  CCCSCR_14_15: CCC15: Y=sec X=adj CCC14: Y=adj X=adj

00000000  CCQR_0_1: Q1: YSw1=pri YSw0=pri XSw1=pri XSw0=pri  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
01000000  CCQR_2_3: Q3: YSw1=pri YSw0=sec XSw1=pri XSw0=pri  Q2: YSw1=pri YSw0=pri XSw1=pri XSw0=pri

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
00000000  YSTATUS: Y-SDATA1: 0000, Y-SDATA0: 0000
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
00000000  YSTATUS: Y-SDATA3: 0000, Y-SDATA2: 0000
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
00000000  YSTATUS: Y-SDATA5: 0000, Y-SDATA4: 0000
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
00000000  YSTATUS: Y-SDATA7: 0000, Y-SDATA6: 0000
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
00000000  YSTATUS: Y-DVALID: 00000000
00000000  XSTATUS: X-SE_CLK: 00000000
00000000  YSTATUS: Y-SE_CLK: 00000000
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.19.2 content of file mcb_reg45.txt.

2.20 autocorrelation

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

2.20.1 organization

A console log and (magnitude and phase) frequency domain plots are provided for each test case. Validation consists of observing the expected number of spectral magnitude lines and linear phase behaviour. The MCB register contents file contains limited information for autocorrelation mode but is included for completeness.

2.20.2 test case 56

Synopsis: expected features are found in the plots of Illustration 2.20.1 and Illustration 2.20.2.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc56_postsim.m : V1.10 run time: Sat Oct 01 16:19:27 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 56.
test case version: v2,05apr20
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


MSSG from add_CCchain_dump: autocorrelation mode detected.

Initial (single dump) timestamp (word 0): 222333444

Writing plot files
***--------------------------------------------------------------------------***
```

Text 2.20.1 console log for script tc56_postsim.

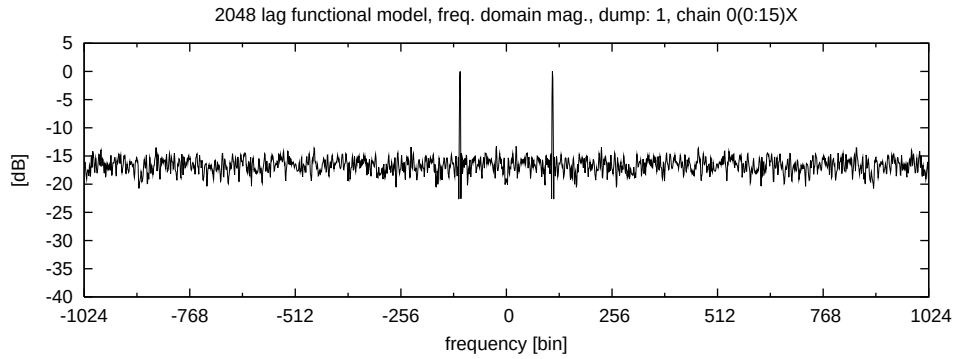

Illustration 2.20.1

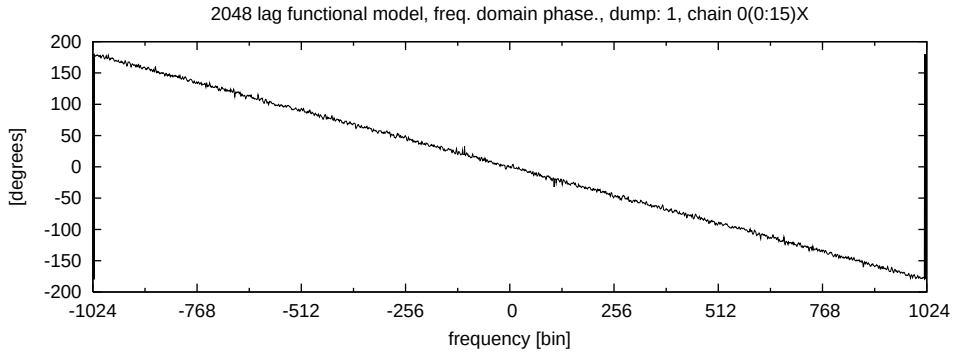

Illustration 2.20.2

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000000  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=adj X=adj
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
00000000  CCCSCR_14_15: CCC15: Y=adj X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
xxxxxxxx  YSTATUS: Y-SDATA1: xxxx, Y-SDATA0: xxxx
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
xxxxxxxx  YSTATUS: Y-SDATA3: xxxx, Y-SDATA2: xxxx
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
xxxxxxxx  YSTATUS: Y-SDATA5: xxxx, Y-SDATA4: xxxx
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
xxxxxxxx  YSTATUS: Y-SDATA7: xxxx, Y-SDATA6: xxxx
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
xxxxxxxx  YSTATUS: Y-DVALID: xxxxxxxx
11111110  XSTATUS: X-SE_CLK: 11111110
11111110  YSTATUS: Y-SE_CLK: 11111110
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.20.2 content of file mcb_reg56.txt.

2.20.3 test case 57

Synopsis: expected features are found in the plots of Illustration 2.20.3 and Illustration 2.20.4.

Notes: narrow vertical lines in the phase plot occur at points adjacent to lines in the magnitude plot and result from a sign change in small values, likely due to noise or artifacts such as ripple in the frequency domain.

```
***--------------------------------------------------------------------------***
tc57_postsim.m : V1.10 run time: Sat Oct 01 16:44:31 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 57.
test case version: v2,05apr20
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


MSSG from add_CCchain_dump: autocorrelation mode detected.

Initial (single dump) timestamp (word 0): 222333444

Writing plot files
***--------------------------------------------------------------------------***
```

Text 2.20.3 console log for script tc57_postsim.

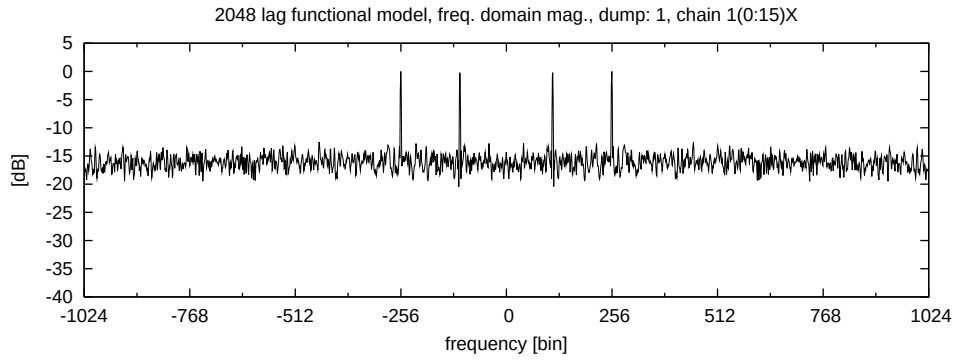

Illustration 2.20.3

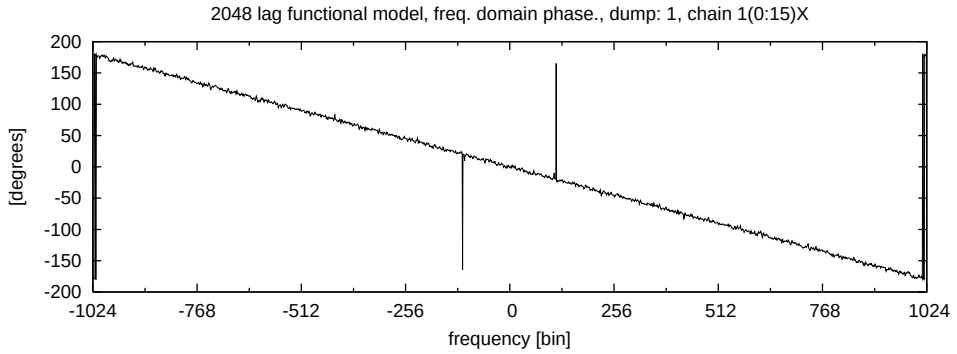

Illustration 2.20.4

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00000100  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=pri X=adj
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
00000000  CCCSCR_14_15: CCC15: Y=adj X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
xxxxxxxx  YSTATUS: Y-SDATA1: xxxx, Y-SDATA0: xxxx
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
xxxxxxxx  YSTATUS: Y-SDATA3: xxxx, Y-SDATA2: xxxx
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
xxxxxxxx  YSTATUS: Y-SDATA5: xxxx, Y-SDATA4: xxxx
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
xxxxxxxx  YSTATUS: Y-SDATA7: xxxx, Y-SDATA6: xxxx
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
xxxxxxxx  YSTATUS: Y-DVALID: xxxxxxxx
11111101  XSTATUS: X-SE_CLK: 11111101
11111101  YSTATUS: Y-SE_CLK: 11111101
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.20.4 content of file mcb_reg57.txt.

2.20.4 test case 58

Synopsis: expected features are found in the plots of Illustration 2.20.5 and Illustration 2.20.6.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc58_postsim.m : V1.10 run time: Sat Oct 01 16:45:12 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 58.
test case version: v2,05apr20
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


MSSG from add_CCchain_dump: autocorrelation mode detected.

Initial (single dump) timestamp (word 0): 222333444

Writing plot files
***--------------------------------------------------------------------------***
```

Text 2.20.5 console log for script tc58_postsim.

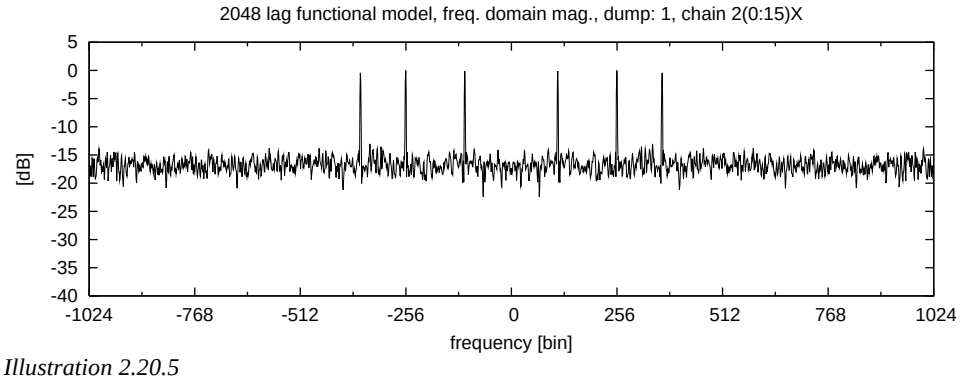

Illustration 2.20.5

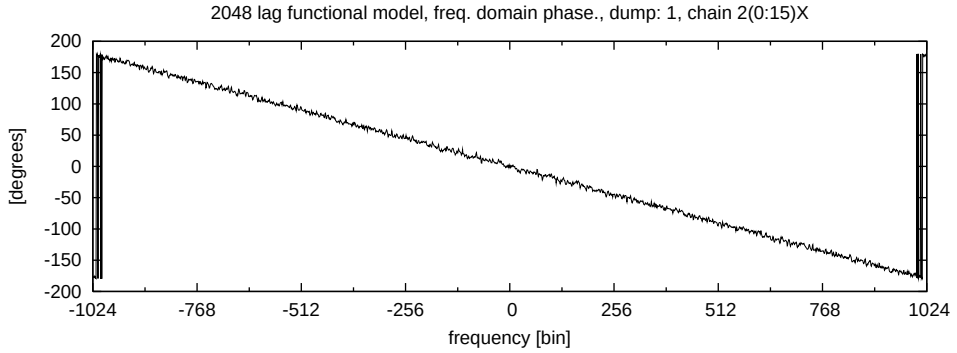

Illustration 2.20.6

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00001000  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=sec X=adj
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
00000000  CCCSCR_14_15: CCC15: Y=adj X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
xxxxxxxx  YSTATUS: Y-SDATA1: xxxx, Y-SDATA0: xxxx
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
xxxxxxxx  YSTATUS: Y-SDATA3: xxxx, Y-SDATA2: xxxx
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
xxxxxxxx  YSTATUS: Y-SDATA5: xxxx, Y-SDATA4: xxxx
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
xxxxxxxx  YSTATUS: Y-SDATA7: xxxx, Y-SDATA6: xxxx
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
xxxxxxxx  YSTATUS: Y-DVALID: xxxxxxxx
11111011  XSTATUS: X-SE_CLK: 11111011
11111011  YSTATUS: Y-SE_CLK: 11111011
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.20.6 content of file mcb_reg58.txt.

2.20.5 test case 59

Synopsis: expected features are found in the plots of Illustration 2.20.7 and Illustration 2.20.8.

Notes: n-a

```
***--------------------------------------------------------------------------***
tc59_postsim.m : V1.10 run time: Sat Oct 01 16:46:13 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 59.
test case version: v2,05apr20
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


MSSG from add_CCchain_dump: autocorrelation mode detected.

Initial (single dump) timestamp (word 0): 222333444

Writing plot files
***--------------------------------------------------------------------------***
```

Text 2.20.7 console log for script tc59_postsim.

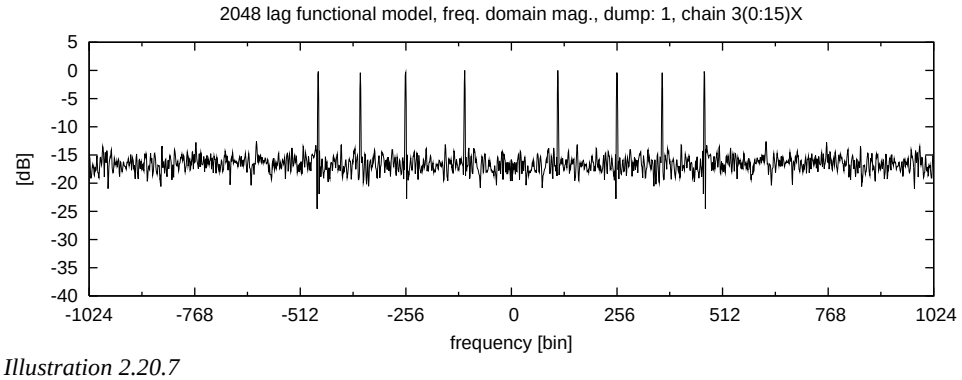

Illustration 2.20.7

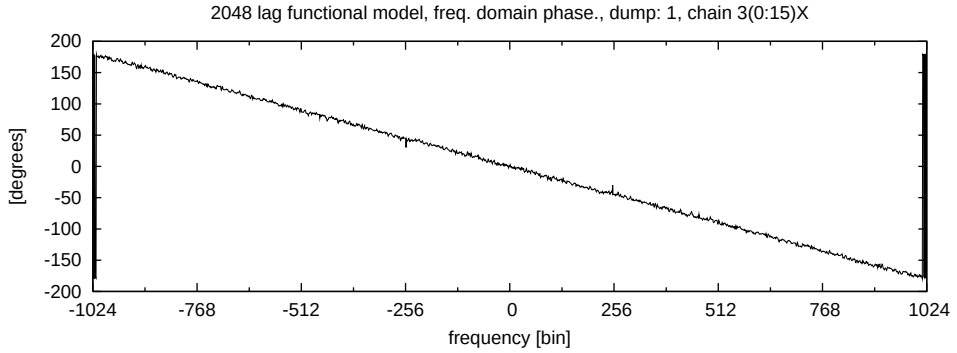

Illustration 2.20.8

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00001100  CCCSCR_0_1:   CCC1: Y=adj X=adj   CCC0: Y=n/c X=adj
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
00000000  CCCSCR_14_15: CCC15: Y=adj X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
xxxxxxxx  YSTATUS: Y-SDATA1: xxxx, Y-SDATA0: xxxx
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
xxxxxxxx  YSTATUS: Y-SDATA3: xxxx, Y-SDATA2: xxxx
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
xxxxxxxx  YSTATUS: Y-SDATA5: xxxx, Y-SDATA4: xxxx
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
xxxxxxxx  YSTATUS: Y-SDATA7: xxxx, Y-SDATA6: xxxx
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
xxxxxxxx  YSTATUS: Y-DVALID: xxxxxxxx
11110111  XSTATUS: X-SE_CLK: 11110111
11110111  YSTATUS: Y-SE_CLK: 11110111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.20.8 content of file mcb_reg59.txt.

2.20.6 test case 60

Synopsis: expected features are found in the plots of Illustration 2.20.9 and Illustration 2.20.10.

Notes: narrow vertical lines in the phase plot occur at points adjacent to lines in the magnitude plot and result from a sign change in small values, likely due to noise or artifacts such as ripple in the frequency domain.

```
***--------------------------------------------------------------------------***
tc60_postsim.m : V1.10 run time: Sat Oct 01 16:47:04 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 60.
test case version: v2,05apr07
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


MSSG from add_CCchain_dump: autocorrelation mode detected.

Initial (single dump) timestamp (word 0): 222333444

Writing plot files
***--------------------------------------------------------------------------***
```

Text 2.20.9 console log for script tc60_postsim.

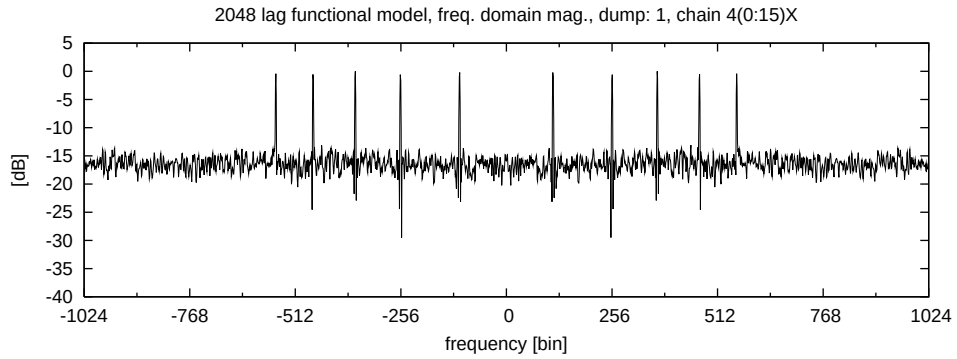

Illustration 2.20.9

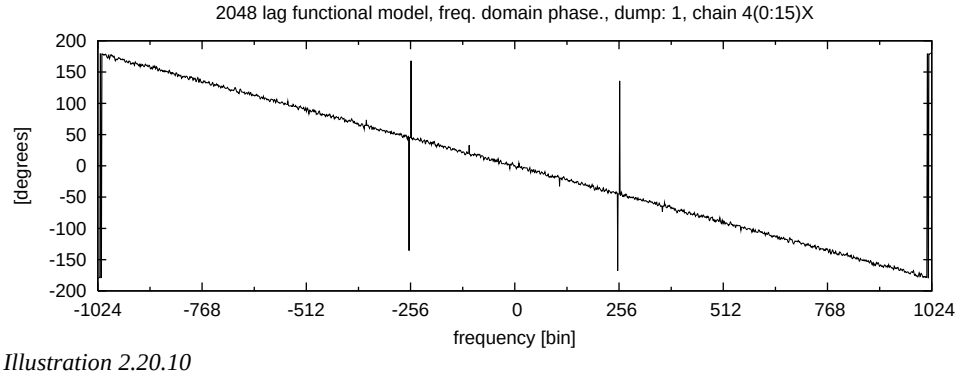

Illustration 2.20.10

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00010000  CCCSCR_0_1:   CCC1: Y=adj X=pri   CCC0: Y=adj X=adj
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
00000000  CCCSCR_14_15: CCC15: Y=adj X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
xxxxxxxx  YSTATUS: Y-SDATA1: xxxx, Y-SDATA0: xxxx
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
xxxxxxxx  YSTATUS: Y-SDATA3: xxxx, Y-SDATA2: xxxx
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
xxxxxxxx  YSTATUS: Y-SDATA5: xxxx, Y-SDATA4: xxxx
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
xxxxxxxx  YSTATUS: Y-SDATA7: xxxx, Y-SDATA6: xxxx
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
xxxxxxxx  YSTATUS: Y-DVALID: xxxxxxxx
11101111  XSTATUS: X-SE_CLK: 11101111
11101111  YSTATUS: Y-SE_CLK: 11101111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.20.10 content of file mcb_reg60.txt.

2.20.7 test case 61

Synopsis: expected features are found in the plots of Illustration 2.20.11 and Illustration 2.20.12.

Notes: narrow vertical lines in the phase plot occur at points adjacent to lines in the magnitude plot and result from a sign change in small values, likely due to noise or artifacts such as ripple in the frequency domain.

```
***--------------------------------------------------------------------------***
tc61_postsim.m : V1.10 run time: Sat Oct 01 17:03:26 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 61.
test case version: v2,05aug07
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


MSSG from add_CCchain_dump: autocorrelation mode detected.

Initial (single dump) timestamp (word 0): 222333444

Writing plot files
***--------------------------------------------------------------------------***
```

Text 2.20.11 console log for script tc61_postsim.

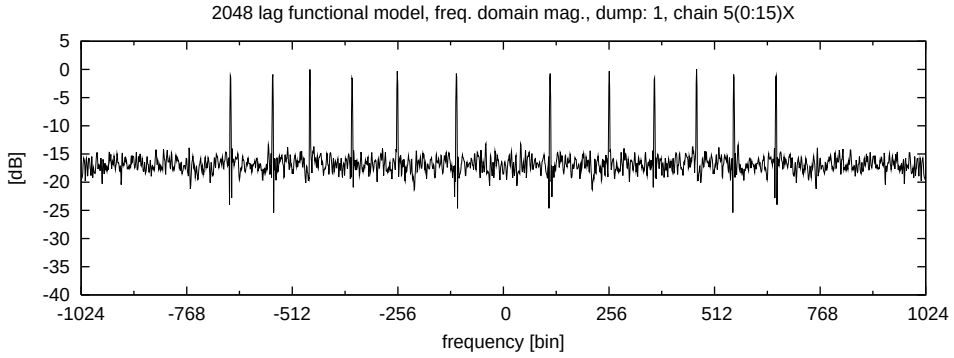

Illustration 2.20.11

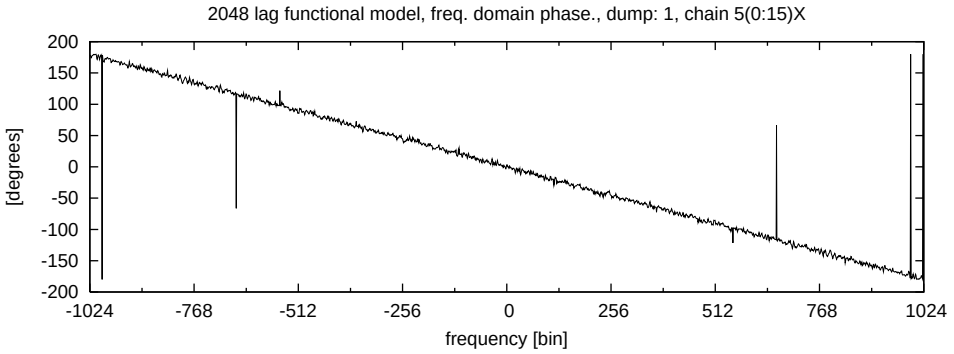

Illustration 2.20.12

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00010100  CCCSCR_0_1:   CCC1: Y=adj X=pri   CCC0: Y=pri X=adj
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
00000000  CCCSCR_14_15: CCC15: Y=adj X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
xxxxxxxx  YSTATUS: Y-SDATA1: xxxx, Y-SDATA0: xxxx
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
xxxxxxxx  YSTATUS: Y-SDATA3: xxxx, Y-SDATA2: xxxx
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
xxxxxxxx  YSTATUS: Y-SDATA5: xxxx, Y-SDATA4: xxxx
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
xxxxxxxx  YSTATUS: Y-SDATA7: xxxx, Y-SDATA6: xxxx
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
xxxxxxxx  YSTATUS: Y-DVALID: xxxxxxxx
11011111  XSTATUS: X-SE_CLK: 11011111
11011111  YSTATUS: Y-SE_CLK: 11011111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.20.12 content of file mcb_reg61.txt.

2.20.8 test case 62

Synopsis: expected features are found in the plots of Illustration 2.20.13 and Illustration 2.20.14.

Notes: narrow vertical lines in the phase plot occur at points adjacent to lines in the magnitude plot and result from a sign change in small values, likely due to noise or artifacts such as ripple in the frequency domain.

```
***--------------------------------------------------------------------------***
tc62_postsim.m : V1.10 run time: Sat Oct 01 17:03:57 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 62.
test case version: v2,05apr20
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


MSSG from add_CCchain_dump: autocorrelation mode detected.

Initial (single dump) timestamp (word 0): 222333444

Writing plot files
***--------------------------------------------------------------------------***
```

Text 2.20.13 console log for script tc62_postsim.

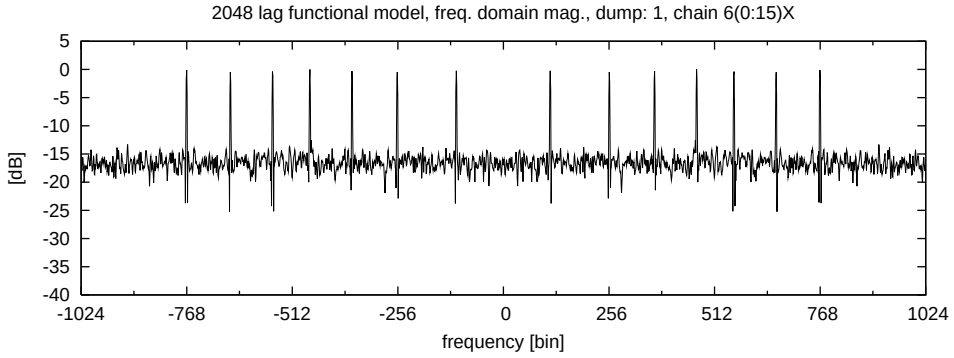

Illustration 2.20.13

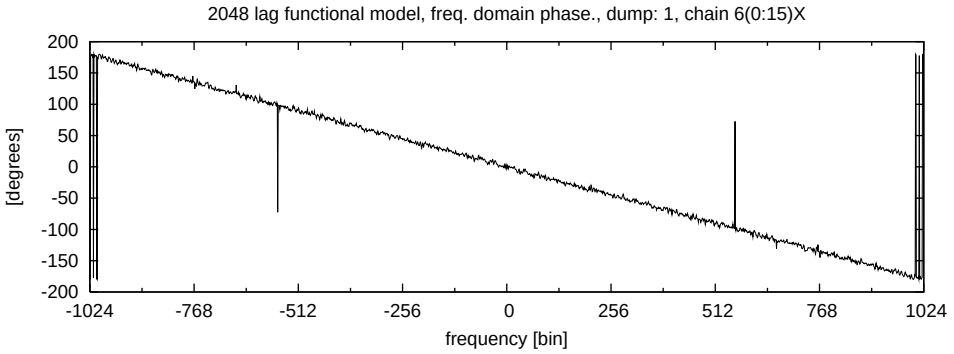

Illustration 2.20.14

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00011000  CCCSCR_0_1:   CCC1: Y=adj X=pri   CCC0: Y=sec X=adj
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
00000000  CCCSCR_14_15: CCC15: Y=adj X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
xxxxxxxx  YSTATUS: Y-SDATA1: xxxx, Y-SDATA0: xxxx
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
xxxxxxxx  YSTATUS: Y-SDATA3: xxxx, Y-SDATA2: xxxx
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
xxxxxxxx  YSTATUS: Y-SDATA5: xxxx, Y-SDATA4: xxxx
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
xxxxxxxx  YSTATUS: Y-SDATA7: xxxx, Y-SDATA6: xxxx
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
xxxxxxxx  YSTATUS: Y-DVALID: xxxxxxxx
10111111  XSTATUS: X-SE_CLK: 10111111
10111111  YSTATUS: Y-SE_CLK: 10111111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.20.14 content of file mcn_reg62.txt.

2.20.9 test case 63

Synopsis: expected features are found in the plots of Illustration 2.20.15 and Illustration 2.20.16.

Notes: narrow vertical lines in the phase plot occur at points adjacent to lines in the magnitude plot and result from a sign change in small values, likely due to noise or artifacts such as ripple in the frequency domain.

```
***--------------------------------------------------------------------------***
tc63_postsim.m : V1.10 run time: Sat Oct 01 17:04:30 2005
Using postsim_lib: 1.40: 05-08-18

16 frames read for test case 63.
test case version: v2,05apr20
Checksum is OK for all frames and corrupt data were not
detected (no 'x' or 'X' hex digits were found).


MSSG from add_CCchain_dump: autocorrelation mode detected.

Initial (single dump) timestamp (word 0): 222333444

Writing plot files
***--------------------------------------------------------------------------***
```

Text 2.20.15 console log for script tc63_postsim.

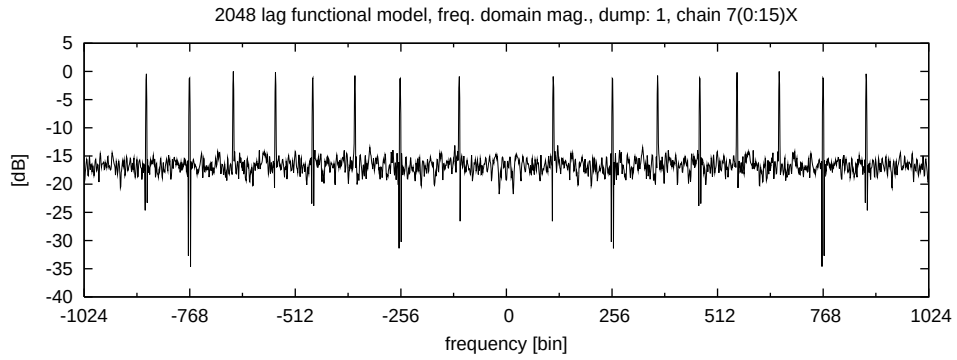

Illustration 2.20.15

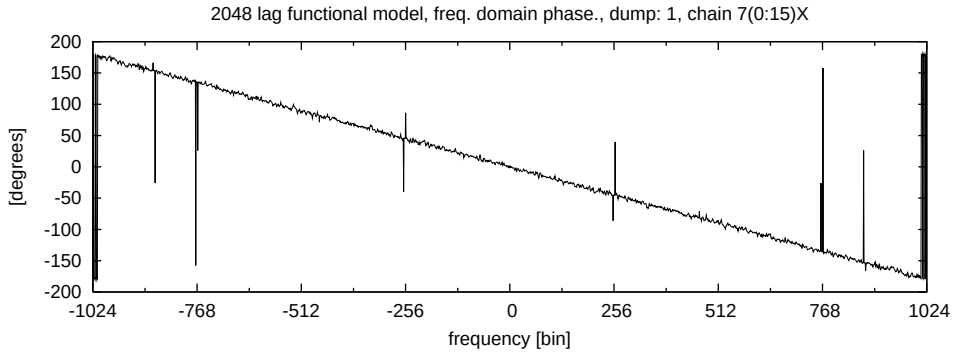

Illustration 2.20.16

```
# ------------------------------------------------------------------------------
# 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:
#
# ------------------------------------------------------------------------------
10000101  MCSR: X/Y_DS=1 SyncER=0 TvER=0 AOV=0 OVR=0 PhEN=1 TVEN=0 CEN=1

00011100  CCCSCR_0_1:   CCC1: Y=adj X=pri   CCC0: Y=n/c X=adj
00000000  CCCSCR_2_3:   CCC3: Y=adj X=adj   CCC2: Y=adj X=adj
00000000  CCCSCR_4_5:   CCC5: Y=adj X=adj   CCC4: Y=adj X=adj
00000000  CCCSCR_6_7:   CCC7: Y=adj X=adj   CCC6: Y=adj X=adj
00000000  CCCSCR_8_9:   CCC9: Y=adj X=adj   CCC8: Y=adj X=adj
00000000  CCCSCR_10_11: CCC11: Y=adj X=adj CCC10: Y=adj X=adj
00000000  CCCSCR_12_13: CCC13: Y=adj X=adj CCC12: Y=adj X=adj
00000000  CCCSCR_14_15: CCC15: Y=adj X=adj CCC14: Y=adj X=adj

11110000  CCQR_0_1: Q1: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q0: YSw1=pri YSw0=pri XSw1=pri XSw0=pri
11111111  CCQR_2_3: Q3: YSw1=sec YSw0=sec XSw1=sec XSw0=sec  Q2: YSw1=sec YSw0=sec XSw1=sec XSw0=sec

00000000  XSTATUS: X-SDATA1: 0000, X-SDATA0: 0000
xxxxxxxx  YSTATUS: Y-SDATA1: xxxx, Y-SDATA0: xxxx
00000000  XSTATUS: X-SDATA3: 0000, X-SDATA2: 0000
xxxxxxxx  YSTATUS: Y-SDATA3: xxxx, Y-SDATA2: xxxx
00000000  XSTATUS: X-SDATA5: 0000, X-SDATA4: 0000
xxxxxxxx  YSTATUS: Y-SDATA5: xxxx, Y-SDATA4: xxxx
00000000  XSTATUS: X-SDATA7: 0000, X-SDATA6: 0000
xxxxxxxx  YSTATUS: Y-SDATA7: xxxx, Y-SDATA6: xxxx
00000000  XSTATUS: X-PHASE1: 0000, X-PHASE0: 0000
00000000  YSTATUS: Y-PHASE1: 0000, Y-PHASE0: 0000
00000000  XSTATUS: X-PHASE3: 0000, X-PHASE2: 0000
00000000  YSTATUS: Y-PHASE3: 0000, Y-PHASE2: 0000
00000000  XSTATUS: X-PHASE5: 0000, X-PHASE4: 0000
00000000  YSTATUS: Y-PHASE5: 0000, Y-PHASE4: 0000
00000000  XSTATUS: X-PHASE7: 0000, X-PHASE6: 0000
00000000  YSTATUS: Y-PHASE7: 0000, Y-PHASE6: 0000
00000000  XSTATUS: X-DVALID: 00000000
xxxxxxxx  YSTATUS: Y-DVALID: xxxxxxxx
01111111  XSTATUS: X-SE_CLK: 01111111
01111111  YSTATUS: Y-SE_CLK: 01111111
00000000  XSTATUS: X-DUMP_EN: 00000000
00000000  YSTATUS: Y-DUMP_EN: 00000000
00000000  XSTATUS: X-TIMESTAMP=0 X-SCHID_SYNC=0 X-DUMP_SYNCs=0
00000000  YSTATUS: Y-TIMESTAMP=0 Y-SCHID_SYNC=0 Y-DUMP_SYNCs=0

00000000  DESSR_0_7: 00000000
00000000  DESSR_8_15: 00000000
```

Text 2.20.16 content of file mcb_reg63.txt.

2.21 missing test cases

Test case 23: The source file for this test case (version: v4,05aug07) contained an incorrect parameter setting which resulted in data stream 7 being generally reset. Since the novelty of this test was based on this data stream, the test as executed provided no additional information. Hence, although output of the test bench was archived, it is not included in this report. Test case 23 is deprecated and has been replaced by test case 68.

Test case 34: This test is redundant and does not contribute significant additional information. Hence, although output of the test bench was archived, it is not included in this report.

Test case 54, 55: These test cases are designed for verification and do not provide significant additional validation over the remaining set of test cases.

Test case 67: This test case is similar to 31 but provides a short single dump for verification purposes. It does not provide additional validation over the remaining set of test cases.

3 Post simulation scripts

3.1 Introduction

3.1.1 organization

This chapter introduces the post simulation process with a summary of the functions involved and then a complete listing of the post simulation scripts, by each operation under test in its own subsection. The order of these subsections mirror that of the previous chapter of post simulation results. The section on register consistency checking, 3.2, is an exception to this organization and describes the function *check_register_capture*.

3.1.2 function summary

A set of functions written in the Octave numerical language was used to process data generated by the (functional) RTL chip model - test bench combination and behavioural model in order to produce data that was then incorporated into the tables, plots and logs of the previous chapter. Most of these functions were collected together as the library *postsim_lib* (Text 3.1.1) and either invoked from within Octave scripts or interactively, via the Octave interpreter, during ad hoc testing. These scripts are referred to generally as *post simulation* scripts. Functions which are not included as part of *postsim_lib* such as *check_register_capture* and those specific to a particular test cases, are identified by (and report) a version number for control purposes. The interrelationship between the functions of this library and data within the context of post simulation scripts is depicted in Illustration 3.1.1. Directional lines represent the flow of data that occurs within the main post simulation script (processing) of most test cases.

Details on function usage, including the field definitions of each structure, can be found in the code listing headers of section 3.24. This information can also be obtained through the built in Octave help facility, invoked by

```
help function name
```

at the interpreter command line. A complete listing of *postsim_lib* is included in 3.24. A summary of the functions follows below. The quantities *chip dump number*, *chain dump number*, and *chain number* are defined in section 2.2.

get_lag_frames : reads the lag frame file (lag_frame##.txt) generated by the test bench and places all data in an array of the structure, *frame*. The frames are unsorted, that is to say, as encountered during a sequential read of the file. Bad data are flagged for down stream processing.

make_dump_index: creates an index of lag frames pertaining to a given timestamp. This index is placed in an array of structures, *dump(id).index* where *id* is the *chip dump* number and *index* a vector of frame indexes associated with a particular timestamp.

add_CCchain_dump: concatenates lag data frames of a dump (common timestamp) according to flags within each frame. These chains are placed in a structure *CCchain* appended to the structure *dump*. Thus, data of a given chain is addressed by *dump(id).CCchain(ic)* where *id* is the chip dump number and *ic* is the chain index (not number) of the dump, *id*. Complex lag coefficients are scaled by **DVCOUNT** and **DATA_BIAS**.

get_mcb_data: reads the MCB register file (mcb_reg##.txt) and places the contents into structure *mcb*.

make_mcb_chains: creates an array of structure *Cchain* containing configuration data for each chain of the chip based on MCB register content. Configuration data for chain number *icn* is obtained from *Cchain(icn).*

get_bmodel_chains: reads behavioural model data corresponding to the chip configuration specified by *Cchain* and places these data in an array of structure *bmodel*. The behavioural model data for chain number *icn* is thus found in *bmodel(icn)*.

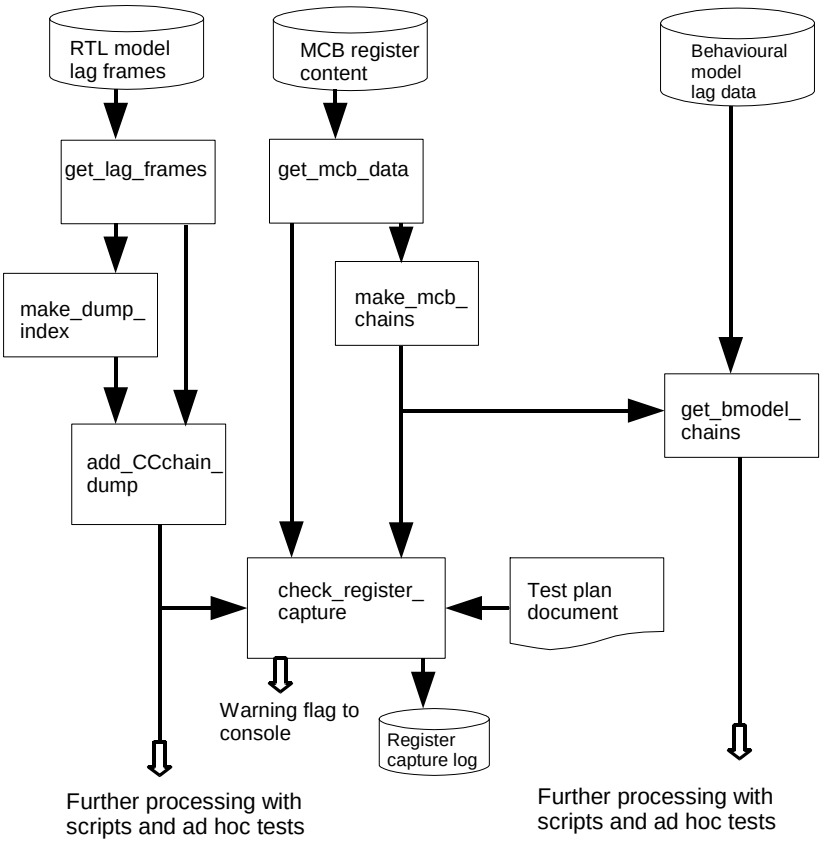

Illustration 3.1.1: Flow chart involving functions that form the basis of post simulation processing in most of the test cases. Ancillary scripts and functions may follow this process chain. All of the above functions with exception of check_register capture, are part of the library, postsim_lib.

The functions *get_bmodel_lags*, *get_bmodel_dumps*, *chain2single*, *lag2fd*, *scale_lag_data*, *single2chain* and *which_DVCOUNT* (of postsim_lib) are not called directly by the top level scripts although can be invoked at the interpreter command line during ad hoc tests. However, many are called from within the functions of the above summary and ancillary functions of 3.22.

The ancillary[7] functions listed in section 3.22; *make_stat_tables*, *model_diff_chains*, *write_fd_plots*, *write_lag_plots* and *check_timestamp* are not part of the library postsim_lib but are nevertheless called by a number of scripts. Scripts and functions that are specific to, and used in, the analysis of a particular test case are listed in corresponding subsections of this chapter.

7 ancillary may not be the most appropriate term as these functions are important but applied to a subset of the test cases that use the library postsim_lib.

The post simulation data of this report were processed with version 2.1.50 of Octave. Although many of the scripts and functions described here appear to work under more recent versions, a key function of the processing chain, *add_CCchain_dump*, does not appear to execute with versions much later than Octave 2.1.50. Since validation is essentially complete, there are no plans to further investigate this version incompatibility. However, the source code for this version of Octave will be included as part of the archive[8].

```
postsim_lib     version: 1.40   date: August 18, 2005

function name           version: date   calls
add_CCchain_dump        1.41: 05-06-26  scale_lag_data.m
chain2single            1.01: 04-10-18
get_bmodel_chains       1.01: 04-10-18  get_bmodel_lags.m
get_bmodel_lags         1.01: 04-10-18
get_lag_frames          1.30: 05-08-18
get_mcb_data            1.10: 04-11-01
lag2fd                  1.01: 04-10-18
make_dump_index         1.10: 04-10-27
make_mcb_chains         1.10: 04-11-02
scale_lag_data          1.10: 04-11-21  which_DVCOUNT.m
single2chain            1.01: 04-10-18
which_DVCOUNT           1.01: 04-10-18
```

Text 3.1.1: content of function library, postsim_lib; list created by the script list_versions within postsim_lib.

8 Octave is available under the GNU public licence and hence, source code (including legacy) is available from public repositories.

3.2 register consistency checking

Almost all test case scripts invoke the function *check_register_capture* which performs a number of consistency[9] checks between frame and register content. In addition, a template of expected register bit states and frame data, based on the test plan document [TVP-02] are supplied as an argument to *check_register_capture.* A warning is issued from *check_register_capture* if a bit exact comparison between the template or expected frame data and model fails. A summary of the checks made by this function now follow.

(1) **SyncER**, **AOV**, **OVR** bit states of the **MCSR** are consistent with those of the output frame status bits;

(2) output frame status bit states are consistent with those of the **XYSTATUS** register;

(3) **SyncER** is reset when **TVEN** is set;

(4) **TvER** of the **MCSR** is consistent with the states of **XYSTATUS** and **TVEN**;

(5) output frame **XYSyncerr** states are identical between cells of a chain;

(6) data bias is identical between dumps of the same overrun status;

(7) data bias is identical between cells of a chain and chains with common input streams;

(8) integration dead time is identical between all cells having the same overrun status;

(9) the source of a non-zero recirculation block corresponds to active dump control;

(10) captured XY stream identification data (**SID**, **SBID**, **BBID**) are identical between dumps, cells of a chain and chains with common data input streams;

(11) captured data from the active dump control lines (phase bin, recirculation block, dump command) are identical between dumps, cells of a chain and chains with common data input streams;

(12) captured data from the input streams and dump control agree with supplied values;

(13) **MCSR** and **XYSTAUS** register content agrees with the supplied template.

The function *check_register_capture* generates a log of a pass or failure of each condition checked. A warning or advisory is issued in the event of a check failure. A warning is issued when the failure is contrary to expectations and could indicate validation failure. An advisory indicates a check failure that, although unexpected, may occur under normal operations and not necessarily imply validation failure. A log that indicates a pass of all conditions is listed in Text 3.2.1. Logs containing no warnings or advisories were not listed in the previous chapter although the result of a call to *check_register_capture* was reported in the console log associated with each test case. The complete source listing of function *check_register_capture* is provided in 3.23.

The function *check_register_capture* generates a table of applied dump interval, accumulated bias (expressed in number of samples) and integration dead time (difference between applied dump and accumulated bias) for cells of both normal and overrun status. These tables are included within the reports of the previous chapter.

9 The term consistent as used in the context of this section, does not imply *identical* but rather that an *expected* relationship (as defined in documents [TVP-02] and [RFS-00]) holds between quantities.

```
Check OK: TvER bit within the MCSR and test vector enable (TVEN)
          are both reset.

Check OK: Accumulator overflow bit in the MCSR (AOV) is consistent
             with status bit settings in output frames.

Check OK: Dump overrun bit in the MCSR (OVR) is consistent
             with status bit settings in output frames.

Check OK: Sync. error bit in the MCSR (SyncER) is consistent
             with status bit settings in output frames.

Check OK: Synch. error status bit (X|YSyncer) settings in output
              frames are identical between cells of a chain.

Check OK: data bias is consistent between dumps.

Check OK: data bias is consistent over chains within a dump.

Check OK: integration dead time is consistent between dump streams;
          on cells reporting no overrun.

Check OK: non-zero recirc. block is on active dump control.

Check OK: Captured X and Y station, subband and baseband IDs
          (SID, SBID, BBID) are consistent between dumps and
          between cells of a chain.

Check OK: Captured X and Y Station ID (SID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station subband IDs (SBID) are consistent
          between chains with common input data streams.

Check OK: Captured X and Y Station baseband IDs (BBID) are consistent
          between chains with common input data streams.

Check OK: Captured X,Y ID data (SID, SBID, BBID) are set
          to expected values.

Check OK: Captured X,Y reciculation blocks, phase bins and dump
           command are consistent between dumps and between
           cells of chains.

Check OK: Captured X,Y recirculation blocks, phase bins and
          dump commands are consistent between chains with common
          input data streams.

Check OK: MCSR and status registers agree with the
                             template of expected states.
```

Text 3.2.1: content of a log generated by function check_register_capture with no warnings or advisories reported.

3.3 CMAM and correlation

3.3.1 tc00_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc00_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);
crc_ver_date= check_register_capture;
printf("Using check_register_capture: %s\n",crc_ver_date);

## if str2num(POSTSIM_LIB_VERSION) < 1.40
##             printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
## endif

## --------------------------------------------------------------------------------------
## --------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc32 and 33
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [0;1;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 85;
tc_SID_Y= 170;

applied_dump00= [660, 0,0,0,0,0,0,0];

## --------------------------------------------------------------------------------------
printf("\nProcess functional model simulator output for test case 00.\n");
[frame00, num_frames00, desc00, bad_frames, dontcare_frames]= get_lag_frames("lag_frames00.txt");
printf("%d frames read for test case 00.\n", num_frames00);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc00,"# ver: v");
printf("test case version: %s\n", desc00((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
              printf("Checksum is OK for all frames and corrupt data were not\n");
              printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb00 = get_mcb_data("mcb_reg00.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain00, config00, unchain]= make_mcb_chains(mcb00);


printf("\nInput configuration from MCSR regsister settings:\n");
if columns(config00) > 72
              printf("%s\n", config00(1:72));
              printf("%s\n", config00(73:end));
else
              printf("%s\n", config00);
endif
printf("\n");
```

```
## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump00, ndump00, ibadstamp] = make_dump_index(frame00);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump00, with_chains00] = add_CCchain_dump(dump00, frame00);

               if !(ndump00 == rows(with_chains00))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp00(1)= dump00(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp00(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump00,mcb00,Cchain00,applied_dump00,"tc00",\
               tc_SID_X,tc_SID_Y,base_SBID_X,base_SBID_Y,0,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file for test case 00.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file for test case 00.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file for test case 00.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb00.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

printf("\nData bias for test case 00 (split accumulator mode): %d\n",dump00(1).CCchain(1).data_bias);
if dump00(1).CCchain(1).data_bias == 5416192
               printf("Check OK: Accumultor value in test case 00 is as expected.\n\n");
else
               printf("WARNING: Accumultor value in test case 00 is NOT as expected.\n\n");
endif

printf("***--------------------------------------------------------------------------***\n");
```

3.3.2 tc32_33_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc32_33_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);
crc_ver_date= check_register_capture;
printf("Using check_register_capture: %s\n",crc_ver_date);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc32 and 33
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
```

```
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [0;1;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

applied_dump32= [10000, 0,0,0,0,0,0,0];
applied_dump33= [20000, 0,0,0,0,0,0,0];

## ----------------------------------------------------------------------------------
printf("\nProcess functional model simulator output for test case 32.\n");
[frame32, num_frames32, desc32, bad_frames, dontcare_frames]= get_lag_frames("lag_frames32.txt");
printf("%d frames read for test case 32.\n", num_frames32);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc32,"# ver: v");
printf("test case version: %s\n", desc32((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb32 = get_mcb_data("mcb_reg32.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain32, config32, unchain]= make_mcb_chains(mcb32);


printf("\nInput configuration from MCSR regsister settings:\n");
if columns(config32) > 72
               printf("%s\n", config32(1:72));
               printf("%s\n", config32(73:end));
else
               printf("%s\n", config32);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump32, ndump32, ibadstamp] = make_dump_index(frame32);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump32, with_chains32] = add_CCchain_dump(dump32, frame32);

               if !(ndump32 == rows(with_chains32))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp32(1)= dump32(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp32(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump32,mcb32,Cchain32,applied_dump32,"tc32",\
               tc_SID_X,tc_SID_Y,base_SBID_X,base_SBID_Y,32,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file for test case 32.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
```

```
                                                checks;\n\
                                        details in log file for test case 32.\n");
                        endif
                        if Ladvis
                                        printf("\nADVISORIES encountered during register status and data capture
                                                checks;\n\
                                        details in log file for test case 32.\n");
                        endif
        endif

        # Check that DESSR bits are all set as expected.
        if all(mcb32.DESSR == test_DESSR)
                        printf("\nCheck OK: DESSR bits are set as expected.\n");
        else
                        printf("\nWARNING: DESSR bits are set NOT as expected.\n");
        endif
endif

printf("\nProcess functional model simulator output for test case 33.\n");
[frame33, num_frames33, desc33, bad_frames, dontcare_frames]= get_lag_frames("lag_frames33.txt");
printf("%d frames read for test case 33.\n", num_frames33);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc33,"# ver: v");
printf("test case version: %s\n", desc33((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
        printf("Checksum is OK for all frames and corrupt data were not\n");
        printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb33 = get_mcb_data("mcb_reg33.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain33, config33, unchain]= make_mcb_chains(mcb33);


printf("\nInput configuration from MCSR regsister settings:\n");
if columns(config33) > 72
        printf("%s\n", config33(1:72));
        printf("%s\n", config33(73:end));
else
        printf("%s\n", config33);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump33, ndump33, ibadstamp] = make_dump_index(frame33);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

        # form contatenated chains for each dump on the basis of the output frame
        # status bits and add these to the dump structure.
        [dump33, with_chains33] = add_CCchain_dump(dump33, frame33);

        if !(ndump33 == rows(with_chains33))
                        printf("\nADVISORY: NOT ALL dumps contain chains!\n");
        endif

        timestamp33(1)= dump33(1).TIMESTAMP_0;
        printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp33(1));

        # check consistency of STATUS and MCSR register states; data_bias and data captured
        # from data and dump control streams. Check consistency of mcb contents with those
        # of the structure test_mcb.
        [Lwarn, Ladvis] = check_register_capture(dump33,mcb33,Cchain33,applied_dump33,"tc33",\
        tc_SID_X,tc_SID_Y,base_SBID_X,base_SBID_Y,33,test_mcb);

        if !Lwarn & !Ladvis
                        printf("\nCheck OK: no inconsistencies reported in captured data\n");
                        printf("          or register contents; no unexpected register reports;\n");
                        printf("          details in log file for test case 33.\n");
        else
                        if Lwarn
                                        printf("\nWARNINGS encountered during register status and data capture
                                                checks;\n\
                                        details in log file for test case 33.\n");
                        endif
                        if Ladvis
                                        printf("\nADVISORIES encountered during register status and data capture
                                                checks;\n\
                                        details in log file for test case 33.\n");
                        endif
        endif
```

```
                # Check that DESSR bits are all set as expected.
                if all(mcb33.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

printf("\n Read lag data for tc32 and tc33 created with the test vector data set\n\
DS14 (used for bit exact tests. Use the same mcb_reg files as for DS4.\n");

[frame32_DS14, num_frames32_DS14, desc32_DS14, bad_frames, dontcare_frames]=\
get_lag_frames("lag_frames32_DS14.txt");
printf("%d frames read for test case 32_DS14.\n", num_frames32_DS14);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc32_DS14,"# ver: v");
printf("test case version: %s\n", desc32_DS14((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
                printf("Checksum is OK for all frames and corrupt data were not\n");
                printf("detected (no 'x' or 'X' hex digits were found).\n");
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump32_DS14, ndump32_DS14, ibadstamp] = make_dump_index(frame32_DS14);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump32_DS14, with_chains32_DS14] = add_CCchain_dump(dump32_DS14, frame32_DS14);

                if !(ndump32_DS14 == rows(with_chains32_DS14))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp32_DS14(1)= dump32_DS14(1).TIMESTAMP_0;
                printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp32_DS14(1));

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump32_DS14,mcb32,Cchain32,applied_dump32,"tc32_DS14",\
                tc_SID_X,tc_SID_Y,base_SBID_X,base_SBID_Y,32,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file for test case 32_DS14.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file for test case 32_DS14.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file for test case 32_DS14.\n");
                                endif
                endif

endif

printf("\n");
[frame33_DS14, num_frames33_DS14, desc33_DS14, bad_frames, dontcare_frames]=\
get_lag_frames("lag_frames33_DS14.txt");
printf("%d frames read for test case 33_DS14.\n", num_frames33_DS14);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc33_DS14,"# ver: v");
printf("test case version: %s\n", desc33_DS14((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
                printf("Checksum is OK for all frames and corrupt data were not\n");
                printf("detected (no 'x' or 'X' hex digits were found).\n");
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump33_DS14, ndump33_DS14, ibadstamp] = make_dump_index(frame33_DS14);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps
```

```
                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump33_DS14, with_chains33_DS14] = add_CCchain_dump(dump33_DS14, frame33_DS14);

                if !(ndump33_DS14 == rows(with_chains33_DS14))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp33_DS14(1)= dump33_DS14(1).TIMESTAMP_0;
                printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp33_DS14(1));

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump33_DS14,mcb33,Cchain33,applied_dump33,"tc33_DS14",\
                tc_SID_X,tc_SID_Y,base_SBID_X,base_SBID_Y,33,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file for test case 33_DS14.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file for test case 33_DS14.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file for test case 33_DS14.\n");
                                endif
                endif

endif

printf("***--------------------------------------------------------------------------***\n");
```

3.3.3 make_lag_tables_tc32_33

```
# get behavioural model data
[blags_DS4, nblags_DS4] = get_bmodel_lags("CC0x0_test32_DS4_2048lagsD0.dat");        # vector set DS4
[blags_DS14, nblags_DS14] = get_bmodel_lags("CC0x0_test32_DS14_2048lagsD0.dat");     # vector set DS14

# find the difference between the functional and behavioral model lag data
# for the case where the behavioural model is based on data set DS4
fb32_DS4= dump32(1).CCchain(1).clag - blags_DS4;
fb33_DS4= dump33(1).CCchain(1).clag - blags_DS4;

# find the difference between the functional and behavioral model lag data
# for the case where the behavioural model is based on data set DS14
fb32_DS14= dump32_DS14(1).CCchain(1).clag - blags_DS14;
fb33_DS14= dump33_DS14(1).CCchain(1).clag - blags_DS14;

printf("\nWriting tables.\n");
CS = "|";  # table column delimiter, 1 row per line

tablefile= fopen("fb32_33_diff.txt","w");

fprintf(tablefile, " test case %s data set %s dvcount %s range_re %s range_im %s mean_re %s\
mean_im %s std_re %s std_im\n", CS,CS,CS,CS,CS,CS,CS,CS);

fprintf(tablefile, " %d %s %s %s %d %s %2.4f %s %2.4f %s %6.2e %s %6.2e %s %2.4f %s %2.4f\n",\
32,CS,"DS4",CS,dump32(1).CCchain.dvcount,CS,range(real(fb32_DS4)),CS,range(imag(fb32_DS4)),\
CS,mean(real(fb32_DS4)),CS,mean(imag(fb32_DS4)),CS,std(real(fb32_DS4)),CS, std(imag(fb32_DS4)));

fprintf(tablefile, " %d %s %s %s %d %s %2.4f %s %2.4f %s %6.2e %s %6.2e %s %2.4f %s %2.4f\n",\
33,CS,"DS4",CS,dump33(1).CCchain.dvcount,CS,range(real(fb33_DS4)),CS,range(imag(fb33_DS4)),\
CS,mean(real(fb33_DS4)),CS,mean(imag(fb33_DS4)),CS,std(real(fb33_DS4)),CS, std(imag(fb33_DS4)));

fprintf(tablefile, " %d %s %s %s %d %s %2.4f %s %2.4f %s %6.2e %s %6.2e %s %2.4f %s %2.4f\n",\
32,CS,"DS14",CS,dump32_DS14(1).CCchain.dvcount,CS,range(real(fb32_DS14)),CS,range(imag(fb32_DS14)),\
CS,mean(real(fb32_DS14)),CS,mean(imag(fb32_DS14)),CS,std(real(fb32_DS14)),CS, std(imag(fb32_DS14)));

fprintf(tablefile, " %d %s %s %s %d %s %2.4f %s %2.4f %s %6.2e %s %6.2e %s %2.4f %s %2.4f\n",\
33,CS,"DS14",CS,dump33_DS14(1).CCchain.dvcount,CS,range(real(fb33_DS14)),CS,range(imag(fb33_DS14)),\
CS,mean(real(fb33_DS14)),CS,mean(imag(fb33_DS14)),CS,std(real(fb33_DS14)),CS, std(imag(fb33_DS14)));

fclose(tablefile);
```

3.3.4 make_fd_tables_tc32_33

```
# get frequency domain behavioural model data
b_fd_DS4= lag2fd(blags_DS4,1,1);       # vector set DS4
b_fd_DS14= lag2fd(blags_DS14,1,1);     # vector set DS14

# find the difference between the functional and behavioral model freq. domain data
# for the case where the behavioural model is based on data set DS4
fb32_fd= lag2fd(dump32(1).CCchain(1).clag,1,1) - b_fd_DS4;
fb33_fd= lag2fd(dump33(1).CCchain(1).clag,1,1) - b_fd_DS4;

# find the difference between the functional and behavioral model freq. domain data
# for the case where the behavioural model is based on data set DS14
fb32_fd_DS14= lag2fd(dump32_DS14(1).CCchain(1).clag,1,1) - b_fd_DS14;
fb33_fd_DS14= lag2fd(dump33_DS14(1).CCchain(1).clag,1,1) - b_fd_DS14;

printf("\nWriting freq. domain difference tables.\n");
CS = "|";  # table column delimiter, 1 row per line

tablefile= fopen("fb32_33_fd_diff.txt","w");

fprintf(tablefile, " test case %s data set %s range_re %s range_im %s mean_re %s\
mean_im %s std_re %s std_im\n",CS,CS,CS,CS,CS,CS,CS);

fprintf(tablefile, " %d %s %s %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e\n",\
32,CS,"DS4",CS,range(real(fb32_fd)),CS,range(imag(fb32_fd)),\
CS,mean(real(fb32_fd)),CS,mean(imag(fb32_fd)),CS,std(real(fb32_fd)),CS, std(imag(fb32_fd)));

fprintf(tablefile, " %d %s %s %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e\n",\
33,CS,"DS4",CS,range(real(fb33_fd)),CS,range(imag(fb33_fd)),\
CS,mean(real(fb33_fd)),CS,mean(imag(fb33_fd)),CS,std(real(fb33_fd)),CS, std(imag(fb33_fd)));

fprintf(tablefile, " %d %s %s %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e\n",\
32,CS,"DS14",CS,range(real(fb32_fd_DS14)),CS,range(imag(fb32_fd_DS14)),\
CS,mean(real(fb32_fd_DS14)),CS,mean(imag(fb32_fd_DS14)),CS,std(real(fb32_fd_DS14)),CS, std(imag(fb32_fd_DS14)));

fprintf(tablefile, " %d %s %s %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e %s %6.2e\n",\
33,CS,"DS14",CS,range(real(fb33_fd_DS14)),CS,range(imag(fb33_fd_DS14)),\
CS,mean(real(fb33_fd_DS14)),CS,mean(imag(fb33_fd_DS14)),CS,std(real(fb33_fd_DS14)),CS, std(imag(fb33_fd_DS14)));

fclose(tablefile);
```

3.3.5 validate_bit_exact

```
# validate correlation at the bit exact level

# read data provided by the behavioural model using test vectors saved by the test
# bench for bit exact tests (input data set DS14 to the test bench)
[blags32_DS14s, nblags_DS14s] = get_bmodel_lags("CC0x0_test32_DS14-S_2048lagsD0.dat");
[blags33_DS14s, nblags_DS14s] = get_bmodel_lags("CC0x0_test33_DS14-S_2048lagsD0.dat");

# find the difference between the functional and behavioral model lag data
fb32_DS14s= dump32_DS14(1).CCchain(1).clag - blags32_DS14s;
fb33_DS14s= dump33_DS14(1).CCchain(1).clag - blags33_DS14s;

printf("\nFinding the difference between functional and behavioural model lags\n\
formed using bit exact test vectors.\n");

# find indices of non zero differences between functional and behavioural model data of
#  test case 32 to the level of floating point precision of the machine
ndiff32_im= find(abs(imag(fb32_DS14s)) >= 10*eps);
ndiff32_re= find(abs(real(fb32_DS14s)) >= 10*eps);
if isempty(ndiff32_re) & isempty(ndiff32_im)
              printf("\nNo differences exceeding the floating point precision limit of this\n\
machine were found beween models; behavioural and functional models for test\n\
case 32 agree on a bit exact basis.\n");
else
              printf("\nBehavioural and functional models for test case 32 DO NOT\n\
               agree on a bit exact basis.\n");
endif

# find indices of non zero differences between functional and behavioural model data of
#  test case 33 to the level of floating point precision of the machine
ndiff33_im= find(abs(imag(fb33_DS14s)) >= 10*eps);
ndiff33_re= find(abs(real(fb33_DS14s)) >= 10*eps);
if isempty(ndiff33_re) & isempty(ndiff33_im)
              printf("\nNo differences exceeding the floating point precision limit of this\n\
machine were found beween models; behavioural and functional models for test\n\
case 33 agree on a bit exact basis.\n");
else
              printf("\nBehavioural and functional models for test case 33 DO NOT\n\
               agree on a bit exact basis.\n");
endif
```

3.4 accumulator overflow

3.4.1 tc01_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc01_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);
crc_ver_date= check_register_capture;
printf("Using check_register_capture: %s\n",crc_ver_date);

## if str2num(POSTSIM_LIB_VERSION) < 1.40
##             printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
## endif

## --------------------------------------------------------------------------------------
## --------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc32 and 33
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 1;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
```

```
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [0;1;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 85;
tc_SID_Y= 170;

tc= 1;
applied_dump01= [1044, 0,0,0,0,0,0,0];

## ----------------------------------------------------------------------------------
printf("\nProcess functional model simulator output for test case 01.\n");
[frame01, num_frames01, desc01, bad_frames, dontcare_frames]= get_lag_frames("lag_frames01.txt");
printf("%d frames read for test case 01.\n", num_frames01);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc01,"# ver: v");
printf("test case version: %s\n", desc01((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb01 = get_mcb_data("mcb_reg01.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain01, config01, unchain]= make_mcb_chains(mcb01);


printf("\nInput configuration from MCSR regsister settings:\n");
if columns(config01) > 72
               printf("%s\n", config01(1:72));
               printf("%s\n", config01(73:end));
else
               printf("%s\n", config01);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump01, ndump01, ibadstamp] = make_dump_index(frame01);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump01, with_chains01] = add_CCchain_dump(dump01, frame01);

               if !(ndump01 == rows(with_chains01))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp01(1)= dump01(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp01(1));


               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump01,mcb01,Cchain01,applied_dump01,"tc01",\
               tc_SID_X,tc_SID_Y,base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file for test case 01.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file for test case 01.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file for test case 01.\n");
```

```
                        endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb01.DESSR == test_DESSR)
                        printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                        printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

printf("\nData bias for test case 01 (split accumulator mode): %d\n",dump01(1).CCchain(1).data_bias);
if dump01(1).CCchain(1).data_bias == 132352
                printf("Check OK: Accumultor value in test case 01 is as expected.\n\n");
else
                printf("WARNING: Accumultor value in test case 01 is NOT as expected.\n\n");
endif

printf("***--------------------------------------------------------------------------***\n");
```

3.5 normal dump and readout

3.5.1 tc02_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc02_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
                printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ------------------------------------------------------------------------------------
## ------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc02_jun14-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [0;1;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 2;
applied_dump= [12000, 0, 0, 0, 0, 0, 0, 0];

## ------------------------------------------------------------------------------------
[frame02,num_frames,desc02,bad_frames,dontcare_frames]= get_lag_frames("lag_frames02.txt");
printf("%d frames read for test case 02.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc02,"# ver: v");
```

```
printf("test case version: %s\n", desc02((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb02 = get_mcb_data("mcb_reg02.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain02, config02, unchain]= make_mcb_chains(mcb02);

## print chip configuration
nchains02= columns(Cchain02);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains02);
for ic=1:4:nchains02
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain02(ic).Xinput,Cchain02(ic).CCC(1),\
               Cchain02(ic).CCC(end),Cchain02(ic).Yinput,\
               Cchain02(ic+1).Xinput,Cchain02(ic+1).CCC(1),\
               Cchain02(ic+1).CCC(end),Cchain02(ic+1).Yinput,\
               Cchain02(ic+2).Xinput,Cchain02(ic+2).CCC(1),\
               Cchain02(ic+2).CCC(end),Cchain02(ic+2).Yinput,\
               Cchain02(ic+3).Xinput,Cchain02(ic+3).CCC(1),\
               Cchain02(ic+3).CCC(end),Cchain02(ic+3).Yinput);
endfor

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump02, ndump02, ibadstamp] = make_dump_index(frame02);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump02, with_chains02] = add_CCchain_dump(dump02, frame02);

               if !(ndump02 == rows(with_chains02))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp02=zeros(ndump02,1);
               for id=1:ndump02; timestamp02(id)= dump02(id).TIMESTAMP_0; endfor;
               printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp02(1));
               if all(diff(timestamp02) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump02,mcb02,Cchain02,applied_dump,"tc02",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb02.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod02] = get_bmodel_chains("test02", Cchain02, 10);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
```

```
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc02";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);

dmult= [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain02,dump02,bmod02,tname,1,1,dmult,\
dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.2 tc02_chain_similarity_check

```
Lnz= !ones(16,ndump02);  # flag any non-zero difference
for id=1:ndump02
               for ic=1:15
                              Lnz(ic,id)= any(abs(dump02(id).CCchain(ic).clag-\
                                             dump02(id).CCchain(ic+1).clag) >= 10*eps);
               endfor
endfor

if !any(any(Lnz))
               printf("\nNo differences exceeding the floating point precision limit of this\n\
machine were found between chains for any of the %d dumps; all chains generated\n\
identical lag data.\n\n", ndump02);
else
               printf("\nWARNING: Differences exceeding the floating point precision limit\n\
                 of this machine were found between chains.\n\n");
endif
```

3.5.3 tc03_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc03_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc03_jun14-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 1;
test_mcb.Y_DUMP_SYNC= 1;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;0;1;0;1;0;1];
```

```
test_mcb.Y_SE_CLK= [0;1;0;1;0;1;0;1];

test_DESSR= [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc=3;
applied_dump= [12000, 0, 24000, 0, 48000, 0, 96000, 0];

## ----------------------------------------------------------------------------------
[frame03,num_frames,desc03,bad_frames,dontcare_frames]= get_lag_frames("lag_frames03.txt");
printf("%d frames read for test case 03.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc03,"# ver: v");
printf("test case version: %s\n", desc03((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data was not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb03 = get_mcb_data("mcb_reg03.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain03, config03, unchain]= make_mcb_chains(mcb03);


## print chip configuration
nchains03= columns(Cchain03);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains03);
for ic=1:4:nchains03
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain03(ic).Xinput,Cchain03(ic).CCC(1),\
               Cchain03(ic).CCC(end),Cchain03(ic).Yinput,\
               Cchain03(ic+1).Xinput,Cchain03(ic+1).CCC(1),\
               Cchain03(ic+1).CCC(end),Cchain03(ic+1).Yinput,\
               Cchain03(ic+2).Xinput,Cchain03(ic+2).CCC(1),\
               Cchain03(ic+2).CCC(end),Cchain03(ic+2).Yinput,\
               Cchain03(ic+3).Xinput,Cchain03(ic+3).CCC(1),\
               Cchain03(ic+3).CCC(end),Cchain03(ic+3).Yinput);
endfor


## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump03, ndump03, ibadstamp] = make_dump_index(frame03);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump03, with_chains03] = add_CCchain_dump(dump03, frame03);

               if !(ndump03 == rows(with_chains03))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp03=zeros(ndump03,1);
               for id=1:ndump03; timestamp03(id)= dump03(id).TIMESTAMP_0; endfor;
               printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp03(1));
               if all(diff(timestamp03) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump03,mcb03,Cchain03,applied_dump,"tc03",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
```

```
                        endif
                        if Ladvis
                                printf("\nADVISORIES encountered during register status and data capture
                                        checks;\n\
                                details in log file.\n");
                        endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb03.DESSR == test_DESSR)
                        printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                        printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

[bmod03] = get_bmodel_chains("test03", Cchain03, 16);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc03";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);

dmult= [1;2;4;8]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain03,dump03,bmod03,tname,1,4,dmult,dmod,lag_peak_thold,\
fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.4 tc04_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc04_postsim.m : V2.01";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
                printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc04_apr21-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [1;1;1;1;1;1;1;1];
test_mcb.Y_DATA= [1;1;1;1;1;1;1;1];
test_mcb.X_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.Y_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [1;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [1;1;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset
```

```
base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 4;
applied_dump= [0, 20000, 0, 0, 0, 0, 0, 0];

## ----------------------------------------------------------------------------------
[frame04,num_frames,desc04,bad_frames,dontcare_frames]= get_lag_frames("lag_frames04.txt");
printf("%d frames read for test case 04.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc04,"# ver: v");
printf("test case version: %s\n", desc04((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb04 = get_mcb_data("mcb_reg04.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain04, config04, unchain]= make_mcb_chains(mcb04);

## print chip configuration
nchains04= columns(Cchain04);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains04);
for ic=1:4:nchains04
               printf("%d(%d:%d)%d\n",\
               Cchain04(ic).Xinput,Cchain04(ic).CCC(1),\
               Cchain04(ic).CCC(end),Cchain04(ic).Yinput);
endfor

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump04, ndump04, ibadstamp] = make_dump_index(frame04);

id=1;
if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump04, with_chains04] = add_CCchain_dump(dump04, frame04);

               if !(ndump04 == rows(with_chains04))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp04(id)= dump04(id).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp04(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump04,mcb04,Cchain04,applied_dump,"tc04",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb04.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod04] = get_bmodel_chains("test04", Cchain04, 1);
```

```
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc04";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);

dmult= [1.6667]; # (number of samples per dump) / 12000
dmod=  [2]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain04,dump04,bmod04,tname,1,1,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.5 tc04_inspect_status_captured

```
# print dumps of captured data that differ between cells of a chain
id=1;
ic=1;

printf("XSyncerr: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",dump04(id).CCchain(ic).XSyncerr');
printf("SID_X: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",dump04(id).CCchain(ic).SID_X');
printf("SBID_X: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",dump04(id).CCchain(ic).SBID_X);
printf("BBID_X: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n\n",dump04(id).CCchain(ic).BBID_X);

printf("YSyncerr: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",dump04(id).CCchain(ic).YSyncerr');
printf("SID_Y: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",dump04(id).CCchain(ic).SID_Y');
printf("SBID_Y: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",dump04(id).CCchain(ic).SBID_Y);
printf("BBID_Y: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",dump04(id).CCchain(ic).BBID_Y);
```

3.5.6 tc05_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc05_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
              printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc05_jun14-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 1;
test_mcb.Y_DUMP_SYNC= 1;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [1;0;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [1;0;1;1;1;1;1;1];

test_DESSR= [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # expected to be all reset
```

```
base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 5;
applied_dump= [0, 12000, 0, 0, 0, 0, 0, 0];

## ------------------------------------------------------------------------------------
[frame05,num_frames,desc05,bad_frames,dontcare_frames]= get_lag_frames("lag_frames05.txt");
printf("%d frames read for test case 05.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc05,"# ver: v");
printf("test case version: %s\n", desc05((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb05 = get_mcb_data("mcb_reg05.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain05, config05, unchain]= make_mcb_chains(mcb05);


## print chip configuration
nchains05= columns(Cchain05);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains05);
for ic=1:4:nchains05
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain05(ic).Xinput,Cchain05(ic).CCC(1),\
               Cchain05(ic).CCC(end),Cchain05(ic).Yinput,\
               Cchain05(ic+1).Xinput,Cchain05(ic+1).CCC(1),\
               Cchain05(ic+1).CCC(end),Cchain05(ic+1).Yinput,\
               Cchain05(ic+2).Xinput,Cchain05(ic+2).CCC(1),\
               Cchain05(ic+2).CCC(end),Cchain05(ic+2).Yinput,\
               Cchain05(ic+3).Xinput,Cchain05(ic+3).CCC(1),\
               Cchain05(ic+3).CCC(end),Cchain05(ic+3).Yinput);
endfor
printf("\n");
## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump05, ndump05, ibadstamp] = make_dump_index(frame05);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump05, with_chains05] = add_CCchain_dump(dump05, frame05);

               if !(ndump05 == rows(with_chains05))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp05=zeros(ndump05,1);
               for id=1:ndump05; timestamp05(id)= dump05(id).TIMESTAMP_0; endfor;
               if timestamp05(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp05(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp05(1));
               endif
               if all(diff(timestamp05) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump05,mcb05,Cchain05,applied_dump,"tc05",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
```

```
                        if Ladvis
                                printf("\nADVISORIES encountered during register status and data capture
                                        checks;\n\
                                details in log file.\n");
                        endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb05.DESSR == test_DESSR)
                        printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                        printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

[bmod05] = get_bmodel_chains("test05", Cchain05, 10);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc05";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);

dmult= [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain05,dump05,bmod05,tname,1,1,dmult,\
dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.7 tc05_chain_similarity_check

```
Lnz= !ones(16,ndump05);  # flag any non-zero difference
for id=1:ndump05
                for ic=1:15
                        Lnz(ic,id)= any(abs(dump05(id).CCchain(ic).clag-\
                                dump05(id).CCchain(ic+1).clag) >= 10*eps);
                endfor
endfor

if !any(any(Lnz))
                printf("\nNo differences exceeding the floating point precision limit of this\n\
machine were found between chains for any of the %d dumps; all chains generated\n\
identical lag data.\n\n", ndump05);
else
                printf("\nWARNING: Differences exceeding the floating point precision limit\n\
                  of this machine were found between chains.\n\n");
endif
```

3.5.8 tc06_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc06_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
                printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc06_jul24-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
```

```
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [1;1;1;1;1;1;1;1];
test_mcb.Y_DATA= [1;1;1;1;1;1;1;1];
test_mcb.X_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.Y_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [1;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [1;1;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 6;
applied_dump= [0, 12000, 0, 24000, 0, 48000, 0, 96000];

## ----------------------------------------------------------------------------------
[frame06,num_frames,desc06,bad_frames,dontcare_frames]= get_lag_frames("lag_frames06.txt");
printf("%d frames read for test case 06.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc06,"# ver: v");
printf("test case version: %s\n", desc06((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb06 = get_mcb_data("mcb_reg06.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain06, config06, unchain]= make_mcb_chains(mcb06);


## print chip configuration
nchains06= columns(Cchain06);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains06);
for ic=1:4:nchains06
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain06(ic).Xinput,Cchain06(ic).CCC(1),\
               Cchain06(ic).CCC(end),Cchain06(ic).Yinput,\
               Cchain06(ic+1).Xinput,Cchain06(ic+1).CCC(1),\
               Cchain06(ic+1).CCC(end),Cchain06(ic+1).Yinput,\
               Cchain06(ic+2).Xinput,Cchain06(ic+2).CCC(1),\
               Cchain06(ic+2).CCC(end),Cchain06(ic+2).Yinput,\
               Cchain06(ic+3).Xinput,Cchain06(ic+3).CCC(1),\
               Cchain06(ic+3).CCC(end),Cchain06(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump06, ndump06, ibadstamp] = make_dump_index(frame06);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump06, with_chains06] = add_CCchain_dump(dump06, frame06);

               if !(ndump06 == rows(with_chains06))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp06=zeros(ndump06,1);
               for id=1:ndump06; timestamp06(id)= dump06(id).TIMESTAMP_0; endfor;
               if timestamp06(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp06(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp06(1));
               endif
```

```
                if all(diff(timestamp06) == 1)
                                printf("Timestamps are sequential and increment by 1.\n");
                else
                                printf("Missing timestamps (not increment by 1).\n");
                endif

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump06,mcb06,Cchain06,applied_dump,"tc06",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb06.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

[bmod06] = get_bmodel_chains("test06", Cchain06, 16);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc06";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1;2;4;8]; # (number of samples per dump) / 12000
dmod=  [2;2;2;2]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain06,dump06,bmod06,tname,1,4,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.9 tc06_inspect_status_captured

```
# print dumps of captured data that differ between cells of a chain
printf("\nchain|dump| XSyncerr | SID_X | SBID_X | BBID_X |\n");
ic=1;
for i=1:16
                if !(all(dump06(i).CCchain(ic).XSyncerr(1) == dump06(i).CCchain(ic).XSyncerr(:)) &&\
                all(dump06(i).CCchain(ic).SID_X(1) == dump06(i).CCchain(ic).SID_X(:)) &&\
                all(dump06(i).CCchain(ic).BBID_X(1) == dump06(i).CCchain(ic).BBID_X(:)) &&\
                all(dump06(i).CCchain(ic).SBID_X(1) == dump06(i).CCchain(ic).SBID_X(:)))
                                printf("%d | %d | %d %d %d %d  | %d %d %d %d  | %d %d %d %d | %d %d %d %d |\n",\
                                ic,i,dump06(i).CCchain(ic).XSyncerr',dump06(i).CCchain(ic).SID_X',\
                                dump06(i).CCchain(ic).SBID_X', dump06(i).CCchain(ic).BBID_X');
                endif
endfor

ic=3;
for i=4:4:16
                if !(all(dump06(i).CCchain(ic).XSyncerr(1) == dump06(i).CCchain(ic).XSyncerr(:)) &&\
                all(dump06(i).CCchain(ic).SID_X(1) == dump06(i).CCchain(ic).SID_X(:)) &&\
                all(dump06(i).CCchain(ic).BBID_X(1) == dump06(i).CCchain(ic).BBID_X(:)) &&\
                all(dump06(i).CCchain(ic).SBID_X(1) == dump06(i).CCchain(ic).SBID_X(:)))
                                printf("%d | %d | %d %d %d %d  | %d %d %d %d  | %d %d %d %d | %d %d %d %d |\n",\
```

```
                        ic,i,dump06(i).CCchain(ic).XSyncerr',dump06(i).CCchain(ic).SID_X',\
                        dump06(i).CCchain(ic).SBID_X', dump06(i).CCchain(ic).BBID_X');
            endif
endfor
```

3.5.10 tc08_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc08_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
            printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc08_apr21-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc=8;
applied_dump= [40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000];

## ----------------------------------------------------------------------------------------
[frame08,num_frames,desc08,bad_frames,dontcare_frames]= get_lag_frames("lag_frames08.txt");
printf("%d frames read for test case 08.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc08,"# ver: v");
printf("test case version: %s\n", desc08((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
            printf("Checksum is OK for all frames and corrupt data were not\n");
            printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb08 = get_mcb_data("mcb_reg08.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain08, config08, unchain]= make_mcb_chains(mcb08);

## print chip configuration
nchains08= columns(Cchain08);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains08);
for ic=1:4:nchains08
            printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
            Cchain08(ic).Xinput,Cchain08(ic).CCC(1),\
            Cchain08(ic).CCC(end),Cchain08(ic).Yinput,\
            Cchain08(ic+1).Xinput,Cchain08(ic+1).CCC(1),\
```

```
                Cchain08(ic+1).CCC(end),Cchain08(ic+1).Yinput,\
                Cchain08(ic+2).Xinput,Cchain08(ic+2).CCC(1),\
                Cchain08(ic+2).CCC(end),Cchain08(ic+2).Yinput,\
                Cchain08(ic+3).Xinput,Cchain08(ic+3).CCC(1),\
                Cchain08(ic+3).CCC(end),Cchain08(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump08, ndump08, ibadstamp] = make_dump_index(frame08);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump08, with_chains08] = add_CCchain_dump(dump08, frame08);
                if !(ndump08 == rows(with_chains08))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp08=zeros(ndump08,1);
                for id=1:ndump08; timestamp08(id)= dump08(id).TIMESTAMP_0; endfor;
                if timestamp08(1) < 0
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp08(1)+2^32);
                else
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp08(1));
                endif
                if all(diff(timestamp08) == 1)
                                printf("Timestamps are sequential and increment by 1.\n");
                else
                                printf("Missing timestamps (not increment by 1).\n");
                endif

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump08,mcb08,Cchain08,applied_dump,"tc08",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb08.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

[bmod08] = get_bmodel_chains("test08", Cchain08, ndump08);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc08";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [3.3333;3.3333;3.3333;3.3333;3.3333;3.3333;3.3333;3.3333]; # (number of samples per dump) / 12000
dmod=  [2;3;4;5;6;7;8;9]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain08,dump08,bmod08,tname,1,8,\
dmult,dmod,lag_peak_thold,fd_peak_thold,1);

disp("List all cells represented in the output frame data for the first dump.");
```

```
disp(dump08(1).CCchain(:).CCC);
printf("***------------------------------------------------------------------------***\n");
```

3.5.11 tc09_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc09_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc09_apr21-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 9;
applied_dump= [40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000];

## ----------------------------------------------------------------------------------------
[frame09,num_frames,desc09,bad_frames,dontcare_frames]= get_lag_frames("lag_frames09.txt");
printf("%d frames read for test case 09.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc09,"# ver: v");
printf("test case version: %s\n", desc09((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data was not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb09 = get_mcb_data("mcb_reg09.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain09, config09, unchain]= make_mcb_chains(mcb09);


## print chip configuration
nchains09= columns(Cchain09);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains09);
for ic=1:4:nchains09
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain09(ic).Xinput,Cchain09(ic).CCC(1),\
               Cchain09(ic).CCC(end),Cchain09(ic).Yinput,\
               Cchain09(ic+1).Xinput,Cchain09(ic+1).CCC(1),\
               Cchain09(ic+1).CCC(end),Cchain09(ic+1).Yinput,\
               Cchain09(ic+2).Xinput,Cchain09(ic+2).CCC(1),\
               Cchain09(ic+2).CCC(end),Cchain09(ic+2).Yinput,\
               Cchain09(ic+3).Xinput,Cchain09(ic+3).CCC(1),\
               Cchain09(ic+3).CCC(end),Cchain09(ic+3).Yinput);
endfor
```

```
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump09, ndump09, ibadstamp] = make_dump_index(frame09);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump09, with_chains09] = add_CCchain_dump(dump09, frame09);

               if !(ndump09 == rows(with_chains09))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp09=zeros(ndump09,1);
               for id=1:ndump09; timestamp09(id)= dump09(id).TIMESTAMP_0; endfor;
               if timestamp09(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp09(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp09(1));
               endif
               if all(diff(timestamp09) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump09,mcb09,Cchain09,applied_dump,"tc09",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb09.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod09] = get_bmodel_chains("test08", Cchain09, ndump09); # bmodel for tc08 and tc09 are identical
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc09";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [3.3333;3.3333;3.3333;3.3333;3.3333;3.3333;3.3333;3.3333]; # (number of samples per dump) / 12000
dmod=  [2;3;4;5;6;7;8;9]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain09,dump09,bmod09,tname,1,8,\
dmult,dmod,lag_peak_thold,fd_peak_thold,1);

disp("List all cells represented in the output frame data for the first dump.");
disp(dump09(1).CCchain(:).CCC);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.12 tc08_09_similarity_check

```
Lnz= !ones(8,ndump08);  # flag any non-zero difference
for id=1:ndump08
               for ic=1:8
                              Lnz(ic,id)= any(abs(dump08(id).CCchain(ic).clag-\
                                             dump09(id).CCchain(ic).clag) >= 10*eps);
               endfor
endfor

if !any(any(Lnz))
               printf("\nNo differences exceeding the floating point precision limit of this\n\
machine were found between corresponding chains and dumps of test\n\
cases 08 and 09.\n\n");
else
               printf("\nADVISORY: Differences exceeding the floating point precision limit\n\
                 of this machine were found between lag data of test cases 08 and 09.\n\n");
endif
```

3.5.13 tc10_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc10_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc10_jul24-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 10;
applied_dump= [40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000];

## ----------------------------------------------------------------------------------------
[frame10,num_frames,desc10,bad_frames,dontcare_frames]= get_lag_frames("lag_frames10.txt");
printf("%d frames read for test case 10.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc10,"# ver: v");
printf("test case version: %s\n", desc10((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
```

```
                printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb10 = get_mcb_data("mcb_reg10.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain10, config10, unchain]= make_mcb_chains(mcb10);


## print chip configuration
nchains10= columns(Cchain10);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains10);
for ic=1:4:nchains10
                printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
                Cchain10(ic).Xinput,Cchain10(ic).CCC(1),\
                Cchain10(ic).CCC(end),Cchain10(ic).Yinput,\
                Cchain10(ic+1).Xinput,Cchain10(ic+1).CCC(1),\
                Cchain10(ic+1).CCC(end),Cchain10(ic+1).Yinput,\
                Cchain10(ic+2).Xinput,Cchain10(ic+2).CCC(1),\
                Cchain10(ic+2).CCC(end),Cchain10(ic+2).Yinput,\
                Cchain10(ic+3).Xinput,Cchain10(ic+3).CCC(1),\
                Cchain10(ic+3).CCC(end),Cchain10(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump10, ndump10, ibadstamp] = make_dump_index(frame10);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump10, with_chains10] = add_CCchain_dump(dump10, frame10);

                if !(ndump10 == rows(with_chains10))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp10=zeros(ndump10,1);
                for id=1:ndump10; timestamp10(id)= dump10(id).TIMESTAMP_0; endfor;
                if timestamp10(1) < 0
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp10(1)+2^32);
                else
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp10(1));
                endif
                if all(diff(timestamp10) == 1)
                                printf("Timestamps are sequential and increment by 1.\n");
                else
                                printf("Missing timestamps (not increment by 1).\n");
                endif

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump10,mcb10,Cchain10,applied_dump,"tc10",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb10.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

[bmod10] = get_bmodel_chains("test10", Cchain10, ndump10);
name_version_mdc= model_diff_chains;
```

```
printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc10";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [3.3333;3.3333;3.3333;3.3333;3.3333;3.3333;3.3333;3.3333]; # (number of samples per dump) / 12000
dmod=  [2;3;4;5;6;7;8;9]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain10,dump10,bmod10,tname,1,8,\
dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.14 tc10_postsim_validation_1

```
# make statistics tables using 12000 samples for all dumps and chains of the bmodel
# use first 2 dumps of bmodel data for tc11
[bmod10a] = get_bmodel_chains("test11", Cchain10, ndump10);

tname= "tc10a";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating a second table of model lag and frequency domain\n\
difference statistics with 12000 samples of the behavioural\n\
model for all chains; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult_a= [1;1;1;1;1;1;1;1];
dmod_a= [1;1;1;1;1;1;1;1];
make_stat_tables(Cchain10,dump10,bmod10a,tname,1,8,\
dmult_a,dmod_a,lag_peak_thold,fd_peak_thold,1);
```

3.5.15 tc11_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc11_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
              printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc12_jun14-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 1;
test_mcb.Y_DUMP_SYNC= 1;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [1;1;1;1;1;1;1;1];
test_mcb.Y_DATA= [1;1;1;1;1;1;1;1];
test_mcb.X_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.Y_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.X_DVALID= [1;1;1;1;1;1;1;1];
```

```
test_mcb.Y_DVALID= [1;1;1;1;1;1;1;1];
test_mcb.X_SE_CLK= [1;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [11;1;1;1;1;1;1;1];

test_DESSR= [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 11;
applied_dump= [12000, 12000, 12000, 12000, 12000, 12000, 12000, 12000];

## ----------------------------------------------------------------------------------
[frame11,num_frames,desc11,bad_frames,dontcare_frames]= get_lag_frames("lag_frames11.txt");
printf("%d frames read for test case 11.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc11,"# ver: v");
printf("test case version: %s\n", desc11((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb11 = get_mcb_data("mcb_reg11.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain11, config11, unchain]= make_mcb_chains(mcb11);


## print chip configuration
nchains11= columns(Cchain11);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains11);
for ic=1:4:nchains11
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain11(ic).Xinput,Cchain11(ic).CCC(1),\
               Cchain11(ic).CCC(end),Cchain11(ic).Yinput,\
               Cchain11(ic+1).Xinput,Cchain11(ic+1).CCC(1),\
               Cchain11(ic+1).CCC(end),Cchain11(ic+1).Yinput,\
               Cchain11(ic+2).Xinput,Cchain11(ic+2).CCC(1),\
               Cchain11(ic+2).CCC(end),Cchain11(ic+2).Yinput,\
               Cchain11(ic+3).Xinput,Cchain11(ic+3).CCC(1),\
               Cchain11(ic+3).CCC(end),Cchain11(ic+3).Yinput);
endfor
printf("\n");


## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump11, ndump11, ibadstamp] = make_dump_index(frame11);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump11, with_chains11] = add_CCchain_dump(dump11, frame11);

               if !(ndump11 == rows(with_chains11))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp11=zeros(ndump11,1);
               for id=1:ndump11; timestamp11(id)= dump11(id).TIMESTAMP_0; endfor;
               if timestamp11(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp11(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp11(1));
               endif
               if all(diff(timestamp11) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump11,mcb11,Cchain11,applied_dump,"tc11",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
```

```
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                        else
                                if Lwarn
                                        printf("\nWARNINGS encountered during register status and data capture
                                                checks;\n\
                                        details in log file.\n");
                                endif
                                if Ladvis
                                        printf("\nADVISORIES encountered during register status and data capture
                                                checks;\n\
                                        details in log file.\n");
                                endif
                        endif

                        # Check that DESSR bits are all set as expected.
                        if all(mcb11.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                        else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                        endif
endif

[bmod11] = get_bmodel_chains("test11", Cchain11, ndump11); # use bmodel for test case 11
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc11";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1;1;1;1;1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1;1;1;1;1]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain11,dump11,bmod11,tname,1,8,\
dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.16 tc11_inspect_status_captured

```
# print dumps of captured data that differ between cells of a chain
printf("\nchain|dump| XSyncerr | SID_X | SBID_X | BBID_X |\n");
ic=1;
for i=1:3
                if !(all(dump11(i).CCchain(ic).XSyncerr(1) == dump11(i).CCchain(ic).XSyncerr(:)) &&\
                all(dump11(i).CCchain(ic).SID_X(1) == dump11(i).CCchain(ic).SID_X(:)) &&\
                all(dump11(i).CCchain(ic).BBID_X(1) == dump11(i).CCchain(ic).BBID_X(:)) &&\
                all(dump11(i).CCchain(ic).SBID_X(1) == dump11(i).CCchain(ic).SBID_X(:)))
                        printf("%d | %d | %d %d | %d %d | %d %d | %d %d |\n",\
                        ic,i,dump11(i).CCchain(ic).XSyncerr',dump11(i).CCchain(ic).SID_X',\
                        dump11(i).CCchain(ic).SBID_X', dump11(i).CCchain(ic).BBID_X');
                endif
endfor

ic=5;
for i=1:3
                if !(all(dump11(i).CCchain(ic).XSyncerr(1) == dump11(i).CCchain(ic).XSyncerr(:)) &&\
                all(dump11(i).CCchain(ic).SID_X(1) == dump11(i).CCchain(ic).SID_X(:)) &&\
                all(dump11(i).CCchain(ic).BBID_X(1) == dump11(i).CCchain(ic).BBID_X(:)) &&\
                all(dump11(i).CCchain(ic).SBID_X(1) == dump11(i).CCchain(ic).SBID_X(:)))
                        printf("%d | %d | %d %d | %d %d | %d %d | %d %d |\n",\
                        ic,i,dump11(i).CCchain(ic).XSyncerr',dump11(i).CCchain(ic).SID_X',\
                        dump11(i).CCchain(ic).SBID_X', dump11(i).CCchain(ic).BBID_X');
                endif
endfor
```

3.5.17 tc12_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc11_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc12_jun14-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 1;
test_mcb.Y_DUMP_SYNC= 1;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [1;1;1;1;1;1;1;1];
test_mcb.Y_DATA= [1;1;1;1;1;1;1;1];
test_mcb.X_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.Y_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.X_DVALID= [1;1;1;1;1;1;1;1];
test_mcb.Y_DVALID= [1;1;1;1;1;1;1;1];
test_mcb.X_SE_CLK= [1;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [11;1;1;1;1;1;1;1];

test_DESSR= [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 11;
applied_dump= [12000, 12000, 12000, 12000, 12000, 12000, 12000, 12000];

## ----------------------------------------------------------------------------------------
[frame11,num_frames,desc11,bad_frames,dontcare_frames]= get_lag_frames("lag_frames11.txt");
printf("%d frames read for test case 11.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc11,"# ver: v");
printf("test case version: %s\n", desc11((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb11 = get_mcb_data("mcb_reg11.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain11, config11, unchain]= make_mcb_chains(mcb11);


## print chip configuration
nchains11= columns(Cchain11);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains11);
for ic=1:4:nchains11
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain11(ic).Xinput,Cchain11(ic).CCC(1),\
               Cchain11(ic).CCC(end),Cchain11(ic).Yinput,\
               Cchain11(ic+1).Xinput,Cchain11(ic+1).CCC(1),\
               Cchain11(ic+1).CCC(end),Cchain11(ic+1).Yinput,\
               Cchain11(ic+2).Xinput,Cchain11(ic+2).CCC(1),\
               Cchain11(ic+2).CCC(end),Cchain11(ic+2).Yinput,\
               Cchain11(ic+3).Xinput,Cchain11(ic+3).CCC(1),\
               Cchain11(ic+3).CCC(end),Cchain11(ic+3).Yinput);
endfor
```

```
printf("\n");


## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump11, ndump11, ibadstamp] = make_dump_index(frame11);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump11, with_chains11] = add_CCchain_dump(dump11, frame11);

               if !(ndump11 == rows(with_chains11))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp11=zeros(ndump11,1);
               for id=1:ndump11; timestamp11(id)= dump11(id).TIMESTAMP_0; endfor;
               if timestamp11(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp11(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp11(1));
               endif
               if all(diff(timestamp11) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump11,mcb11,Cchain11,applied_dump,"tc11",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb11.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod11] = get_bmodel_chains("test11", Cchain11, ndump11); # use bmodel for test case 11
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc11";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1;1;1;1;1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1;1;1;1;1]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain11,dump11,bmod11,tname,1,8,\
dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.18 tc14_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc14_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc14_apr22-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [0;0;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 85;
tc_SID_Y= 170;

tc= 14;
applied_dump= [20000, 20000, 0, 0, 0, 0, 0, 0];

## ----------------------------------------------------------------------------------------
[frame14,num_frames,desc14,bad_frames,dontcare_frames]= get_lag_frames("lag_frames14.txt");
printf("%d frames read for test case 14.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc14,"# ver: v");
printf("test case version: %s\n", desc14((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb14 = get_mcb_data("mcb_reg14.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain14, config14, unchain]= make_mcb_chains(mcb14);


## print chip configuration
nchains14= columns(Cchain14);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains14);
for ic=1:4:nchains14
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain14(ic).Xinput,Cchain14(ic).CCC(1),\
               Cchain14(ic).CCC(end),Cchain14(ic).Yinput,\
               Cchain14(ic+1).Xinput,Cchain14(ic+1).CCC(1),\
               Cchain14(ic+1).CCC(end),Cchain14(ic+1).Yinput,\
               Cchain14(ic+2).Xinput,Cchain14(ic+2).CCC(1),\
               Cchain14(ic+2).CCC(end),Cchain14(ic+2).Yinput,\
               Cchain14(ic+3).Xinput,Cchain14(ic+3).CCC(1),\
               Cchain14(ic+3).CCC(end),Cchain14(ic+3).Yinput);
endfor
```

```
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump14, ndump14, ibadstamp] = make_dump_index(frame14);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump14, with_chains14] = add_CCchain_dump(dump14, frame14);

               if !(ndump14 == rows(with_chains14))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp14=zeros(ndump14,1);
               for id=1:ndump14; timestamp14(id)= dump14(id).TIMESTAMP_0; endfor;
               if timestamp14(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp14(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp14(1));
               endif
               if all(diff(timestamp14) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump14,mcb14,Cchain14,applied_dump,"tc14",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
               checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
               checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb14.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod14] = get_bmodel_chains("test14", Cchain14, 3);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc14";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1.6667;1.6667;1.6667;1.6667]; # (number of samples per dump) / 12000
dmod=  [2;2;2;2]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain14,dump14,bmod14,tname,1,4,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.19 tc25_postsim

```
printf("***------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc25_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc25_jun14-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [0;0;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 25;
applied_dump= [12000, 12000, 0, 0, 0, 0, 0, 0];

## ----------------------------------------------------------------------------------------
[frame25,num_frames,desc25,bad_frames,dontcare_frames]= get_lag_frames("lag_frames25.txt");
printf("%d frames read for test case 25.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc25,"# ver: v");
printf("test case version: %s\n", desc25((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb25 = get_mcb_data("mcb_reg25.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain25, config25, unchain]= make_mcb_chains(mcb25);


## print chip configuration
nchains25= columns(Cchain25);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains25);
for ic=1:4:nchains25
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain25(ic).Xinput,Cchain25(ic).CCC(1),\
               Cchain25(ic).CCC(end),Cchain25(ic).Yinput,\
               Cchain25(ic+1).Xinput,Cchain25(ic+1).CCC(1),\
               Cchain25(ic+1).CCC(end),Cchain25(ic+1).Yinput,\
               Cchain25(ic+2).Xinput,Cchain25(ic+2).CCC(1),\
               Cchain25(ic+2).CCC(end),Cchain25(ic+2).Yinput,\
               Cchain25(ic+3).Xinput,Cchain25(ic+3).CCC(1),\
               Cchain25(ic+3).CCC(end),Cchain25(ic+3).Yinput);
endfor
```

```
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump25, ndump25, ibadstamp] = make_dump_index(frame25);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump25, with_chains25] = add_CCchain_dump(dump25, frame25);

               if !(ndump25 == rows(with_chains25))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp25=zeros(ndump25,1);
               for id=1:ndump25; timestamp25(id)= dump25(id).TIMESTAMP_0; endfor;
               if timestamp25(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp25(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp25(1));
               endif
               if all(diff(timestamp25) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump25,mcb25,Cchain25,applied_dump,"tc25",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb25.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod25] = get_bmodel_chains("test25", Cchain25, 10);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc25";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1;1;1;1;1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [2;2;2;2;2;2;2;2]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain25,dump25,bmod25,tname,1,8,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.20 tc25_chain_similarity_check

```
Lnz= !ones(4,ndump25);  # flag any non-zero difference
for id=1:ndump25
                for ic=1:4
                                Lnz(ic,id)= any(abs(dump25(id).CCchain(ic).clag-\
                                                dump25(id).CCchain(ic+4).clag) >= 10*eps);
                endfor
endfor

if !any(any(Lnz))
                printf("\nNo differences exceeding the floating point precision limit of this\n\
machine were found between lag data of similar chains for any of the %d dumps.\n\n", ndump25);
else
                printf("\nWARNING: Differences exceeding the floating point precision limit\n\
                  of this machine were found between similar chains.\n\n");
endif
```

3.5.21 tc30_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc30_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
                printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc30_jun14-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 30;
applied_dump= [12000, 12000, 24000, 24000, 48000, 48000, 96000, 96000];

## ----------------------------------------------------------------------------------------
[frame30,num_frames,desc30,bad_frames,dontcare_frames]= get_lag_frames("lag_frames30.txt");
printf("%d frames read for test case 30.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc30,"# ver: v");
printf("test case version: %s\n", desc30((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
                printf("Checksum is OK for all frames and corrupt data were not\n");
                printf("detected (no 'x' or 'X' hex digits were found).\n");
endif
```

```
## Read the MCB register data from the register contents file.
mcb30 = get_mcb_data("mcb_reg30.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain30, config30, unchain]= make_mcb_chains(mcb30);


## print chip configuration
nchains30= columns(Cchain30);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains30);
for ic=1:4:nchains30
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain30(ic).Xinput,Cchain30(ic).CCC(1),\
               Cchain30(ic).CCC(end),Cchain30(ic).Yinput,\
               Cchain30(ic+1).Xinput,Cchain30(ic+1).CCC(1),\
               Cchain30(ic+1).CCC(end),Cchain30(ic+1).Yinput,\
               Cchain30(ic+2).Xinput,Cchain30(ic+2).CCC(1),\
               Cchain30(ic+2).CCC(end),Cchain30(ic+2).Yinput,\
               Cchain30(ic+3).Xinput,Cchain30(ic+3).CCC(1),\
               Cchain30(ic+3).CCC(end),Cchain30(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump30, ndump30, ibadstamp] = make_dump_index(frame30);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump30, with_chains30] = add_CCchain_dump(dump30, frame30);

               if !(ndump30 == rows(with_chains30))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp30=zeros(ndump30,1);
               for id=1:ndump30; timestamp30(id)= dump30(id).TIMESTAMP_0; endfor;
               if timestamp30(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp30(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp30(1));
               endif
               if all(diff(timestamp30) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif


               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump30,mcb30,Cchain30,applied_dump,"tc30",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb30.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod30] = get_bmodel_chains("test30", Cchain30, 16);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
```

```
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc30";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1;1;1;1;2;2;2;2;4;4;4;4;8;8;8;8]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain30,dump30,bmod30,tname,1,16,dmult,dmod,\
lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.5.22 tc30_bit_exact

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc30_bit_exact.m : V1.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
[frame30,num_frames,desc30,bad_frames,dontcare_frames]= get_lag_frames("lag_frames30-S.txt");
printf("%d frames read for test case 30.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc30,"# ver: v");
printf("test case version: %s\n", desc30((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb30 = get_mcb_data("mcb_reg30.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain30, config30, unchain]= make_mcb_chains(mcb30);


## print chip configuration
nchains30= columns(Cchain30);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains30);
for ic=1:4:nchains30
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain30(ic).Xinput,Cchain30(ic).CCC(1),\
               Cchain30(ic).CCC(end),Cchain30(ic).Yinput,\
               Cchain30(ic+1).Xinput,Cchain30(ic+1).CCC(1),\
               Cchain30(ic+1).CCC(end),Cchain30(ic+1).Yinput,\
               Cchain30(ic+2).Xinput,Cchain30(ic+2).CCC(1),\
               Cchain30(ic+2).CCC(end),Cchain30(ic+2).Yinput,\
               Cchain30(ic+3).Xinput,Cchain30(ic+3).CCC(1),\
               Cchain30(ic+3).CCC(end),Cchain30(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump30, ndump30, ibadstamp] = make_dump_index(frame30);

id=1;
if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump30, with_chains30] = add_CCchain_dump(dump30, frame30);
```

```
                if !(ndump30 == rows(with_chains30))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp30(id)= dump30(id).TIMESTAMP_0;
                printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp30(1));
endif

[bmod30] = get_bmodel_chains("test30-S", Cchain30, 1);
name_version_mdc= model_diff_chains;


printf("***--------------------------------------------------------------------------***\n");
```

3.5.23 tc64_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc64_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
                printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ------------------------------------------------------------------------------------
## ------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc64_apr22-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [0;0;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc=64;
applied_dump= [20000, 20000, 0, 0, 0, 0, 0, 0];

## ------------------------------------------------------------------------------------
[frame64,num_frames,desc64,bad_frames,dontcare_frames]= get_lag_frames("lag_frames64.txt");
printf("%d frames read for test case 64.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc64,"# ver: v");
printf("test case version: %s\n", desc64((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
                printf("Checksum is OK for all frames and corrupt data were not\n");
                printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb64 = get_mcb_data("mcb_reg64.txt");

## Form chains and input configuration based on the contents of the MCB register file.
```

```
[Cchain64, config64, unchain]= make_mcb_chains(mcb64);


## print chip configuration
nchains64= columns(Cchain64);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains64);
for ic=1:4:nchains64
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain64(ic).Xinput,Cchain64(ic).CCC(1),\
               Cchain64(ic).CCC(end),Cchain64(ic).Yinput,\
               Cchain64(ic+1).Xinput,Cchain64(ic+1).CCC(1),\
               Cchain64(ic+1).CCC(end),Cchain64(ic+1).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump64, ndump64, ibadstamp] = make_dump_index(frame64);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump64, with_chains64] = add_CCchain_dump(dump64, frame64);

               if !(ndump64 == rows(with_chains64))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp64=zeros(ndump64,1);
               for id=1:ndump64; timestamp64(id)= dump64(id).TIMESTAMP_0; endfor;
               if timestamp64(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp64(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp64(1));
               endif
               if all(diff(timestamp64) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif


               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump64,mcb64,Cchain64,applied_dump,"tc64",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb64.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod64] = get_bmodel_chains("test64", Cchain64, 2);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc64";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);
```

```
printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1.6667;1.6667]; # (number of samples per dump) / 12000
dmod=  [2;2]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain64,dump64,bmod64,tname,1,2,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.6 shift enable clock (X|Y-SE_CLK)

3.6.1 tc21_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc21_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc21_jun14-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;1;1;0;1;1;1];
test_mcb.Y_SE_CLK= [1;1;1;0;1;1;1;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc=21;
applied_dump= [24000, 0, 0, 0, 12000, 0, 0, 0];

## ----------------------------------------------------------------------------------------
[frame21,num_frames,desc21,bad_frames,dontcare_frames]= get_lag_frames("lag_frames21.txt");
printf("%d frames read for test case 21.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc21,"# ver: v");
printf("test case version: %s\n", desc21((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb21 = get_mcb_data("mcb_reg21.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain21, config21, unchain]= make_mcb_chains(mcb21);
```

```
## print chip configuration
nchains21= columns(Cchain21);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains21);
for ic=1:4:nchains21
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain21(ic).Xinput,Cchain21(ic).CCC(1),\
               Cchain21(ic).CCC(end),Cchain21(ic).Yinput,\
               Cchain21(ic+1).Xinput,Cchain21(ic+1).CCC(1),\
               Cchain21(ic+1).CCC(end),Cchain21(ic+1).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump21, ndump21, ibadstamp] = make_dump_index(frame21);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump21, with_chains21] = add_CCchain_dump(dump21, frame21);

               if !(ndump21 == rows(with_chains21))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp21=zeros(ndump21,1);
               for id=1:ndump21; timestamp21(id)= dump21(id).TIMESTAMP_0; endfor;
               if timestamp21(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp21(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp21(1));
               endif
               if all(diff(timestamp21) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif


               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump21,mcb21,Cchain21,applied_dump,"tc21",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb21.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod21] = get_bmodel_chains("test21", Cchain21, 4);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc21";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);
```

```
printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [2;1]; # (number of samples per dump) / 12000
dmod=  [2;1]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain21,dump21,bmod21,tname,1,2,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.6.2 tc22_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc22_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc22_jul24-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;1;1;0;1;1;1];
test_mcb.Y_SE_CLK= [1;1;1;0;1;1;1;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc=22;
applied_dump= [24000, 0, 0, 0, 24000, 0, 0, 0];

## ----------------------------------------------------------------------------------------
[frame22,num_frames,desc22,bad_frames,dontcare_frames]= get_lag_frames("lag_frames22.txt");
printf("%d frames read for test case 22.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc22,"# ver: v");
printf("test case version: %s\n", desc22((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb22 = get_mcb_data("mcb_reg22.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain22, config22, unchain]= make_mcb_chains(mcb22);


## print chip configuration
nchains22= columns(Cchain22);
```

```
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains22);
for ic=1:4:nchains22
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain22(ic).Xinput,Cchain22(ic).CCC(1),\
               Cchain22(ic).CCC(end),Cchain22(ic).Yinput,\
               Cchain22(ic+1).Xinput,Cchain22(ic+1).CCC(1),\
               Cchain22(ic+1).CCC(end),Cchain22(ic+1).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump22, ndump22, ibadstamp] = make_dump_index(frame22);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump22, with_chains22] = add_CCchain_dump(dump22, frame22);

               if !(ndump22 == rows(with_chains22))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp22=zeros(ndump22,1);
               for id=1:ndump22; timestamp22(id)= dump22(id).TIMESTAMP_0; endfor;
               if timestamp22(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp22(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp22(1));
               endif
               if all(diff(timestamp22) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif


               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump22,mcb22,Cchain22,applied_dump,"tc22",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb22.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod22] = get_bmodel_chains("test22", Cchain22, 2);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc22";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [2;2]; # (number of samples per dump) / 12000
```

```
dmod=  [1;2]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain22,dump22,bmod22,tname,1,2,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.7 input configuration switches

3.7.1 tc15_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc15_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ------------------------------------------------------------------------------------
## ------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc15_apr22-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
tc= 15; # test case number (required for BBID checks)

applied_dump= [0, 10000, 0, 0, 0, 0, 0, 0];

## ------------------------------------------------------------------------------------
[frame15,num_frames,desc15,bad_frames,dontcare_frames]= get_lag_frames("lag_frames15.txt");
printf("%d frames read for test case 15.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc15,"# ver: v");
printf("test case version: %s\n", desc15((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb15 = get_mcb_data("mcb_reg15.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain15, config15, unchain]= make_mcb_chains(mcb15);


## print chip configuration
nchains15= columns(Cchain15);
```

```
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains15);
if columns(config15) > 72
              printf("%s\n", config15(1:72));
              printf("%s\n", config15(73:end));
else
              printf("%s\n", config15);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump15, ndump15, ibadstamp] = make_dump_index(frame15);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

              # form contatenated chains for each dump on the basis of the output frame
              # status bits and add these to the dump structure.
              [dump15, with_chains15] = add_CCchain_dump(dump15, frame15);

              if !(ndump15 == rows(with_chains15))
                            printf("\nADVISORY: NOT ALL dumps contain chains!\n");
              endif
              timestamp15(1)= dump15(1).TIMESTAMP_0;
              printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp15(1));

              # check consistency of STATUS and MCSR register states; data_bias and data captured
              # from data and dump control streams. Check consistency of mcb contents with those
              # of the structure test_mcb.
              [Lwarn, Ladvis] = check_register_capture(dump15,mcb15,Cchain15,applied_dump,"tc15",tc_SID_X,\
              tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

              if !Lwarn & !Ladvis
                            printf("\nCheck OK: no inconsistencies reported in captured data\n");
                            printf("          or register contents; no unexpected register reports;\n");
                            printf("          details in log file.\n");
              else
                            if Lwarn
                                          printf("\nWARNINGS encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
                            if Ladvis
                                          printf("\nADVISORIES encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
              endif

              # Check that DESSR bits are all set as expected.
              if all(mcb15.DESSR == test_DESSR)
                            printf("\nCheck OK: DESSR bits are set as expected.\n");
              else
                            printf("\nWARNING: DESSR bits are set NOT as expected.\n");
              endif
endif

[bmod15] = get_bmodel_chains("test15", Cchain15, ndump15); # use bmodel for test case 15
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc15";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [0.83333;0.83333;0.83333;0.83333]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain15,dump15,bmod15,tname,1,4,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.7.2 tc15_similarity_check

```
Lnz= !ones(8,ndump15);  # flag any non-zero difference
id=1;
for ic=1:2
               Lnz(ic,id)= any(abs(dump15(id).CCchain(ic).clag-\
               dump15(id).CCchain(ic+1).clag) >= 10*eps);
endfor

if !any(any(Lnz))
               printf("\nNo differences exceeding the floating point precision limit of this\n\
               machine were found between similar chains of test case 15.\n\n");
else
               printf("\nADVISORY: Differences exceeding the floating point precision limit\n\
                 of this machine were found between lag data of similar chains\n\
                 within test case 15.\n\n");
endif
```

3.7.3 tc16_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc16_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc16_apr22-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
tc= 16; # test case number (required for BBID checks)

applied_dump= [0, 10000, 0, 0, 0, 0, 0, 0];

## ----------------------------------------------------------------------------------------
[frame16,num_frames,desc16,bad_frames,dontcare_frames]= get_lag_frames("lag_frames16.txt");
printf("%d frames read for test case 16.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc16,"# ver: v");
printf("test case version: %s\n", desc16((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
```

```
mcb16 = get_mcb_data("mcb_reg16.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain16, config16, unchain]= make_mcb_chains(mcb16);
nchains16= columns(Cchain16);

printf("\nInput configuration from MCSR regsister settings:\n");
printf("List chains from 1 to %d.\n", nchains16);
if columns(config16) > 72
               printf("%s\n", config16(1:72));
               printf("%s\n", config16(73:end));
else
               printf("%s\n", config16);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump16, ndump16, ibadstamp] = make_dump_index(frame16);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump16, with_chains16] = add_CCchain_dump(dump16, frame16);

               if !(ndump16 == rows(with_chains16))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp16(1)= dump16(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp16(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump16,mcb16,Cchain16,applied_dump,"tc16",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb16.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

[bmod16] = get_bmodel_chains("test16", Cchain16, ndump16); # use bmodel for test case 16
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc16";
lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [0.83333;0.83333;0.83333;0.83333]; # (number of samples per dump) / 12000
dmod=  [2;2;2;2]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain16,dump16,bmod16,tname,1,4,dmult,dmod,lag_peak_thold,fd_peak_thold,1);

printf("***--------------------------------------------------------------------------***\n");
```

3.7.4 tc16_similarity_check

```
Lnz= !ones(8,ndump16);  # flag any non-zero difference
id=1;
for ic=1:3
               Lnz(ic,id)= any(abs(dump16(id).CCchain(ic).clag-\
               dump16(id).CCchain(ic+1).clag) >= 10*eps);
endfor

if !any(any(Lnz))
               printf("\nNo differences exceeding the floating point precision limit of this\n\
               machine were found between similar chains of test case 16.\n\n");
else
               printf("\nADVISORY: Differences exceeding the floating point precision limit\n\
                 of this machine were found between lag data of similar chains\n\
                 within test case 16.\n\n");
endif
```

3.7.5 tc17_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc17_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc17_apr27-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [1;0;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [1;0;1;0;1;0;1;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 21;
base_SBID_Y= 10;
tc_SID_X= 85;
tc_SID_Y= 170;

tc= 17;
applied_dump= [0; 12000; 0; 0; 0; 0; 0; 0];

## ----------------------------------------------------------------------------------------
[frame17,num_frames,desc17,bad_frames,dontcare_frames]= get_lag_frames("lag_frames17.txt");
printf("%d frames read for test case 17.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc17,"# ver: v");
printf("test case version: %s\n", desc17((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
```

```
mcb17 = get_mcb_data("mcb_reg17.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain17, config17, unchain]= make_mcb_chains(mcb17);

## print chip configuration
nchains17= columns(Cchain17);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains17);
if columns(config17) > 72
               printf("%s\n", config17(1:72));
               printf("%s\n", config17(73:end));
else
               printf("%s\n", config17);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump17, ndump17, ibadstamp] = make_dump_index(frame17);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump17, with_chains17] = add_CCchain_dump(dump17, frame17);

               if !(ndump17 == rows(with_chains17))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp17=zeros(ndump17,1);
               for id=1:ndump17; timestamp17(id)= dump17(id).TIMESTAMP_0; endfor;
               if timestamp17(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp17(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp17(1));
               endif
               if all(diff(timestamp17) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump17,mcb17,Cchain17,applied_dump,"tc17",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb17.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

tname="tc17";
printf("\nWriting frequency domain plot files.\n");
for ic=1:4
               write_fmod_fd_plots(ic,1,dump17,Cchain17,tname,-40,5,1,1);
endfor
printf("Writing data ID capture tables.\n");
tc17_data_ID_capture
printf("Writing timestamp capture tables.\n");
tc17_timestamp_capture

printf("***--------------------------------------------------------------------------***\n");
```

3.7.6 tc17_make_fb_stat_tables

```
## Generate tables of functional-behavioural model statistics
[bmod17] = get_bmodel_chains("test17", Cchain17, 2);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
 dmult= [1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain17,dump17,bmod17,tname,1,4,dmult,dmod,lag_peak_thold,fd_peak_thold,1);
```

3.7.7 tc18_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc18_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc18_apr27-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [1;0;1;0;1;0;1;0];
test_mcb.Y_SE_CLK= [1;0;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 21;
base_SBID_Y= 10;
tc_SID_X= 85;
tc_SID_Y= 170;
tc=18;
applied_dump= [0; 12000; 0; 12000; 0; 12000; 0; 12000];

## ----------------------------------------------------------------------------------
[frame18,num_frames,desc18,bad_frames,dontcare_frames]= get_lag_frames("lag_frames18.txt");
printf("%d frames read for test case 18.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc18,"# ver: v");
printf("test case version: %s\n", desc18((verdate_start+7):(verdate_start+16)));
```

```
if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb18 = get_mcb_data("mcb_reg18.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain18, config18, unchain]= make_mcb_chains(mcb18);


## print chip configuration
nchains18= columns(Cchain18);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains18);

if columns(config18) > 72
               printf("%s\n", config18(1:72));
               printf("%s\n", config18(73:end));
else
               printf("%s\n", config18);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump18, ndump18, ibadstamp] = make_dump_index(frame18);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump18, with_chains18] = add_CCchain_dump(dump18, frame18);

               if !(ndump18 == rows(with_chains18))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp18=zeros(ndump18,1);
               for id=1:ndump18; timestamp18(id)= dump18(id).TIMESTAMP_0; endfor;
               if timestamp18(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp18(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp18(1));
               endif
               if all(diff(timestamp18) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump18,mcb18,Cchain18,applied_dump,"tc18",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb18.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

tname="tc18";
printf("\nWriting frequency domain plot files.\n");
for ic=1:4
```

```
            write_fmod_fd_plots(ic,1,dump18,Cchain18,tname,-40,5,1,1);
endfor
printf("Writing data ID capture tables.\n");
tc18_data_ID_capture
printf("Writing timestamp capture tables.\n");
tc18_timestamp_capture

printf("***--------------------------------------------------------------------------***\n");
```

3.7.8 tc18_make_fb_stat_tables

```
## Generate tables of functional-behavioural model statistics
[bmod19] = get_bmodel_chains("test19", Cchain19, 2);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain19,dump19,bmod19,tname,1,4,dmult,dmod,lag_peak_thold,fd_peak_thold,1);
```

3.7.9 tc19_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc19_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
            printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc19_jun14-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;1;1;1;1;1;1];
test_mcb.Y_SE_CLK= [0;1;0;1;0;1;0;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 10;
base_SBID_Y= 21;
tc_SID_X= 170;
tc_SID_Y= 85;
```

```
tc=19;
applied_dump= [12000, 0, 12000, 0, 12000, 0, 12000, 0];

## ----------------------------------------------------------------------------------
[frame19,num_frames,desc19,bad_frames,dontcare_frames]= get_lag_frames("lag_frames19.txt");
printf("%d frames read for test case 19.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc19,"# ver: v");
printf("test case version: %s\n", desc19((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb19 = get_mcb_data("mcb_reg19.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain19, config19, unchain]= make_mcb_chains(mcb19);

## print chip configuration
nchains19= columns(Cchain19);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains19);
if columns(config19) > 72
               printf("%s\n", config19(1:72));
               printf("%s\n", config19(73:end));
else
               printf("%s\n", config19);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump19, ndump19, ibadstamp] = make_dump_index(frame19);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump19, with_chains19] = add_CCchain_dump(dump19, frame19);

               if !(ndump19 == rows(with_chains19))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp19=zeros(ndump19,1);
               for id=1:ndump19; timestamp19(id)= dump19(id).TIMESTAMP_0; endfor;
               if timestamp19(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp19(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp19(1));
               endif
               if all(diff(timestamp19) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif


               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump19,mcb19,Cchain19,applied_dump,"tc19",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb19.DESSR == test_DESSR)
```

```
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

tname= "tc19";
printf("\nWriting frequency domain plot files.\n");
for ic=1:4
                write_fmod_fd_plots(ic,1,dump19,Cchain19,tname,-40,5,1,1);
endfor
printf("Writing data ID capture tables.\n");
tc19_data_ID_capture
printf("Writing timestamp capture tables.\n");
tc19_timestamp_capture

printf("***--------------------------------------------------------------------------***\n");
```

3.7.10 tc19_make_fb_stat_tables

```
## Generate tables of functional-behavioural model statistics
[bmod19] = get_bmodel_chains("test19", Cchain19, 2);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain19,dump19,bmod19,tname,1,4,dmult,dmod,lag_peak_thold,fd_peak_thold,1);
```

3.7.11 tc20_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc20_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
                printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc20_apr27-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;0;1;0;1;0;1];
```

```
test_mcb.Y_SE_CLK= [0;1;1;1;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 10;
base_SBID_Y= 21;
tc_SID_X= 170;
tc_SID_Y= 85;
tc=20;
applied_dump= [12000; 0; 0; 0; 0; 0; 0; 0];

## ----------------------------------------------------------------------------
[frame20,num_frames,desc20,bad_frames,dontcare_frames]= get_lag_frames("lag_frames20.txt");
printf("%d frames read for test case 20.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc20,"# ver: v");
printf("test case version: %s\n", desc20((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
              printf("Checksum is OK for all frames and corrupt data were not\n");
              printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb20 = get_mcb_data("mcb_reg20.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain20, config20, unchain]= make_mcb_chains(mcb20);

## print chip configuration
nchains20= columns(Cchain20);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains20);
if columns(config20) > 72
              printf("%s\n", config20(1:72));
              printf("%s\n", config20(73:end));
else
              printf("%s\n", config20);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump20, ndump20, ibadstamp] = make_dump_index(frame20);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

              # form contatenated chains for each dump on the basis of the output frame
              # status bits and add these to the dump structure.
              [dump20, with_chains20] = add_CCchain_dump(dump20, frame20);

              if !(ndump20 == rows(with_chains20))
                            printf("\nADVISORY: NOT ALL dumps contain chains!\n");
              endif

              timestamp20=zeros(ndump20,1);
              for id=1:ndump20; timestamp20(id)= dump20(id).TIMESTAMP_0; endfor;
              if timestamp20(1) < 0
                            printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp20(1)+2^32);
              else
                            printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp20(1));
              endif
              if all(diff(timestamp20) == 1)
                            printf("Timestamps are sequential and increment by 1.\n");
              else
                            printf("Missing timestamps (not increment by 1).\n");
              endif


              # check consistency of STATUS and MCSR register states; data_bias and data captured
              # from data and dump control streams. Check consistency of mcb contents with those
              # of the structure test_mcb.
              [Lwarn, Ladvis] = check_register_capture(dump20,mcb20,Cchain20,applied_dump,"tc20",tc_SID_X,\
              tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

              if !Lwarn & !Ladvis
                            printf("\nCheck OK: no inconsistencies reported in captured data\n");
                            printf("          or register contents; no unexpected register reports;\n");
                            printf("          details in log file.\n");
              else
                            if Lwarn
                                          printf("\nWARNINGS encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
                            if Ladvis
```

```
                                        printf("\nADVISORIES encountered during register status and data capture
                                                checks;\n\
                                        details in log file.\n");
                        endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb20.DESSR == test_DESSR)
                        printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                        printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

tname="tc20";
printf("\nWriting frequency domain plot files.\n");
for ic=1:4
                write_fmod_fd_plots(ic,1,dump20,Cchain20,tname,-40,5,1,1);
endfor
printf("Writing data ID capture tables.\n");
tc20_data_ID_capture
printf("Writing timestamp capture tables.\n");
tc20_timestamp_capture

printf("***--------------------------------------------------------------------------***\n");
```

3.7.12 tc20_make_fb_stat_tables

```
## Generate tables of functional-behavioural model statistics
[bmod20] = get_bmodel_chains("test20", Cchain20, 2);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

lag_peak_thold= 3;
fd_peak_thold= 10;
printf("Generating tables of model lag and frequency domain\n\
difference statistics; using name: %s\n",tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [1;1;1;1]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1]; # SE_CLK modulus (1 -> always set)
make_stat_tables(Cchain20,dump20,bmod20,tname,1,4,dmult,dmod,lag_peak_thold,fd_peak_thold,1);
```

3.7.13 tc68_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc68_postsim.m : V1.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc68_nov04-05_v1.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;1;1;1;1;0;0];
test_mcb.Y_SE_CLK= [0;0;1;1;1;1;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
tc= 68; # test case number (required for BBID checks)

applied_dump= [1000, 1000, 0, 0, 0, 0, 1000, 1000];

## ----------------------------------------------------------------------------------------
[frame68,num_frames,desc68,bad_frames,dontcare_frames]= get_lag_frames("lag_frames68.txt");
printf("%d frames read for test case 68.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc68,"# ver: v");
printf("test case version: %s\n", desc68((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb68 = get_mcb_data("mcb_reg68.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain68, config68, unchain]= make_mcb_chains(mcb68);


## print chip configuration
nchains68= columns(Cchain68);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains68);
if columns(config68) > 72
               printf("%s\n", config68(1:72));
               printf("%s\n", config68(73:end));
else
               printf("%s\n", config68);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump68, ndump68, ibadstamp] = make_dump_index(frame68);
```

```
if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump68, with_chains68] = add_CCchain_dump(dump68, frame68);

               if !(ndump68 == rows(with_chains68))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp68(1)= dump68(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp68(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump68,mcb68,Cchain68,applied_dump,"tc68",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb68.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

tname= "tc68";

for ic=1:12
               write_fmod_fd_plots(ic,1,dump68,Cchain68,tname,-40,5,1,1);
endfor

printf("***--------------------------------------------------------------------------***\n");
```

3.7.14 tc68_similarity_check

```
Lnz= !ones(4,1);  # flag any non-zero difference
id=1;
for ic=1:4
               Lnz(ic,id)= any(abs(dump68(id).CCchain(ic).clag-\
               dump68(id).CCchain(ic+8).clag) >= 10*eps);
endfor

if !any(any(Lnz))
               printf("\nNo differences exceeding the floating point precision limit of this\n\
               machine were found between similar chains of test case 68.\n\n");
else
               printf("\nADVISORY: Differences exceeding the floating point precision limit\n\
                 of this machine were found between lag data of similar chains\n\
                 within test case 68.\n\n");
endif
```

3.8 extreme dump rate operation

3.8.1 tc07_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc07_postsim.m : V3.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
        printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc07_apr21-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 1;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;0;1;0;1;0;1];
test_mcb.Y_SE_CLK= [0;1;0;1;0;1;0;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
tc= 7;
applied_dump=[80, 0, 2000, 0, 10000, 0, 20000, 0];

[frame07, num_frames, desc07, bad_frames, dontcare_frames]= get_lag_frames("lag_frames07.txt");
printf("%d frames read for test case 07.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc07,"# ver: v");
printf("test case version: %s\n", desc07((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
        printf("Checksum is OK for all frames and corrupt data were not\n");
        printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb07 = get_mcb_data("mcb_reg07.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain07, config07, unchain]= make_mcb_chains(mcb07);

## print chip configuration
nchains07= columns(Cchain07);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains07);
for ic=1:4:nchains07
        printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
        Cchain07(ic).Xinput,Cchain07(ic).CCC(1),\
        Cchain07(ic).CCC(end),Cchain07(ic).Yinput,\
        Cchain07(ic+1).Xinput,Cchain07(ic+1).CCC(1),\
        Cchain07(ic+1).CCC(end),Cchain07(ic+1).Yinput,\
        Cchain07(ic+2).Xinput,Cchain07(ic+2).CCC(1),\
        Cchain07(ic+2).CCC(end),Cchain07(ic+2).Yinput,\
        Cchain07(ic+3).Xinput,Cchain07(ic+3).CCC(1),\
        Cchain07(ic+3).CCC(end),Cchain07(ic+3).Yinput);
endfor

printf("\n");
```

```
[dump07, ndump07, ibadstamp] = make_dump_index(frame07);
if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

        # form contatenated chains for each dump on the basis of the output frame
        # status bits and add these to the dump structure.
        [dump07, with_chains07] = add_CCchain_dump(dump07, frame07);

        if !(ndump07 == rows(with_chains07))
                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
        endif

        timestamp07=zeros(ndump07,1);
        for id=1:ndump07; timestamp07(id)= dump07(id).TIMESTAMP_0; endfor;
        if timestamp07(1) < 0
                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp07(1)+2^32);
        else
                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp07(1));
        endif
        if all(diff(timestamp07) == 1)
                printf("Timestamps are sequential and increment by 1.\n");
        else
                printf("Missing timestamps (not increment by 1).\n");
        endif

        # check consistency of STATUS and MCSR register states; data_bias and data captured
        # from data and dump control streams. Check consistency of mcb contents with those
        # of the structure test_mcb.
        [Lwarn, Ladvis] = check_register_capture(dump07,mcb07,Cchain07,applied_dump,"tc07",tc_SID_X,\
        tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

        if !Lwarn & !Ladvis
                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                printf("          or register contents; no unexpected register reports;\n");
                printf("          details in log file.\n");
        else
                if Lwarn
                        printf("\nWARNINGS encountered during register status and data capture checks;\n\
                        details in log file.\n");
                endif
                if Ladvis
                        printf("\nADVISORIES encountered during register status and data capture checks;\n\
                        details in log file.\n");
                endif
        endif

        # Check that DESSR bits are all set as expected.
        if all(mcb07.DESSR == test_DESSR)
                printf("\nCheck OK: DESSR bits are set as expected.\n");
        else
                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
        endif
endif

printf("\nNumber of dumps: %d\n",ndump07);

rate1_cells= [0;1;2;3];
rate2_cells= [4;5;6;7];
rate3_cells= [8;9;10;11];
rate4_cells= [12;13;14;15];

## get dump overrun information from the output frames
## the first dump does not occur; supply timestamp of initial dump
## Assure that version 1.10 or greater of list_dump_ovr is available!
name_version_ldo= list_dump_ovr;
printf("\nUsing list_dump_ovr: %s\n",name_version_ldo);
tstart= 222333444;

[out1, nrate1dump, tsout1, cellout1, OVRout1, idump1, icell1, icell_OVR1]= ...
                        list_dump_ovr(frame07,dump07,1, rate1_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate: %d\n", nrate1dump);

[out2, nrate2dump, tsout2, cellout2, OVRout2, idump2, icell2, icell_OVR2]= ...
                        list_dump_ovr(frame07,dump07,4, rate2_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/10: %d\n", nrate2dump);

[out3, nrate2dump, tsout3, cellout3, OVRout3, idump3, icell3, icell_OVR3]= ...
                        list_dump_ovr(frame07,dump07, 20, rate3_cells,tstart);

printf("Requested number of dumps of cells dumping at the highest rate/50: %d\n", nrate2dump);

[out4, nrate4dump, tsout4, cellout4, OVRout4, idump4, icell4, icell_OVR4]= ...
                        list_dump_ovr(frame07,dump07,40, rate4_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/100: %d\n", nrate4dump);
printf("\nWrite dump overrun tables.\n");

tc07_dump_overrun_tables
```

```
# printf("\nWrite dump to dump difference tables.\n");
# tc07_dump2dump_tables

# printf("\nWrite chain difference tables.\n");
# tc07_chain_diff_tables  # under construction
printf("***--------------------------------------------------------------------------***\n");
```

3.8.2 tc07_dump_overrun_tables

```
## make dump overrun tables

CS="|";

alltsout= [tsout1;tsout2;tsout3;tsout4];
allcellout=[cellout1;cellout2;cellout3;cellout4];
allOVRout= [OVRout1;OVRout2;OVRout3;OVRout4];

[ss,is]=sort(str2num(alltsout));

fdu2= fopen("tc07_dump_overrun-f2.txt","w"); # table format 2

fprintf(fdu2,"timestamp%s0   1   2   3       4   5   6   7       8   9   10   11      12   13   14   15\n",\
CS);

idt=1;
for id=1:ndump07
               if strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),allOVRout(is(idt++),:));
               elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:))&\
               !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))
                              fprintf(fdu2,"%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                              allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                                             allOVRout(is(idt++),:),allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif !strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:)) &\
                                             fprintf(fdu2,"%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:));
               endif
endfor

fclose(fdu2);
```

3.8.3 tc07_cell_bias_table

```
bias= -1*ones(ndump07,16);
fbiastable= fopen("tc07_all_cell_bias.txt","w");
for k= 1:ndump07
               nq= columns(dump07(k).CCchain);
               for ic=1:nq
                              bias(k,dump07(k).CCchain(ic).CCC+1)=\
                                             dump07(k).CCchain(ic).data_bias/64;
               endfor
               fprintf(fbiastable,"%d %s %d %s %d %s %d %s %d %s %d %s %d %s %d %s %d %s %d %s %d %s %d %s %d %s %d
%s %d %s %d %s
%d\n",dump07(k).TIMESTAMP_0,CS,bias(k,1),CS,bias(k,2),CS,bias(k,3),CS,bias(k,4),CS,bias(k,5),CS,bias(k,6),\
               CS,bias(k,7),CS,bias(k,8),CS,bias(k,9),CS,bias(k,10),CS,bias(k,11),CS,bias(k,12),CS,bias(k,13),CS,\
               bias(k,14),CS,bias(k,15),CS,bias(k,16));
endfor
fclose(fbiastable);
```

3.9 overrun detection and recovery

3.9.1 tc24_postsim

```
rintf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc24_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc24_apr23-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 1;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;1;1;1;1];
test_mcb.Y_SE_CLK= [0;0;0;0;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
applied_dump=[8000, 8000, 0, 0, 0, 0, 0, 0];
tc= 24;

[frame24, num_frames, desc24, bad_frames, dontcare_frames]= get_lag_frames("lag_frames24.txt");
printf("%d frames read for test case 24.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc24,"# ver: v");
printf("test case version: %s\n", desc24((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb24 = get_mcb_data("mcb_reg24.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain24, config24, unchain]= make_mcb_chains(mcb24);

## print chip configuration
nchains24= columns(Cchain24);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains24);
for ic=1:4:nchains24
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain24(ic).Xinput,Cchain24(ic).CCC(1),\
               Cchain24(ic).CCC(end),Cchain24(ic).Yinput,\
               Cchain24(ic+1).Xinput,Cchain24(ic+1).CCC(1),\
               Cchain24(ic+1).CCC(end),Cchain24(ic+1).Yinput,\
               Cchain24(ic+2).Xinput,Cchain24(ic+2).CCC(1),\
               Cchain24(ic+2).CCC(end),Cchain24(ic+2).Yinput,\
               Cchain24(ic+3).Xinput,Cchain24(ic+3).CCC(1),\
               Cchain24(ic+3).CCC(end),Cchain24(ic+3).Yinput);
endfor

printf("\n");
```

```
[dump24, ndump24, ibadstamp] = make_dump_index(frame24);
if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump24, with_chains24] = add_CCchain_dump(dump24, frame24);


               if !(ndump24 == rows(with_chains24))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp24=zeros(ndump24,1);
               for id=1:ndump24; timestamp24(id)= dump24(id).TIMESTAMP_0; endfor;
               if timestamp24(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp24(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp24(1));
               endif
               if all(diff(timestamp24) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump24,mcb24,Cchain24,applied_dump,"tc24",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb24.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

printf("\nNumber of dumps: %d\n",ndump24);

rate1_cells= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # cells dumping at the highest rate

## get dump overrun information from the output frames
[out, nrateNdump, tsout, cellout, OVRout, iodump, icell, icell_OVR]= ...
                                             list_dump_ovr(frame24,dump24,1,rate1_cells);
printf("Number of dumps of cells dumping at the highest rate: %d\n", nrateNdump);

printf("\nDiscarding isolated (unpaired) cells from each dump.\n");
printf("\nConcatenate chains and scale data regardless of OVR bit status.\n");
for its=1:nrateNdump   # for all dumps

               # combine cell index vectors of non-OVR and OVR cases
               allcells_index= icell(its,:);
               OVR_index= find(icell_OVR(its,:));
               allcells_index(OVR_index)= icell_OVR(its,OVR_index);

               # find incomplete pairs, that is, pairs in which one or more cells are absent in this dump
               missing_pairs_index= find((allcells_index(1:2:15) == 0) | (allcells_index(2:2:16) == 0));

               # set to zero, indices of cells that are part of a missing pair
               for j=1:columns(missing_pairs_index)
                              allcells_index((2*missing_pairs_index(j)-1):(2*missing_pairs_index(j)))= 0;
               endfor

               # extract the non-zero (complete pair cells) indices
        dumped_pairs_index= allcells_index(find(allcells_index));

               dump(its).index= dump24(its).index(dumped_pairs_index);
```

```
                # NOTE: dump is overwritten, orphans cells discarded, hence
                # icell is no longer relevent to locating a particular chain within dump.

endfor

printf("Cells with mixed OVR status bits within a chain will have\n");
printf("the data bias applied independently to each cell.\n");
printf("A single DVCOUNT will be applied to the chain.\n\n");

printf("\nWrite dump overrun tables.\n");
tc24_dump_overrun_tables

# printf("\nWrite dump to dump difference tables.\n");
# tc24_dump2dump_tables

# printf("\nWrite chain difference tables.\n");
# tc24_chain_diff_tables  # under construction
printf("***--------------------------------------------------------------------------***\n");
```

3.9.2 tc24_chain_diff_tables

```
## make tables of lag difference between like chains the same dump
## based on examination of the dump overrun tables


CS="|";

fcd_filename= "tc24_chain_diff.txt";
printf("Write table to file: %s.\n", fcd_filename);
fcd= fopen(fcd_filename,"w");

fprintf(fcd,"chain 1%schain 2%stimestamp%sOVR ch1%sOVR ch2%sre mean%sim mean%sre std%sim std\n",...
CS,CS,CS,CS,CS,CS,CS,CS);

id= [1;1;1;1;  2;2; 3;3; 4;4; 5;5;5; 6;6; 7;7; 8;8; 9;9; 10;10];
ic1=[1;2;3;4;  3;4; 1;2; 1;2; 1;2;3; 1;2; 1;4; 2;3; 1;2;  1;2];
ic2=[5;6;7;8;  5;6; 3;4; 3;6; 5;6;7; 5;6; 5;6; 4;5; 3;4;  5;6];


for ik= 1:rows(id)
               fprintf(fcd,"%d:%d%s%d:%d%s%d%s%d:%d%s%d:%d%s%3.4f%s%3.4f%s%3.4f%s%3.4f\n",
               dump24(id(ik)).CCchain(ic1(ik)).CCC(1),dump24(id(ik)).CCchain(ic1(ik)).CCC(2),CS,...
               dump24(id(ik)).CCchain(ic2(ik)).CCC(1),dump24(id(ik)).CCchain(ic2(ik)).CCC(2),CS,...
               dump24(id(ik)).TIMESTAMP_0,CS,...
               dump24(id(ik)).CCchain(ic1(ik)).OVR(1),dump24(id(ik)).CCchain(ic1(ik)).OVR(2),CS,...
               dump24(id(ik)).CCchain(ic2(ik)).OVR(1),dump24(id(ik)).CCchain(ic2(ik)).OVR(2),CS,...
               real(mean(dump24(id(ik)).CCchain(ic1(ik)).clag-dump24(id(ik)).CCchain(ic2(ik)).clag)),CS,...
               imag(mean(dump24(id(ik)).CCchain(ic1(ik)).clag-dump24(id(ik)).CCchain(ic2(ik)).clag)),CS,...
               std(real(dump24(id(ik)).CCchain(ic1(ik)).clag-dump24(id(ik)).CCchain(ic2(ik)).clag)),CS,...
               std(imag(dump24(id(ik)).CCchain(ic1(ik)).clag-dump24(id(ik)).CCchain(ic2(ik)).clag)));

endfor

fclose(fcd);
```

3.9.3 tc24_dump_overrun_tables

```
## make dump overrun tables

CS="|";

fdu1= fopen("tc24_dump_overrun-f1.txt","w"); # table format 1
fdu2= fopen("tc24_dump_overrun-f2.txt","w"); # table format 2

fprintf(fdu1,"timestamp%s cells dump request and overrun bit settings\n",CS);

for id=1:nrateNdump
               fprintf(fdu1,"%s%s%s\n",tsout(id,:),CS,cellout(id,:));
               fprintf(fdu1,"%s  %s%s\n","overrun",CS,OVRout(id,:));
               fprintf(fdu1,"%s%s%s\n","         ",CS,"  ");
endfor

fclose(fdu1);

fprintf(fdu2,"timestamp%s%s\n",CS, cellout(1,:));

for id=1:nrateNdump
               fprintf(fdu2,"%s%s%s\n",tsout(id,:),CS,OVRout(id,:));
endfor
```

```
fclose(fdu2);
```

3.9.4 tc29_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc29_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc29_apr11-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 1;  # dump overrun bit is set
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
applied_dump=[3000, 0, 6000, 0, 12000, 0, 24000, 0];
tc= 29;

[frame29, num_frames, desc29, bad_frames, dontcare_frames]= get_lag_frames("lag_frames29.txt");
printf("%d frames read for test case 29.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc29,"# ver: v");
printf("test case version: %s\n", desc29((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb29 = get_mcb_data("mcb_reg29.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain29, config29, unchain]= make_mcb_chains(mcb29);

## print chip configuration
nchains29= columns(Cchain29);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains29);
for ic=1:4:nchains29
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain29(ic).Xinput,Cchain29(ic).CCC(1),\
               Cchain29(ic).CCC(end),Cchain29(ic).Yinput,\
               Cchain29(ic+1).Xinput,Cchain29(ic+1).CCC(1),\
               Cchain29(ic+1).CCC(end),Cchain29(ic+1).Yinput,\
               Cchain29(ic+2).Xinput,Cchain29(ic+2).CCC(1),\
               Cchain29(ic+2).CCC(end),Cchain29(ic+2).Yinput,\
               Cchain29(ic+3).Xinput,Cchain29(ic+3).CCC(1),\
               Cchain29(ic+3).CCC(end),Cchain29(ic+3).Yinput);
endfor
```

```
printf("\n");

[dump29, ndump29, ibadstamp] = make_dump_index(frame29);
if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump29, with_chains29] = add_CCchain_dump(dump29, frame29);


               if !(ndump29 == rows(with_chains29))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               timestamp29=zeros(ndump29,1);
               for id=1:ndump29; timestamp29(id)= dump29(id).TIMESTAMP_0; endfor;
               if timestamp29(1) < 0
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp29(1)+2^32);
               else
                              printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp29(1));
               endif
               if all(diff(timestamp29) == 1)
                              printf("Timestamps are sequential and increment by 1.\n");
               else
                              printf("Missing timestamps (not increment by 1).\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump29,mcb29,Cchain29,applied_dump,"tc29",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb29.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

printf("\nNumber of dumps: %d\n",ndump29);

rate1_cells= [0;1;2;3];
rate2_cells= [4;5;6;7];
rate3_cells= [8;9;10;11];
rate4_cells= [12;13;14;15];

## get dump overrun information from the output frames
## the first dump does not occur; supply timestamp of initial dump
## Assure that version 1.10 or greater of list_dump_ovr is available!
name_version_ldo= list_dump_ovr;
printf("\nUsing list_dump_ovr: %s\n",name_version_ldo);
tstart= 222333444;

[out1, nrate1dump, tsout1, cellout1, OVRout1, idump1, icell1, icell_OVR1]= ...
                                             list_dump_ovr(frame29,dump29,1, rate1_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate: %d\n", nrate1dump);

[out2, nrate2dump, tsout2, cellout2, OVRout2, idump2, icell2, icell_OVR2]= ...
                                             list_dump_ovr(frame29,dump29,2, rate2_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/2: %d\n", nrate2dump);

[out3, nrate2dump, tsout3, cellout3, OVRout3, idump3, icell3, icell_OVR3]= ...
                                             list_dump_ovr(frame29,dump29,4, rate3_cells,tstart);

printf("Requested number of dumps of cells dumping at the highest rate/4: %d\n", nrate2dump);

[out4, nrate4dump, tsout4, cellout4, OVRout4, idump4, icell, icell_OVR4]= ...
                                             list_dump_ovr(frame29,dump29,8, rate4_cells,tstart);
```

```
printf("Requested number of dumps of cells dumping at the highest rate/8: %d\n", nrate4dump);
printf("\nWrite dump overrun tables.\n");

tc29_dump_overrun_tables

# printf("\nWrite dump to dump difference tables.\n");
# tc29_dump2dump_tables

# printf("\nWrite chain difference tables.\n");
# tc29_chain_diff_tables  # under construction
printf("***--------------------------------------------------------------------------***\n");
```

3.9.5 tc29_dump_overrun_tables

```
## make dump overrun tables

CS="|";

alltsout= [tsout1;tsout2;tsout3;tsout4];
allcellout=[cellout1;cellout2;cellout3;cellout4];
allOVRout= [OVRout1;OVRout2;OVRout3;OVRout4];

[ss,is]=sort(str2num(alltsout));

fdu2= fopen("tc29_dump_overrun-f2.txt","w"); # table format 2

fprintf(fdu2,"timestamp%s0   1   2   3       4   5   6   7       8   9   10   11      12   13   14   15\n",\
CS);

idt=1;
for id=1:ndump29
               if strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),allOVRout(is(idt++),:));
               elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:))&\
               !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))
                              fprintf(fdu2,"%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                              allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                                           allOVRout(is(idt++),:),allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif !strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:)) &\
                                           fprintf(fdu2,"%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:));
               endif
endfor

fclose(fdu2);
```

3.9.6 tc29 chain_similarity_check

```
Lnz= !ones(ndump29,1);  # flag any non-zero difference

id=1;
for ic=1:3
               Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=2;
for ic=1:3
               Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor
for ic=5:7
               Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=3;
for ic=1:3
               Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=4;
for ic=1:3
               Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor
for ic=5:7
```

```
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

ic=5;
for ic=1:3
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=6;
for ic=1:3
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=7;
for ic=1:2
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=8;
for ic=2:4
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor
for ic=6:8
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor
for ic=10:12
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=9;
for ic=1:2
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id= 10;
for ic=1:3
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id= 11;
for ic=1:2
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor
for ic=3:5
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=13;
for ic=2:4
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=14;
for ic=1:2
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

id=15;
for ic=2:3
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor
for ic=4:6
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor
for ic=8:10
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor
for ic=12:14
                Lnz(id)= any(abs(dump29(id).CCchain(ic).clag-dump29(id).CCchain(ic+1).clag) >= 10*eps);
endfor

if !any(Lnz)
                printf("\nNo differences exceeding the floating point precision limit of this\n\
machine were found between chains for any of the %d dumps; all chains generated\n\
identical lag data.\n\n", ndump29);
else
                printf("\nWARNING: Differences exceeding the floating point precision limit\n\
                  of this machine were found between chains.\n\n");
```

3.9.7 tc31_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc31_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc31_apr23-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 1;  # dump overrun bit is set
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all reset

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
applied_dump=[4000,4000,8000,8000,16000,16000,32000,32000];
tc= 31;

[frame31, num_frames, desc31, bad_frames, dontcare_frames]= get_lag_frames("lag_frames31.txt");
printf("%d frames read for test case 31.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc31,"# ver: v");
printf("test case version: %s\n", desc31((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb31 = get_mcb_data("mcb_reg31.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain31, config31, unchain]= make_mcb_chains(mcb31);

## print chip configuration
nchains31= columns(Cchain31);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains31);
for ic=1:4:nchains31
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain31(ic).Xinput,Cchain31(ic).CCC(1),\
               Cchain31(ic).CCC(end),Cchain31(ic).Yinput,\
               Cchain31(ic+1).Xinput,Cchain31(ic+1).CCC(1),\
               Cchain31(ic+1).CCC(end),Cchain31(ic+1).Yinput,\
               Cchain31(ic+2).Xinput,Cchain31(ic+2).CCC(1),\
               Cchain31(ic+2).CCC(end),Cchain31(ic+2).Yinput,\
               Cchain31(ic+3).Xinput,Cchain31(ic+3).CCC(1),\
               Cchain31(ic+3).CCC(end),Cchain31(ic+3).Yinput);
endfor
printf("\n");

[dump31, ndump31, ibadstamp] = make_dump_index(frame31);
if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps
```

```
                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump31, with_chains31] = add_CCchain_dump(dump31, frame31);


                if !(ndump31 == rows(with_chains31))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp31=zeros(ndump31,1);
                for id=1:ndump31; timestamp31(id)= dump31(id).TIMESTAMP_0; endfor;
                if timestamp31(1) < 0
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp31(1)+2^32);
                else
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp31(1));
                endif
                if all(diff(timestamp31) == 1)
                                printf("Timestamps are sequential and increment by 1.\n");
                else
                                printf("Missing timestamps (not increment by 1).\n");
                endif

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump31,mcb31,Cchain31,applied_dump,"tc31",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb31.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

printf("\nNumber of dumps: %d\n",ndump31);

rate1_cells= [0;1;2;3];
rate2_cells= [4;5;6;7];
rate3_cells= [8;9;10;11];
rate4_cells= [12;13;14;15];

## get dump overrun information from the output frames
## the first dump does not occur; supply timestamp of initial dump
## Assure that version 1.10 or greater of list_dump_ovr is available!
name_version_ldo= list_dump_ovr;
printf("\nUsing list_dump_ovr: %s\n",name_version_ldo);
tstart= 222333444;

[out1, nrate1dump, tsout1, cellout1, OVRout1, idump1, icell1, icell_OVR1]= ...
                                        list_dump_ovr(frame31,dump31,1, rate1_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate: %d\n", nrate1dump);

[out2, nrate2dump, tsout2, cellout2, OVRout2, idump2, icell2, icell_OVR2]= ...
                                        list_dump_ovr(frame31,dump31,2, rate2_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/2: %d\n", nrate2dump);

[out3, nrate2dump, tsout3, cellout3, OVRout3, idump3, icell3, icell_OVR3]= ...
                                        list_dump_ovr(frame31,dump31,4, rate3_cells,tstart);

printf("Requested number of dumps of cells dumping at the highest rate/4: %d\n", nrate2dump);

[out4, nrate4dump, tsout4, cellout4, OVRout4, idump4, icell, icell_OVR4]= ...
                                        list_dump_ovr(frame31,dump31,8, rate4_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/8: %d\n", nrate4dump);
printf("\nWrite dump overrun tables.\n");

tc31_dump_overrun_tables
```

```
# printf("\nWrite dump to dump difference tables.\n");
# tc31_dump2dump_tables

# printf("\nWrite chain difference tables.\n");
# tc31_chain_diff_tables  # under construction
printf("***--------------------------------------------------------------------------***\n");
```

3.9.8 tc31_dump_overrun_tables

```
## make dump overrun tables

CS="|";

alltsout= [tsout1;tsout2;tsout3;tsout4];
allcellout=[cellout1;cellout2;cellout3;cellout4];
allOVRout= [OVRout1;OVRout2;OVRout3;OVRout4];

[ss,is]=sort(str2num(alltsout));

fdu2= fopen("tc31_dump_overrun-f2.txt","w"); # table format 2

fprintf(fdu2,"timestamp%s0   1   2   3       4   5   6   7       8   9   10   11      12   13   14   15\n",\
CS);

idt=1;
for id=1:ndump31
               if strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),allOVRout(is(idt++),:));
               elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:))&\
               !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))
                              fprintf(fdu2,"%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                              allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                                           allOVRout(is(idt++),:),allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif !strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:)) &\
                                           fprintf(fdu2,"%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:));
               endif
endfor

fclose(fdu2);
```

3.10 on-line detection and reporting of errors

3.10.1 tc13_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc13_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc13_apr22-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;1;1;1;1];
test_mcb.Y_SE_CLK= [0;0;0;0;1;1;1;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 13;
applied_dump= [10000, 0, 0, 0, 0, 0, 0, 0];

## ----------------------------------------------------------------------------------------
[frame13,num_frames,desc13,bad_frames,dontcare_frames]= get_lag_frames("lag_frames13.txt");
printf("%d frames read for test case 13.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc13,"# ver: v");
printf("test case version: %s\n", desc13((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb13 = get_mcb_data("mcb_reg13.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain13, config13, unchain]= make_mcb_chains(mcb13);


## print chip configuration
nchains13= columns(Cchain13);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains13);
for ic=1:4:nchains13
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain13(ic).Xinput,Cchain13(ic).CCC(1),\
               Cchain13(ic).CCC(end),Cchain13(ic).Yinput,\
               Cchain13(ic+1).Xinput,Cchain13(ic+1).CCC(1),\
               Cchain13(ic+1).CCC(end),Cchain13(ic+1).Yinput,\
               Cchain13(ic+2).Xinput,Cchain13(ic+2).CCC(1),\
               Cchain13(ic+2).CCC(end),Cchain13(ic+2).Yinput,\
```

```
              Cchain13(ic+3).Xinput,Cchain13(ic+3).CCC(1),\
              Cchain13(ic+3).CCC(end),Cchain13(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump13, ndump13, ibadstamp] = make_dump_index(frame13);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

              # form contatenated chains for each dump on the basis of the output frame
              # status bits and add these to the dump structure.
              [dump13, with_chains13] = add_CCchain_dump(dump13, frame13);

              if !(ndump13 == rows(with_chains13))
                            printf("\nADVISORY: NOT ALL dumps contain chains!\n");
              endif
              timestamp13(1)= dump13(1).TIMESTAMP_0;
              printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp13(1));

              # check consistency of STATUS and MCSR register states; data_bias and data captured
              # from data and dump control streams. Check consistency of mcb contents with those
              # of the structure test_mcb.
              [Lwarn, Ladvis] = check_register_capture(dump13,mcb13,Cchain13,applied_dump,"tc13",tc_SID_X,\
              tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

              if !Lwarn & !Ladvis
                            printf("\nCheck OK: no inconsistencies reported in captured data\n");
                            printf("          or register contents; no unexpected register reports;\n");
                            printf("          details in log file.\n");
              else
                            if Lwarn
                                          printf("\nWARNINGS encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
                            if Ladvis
                                          printf("\nADVISORIES encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
              endif

              # Check that DESSR bits are all set as expected.
              if all(mcb13.DESSR == test_DESSR)
                            printf("\nCheck OK: DESSR bits are set as expected.\n");
              else
                            printf("\nWARNING: DESSR bits are set NOT as expected.\n");
              endif
endif
printf("***--------------------------------------------------------------------------***\n");
```

3.10.2 tc13_list_dumped_chains

```
ndumped_chains= columns(dump13(1).CCchain);
for ich=1:ndumped_chains
              printf("(%d:%d)  ",dump13(1).CCchain(ich).CCC(1),dump13(1).CCchain(ich).CCC(end));
endfor
printf("\n");
```

3.10.3 tc13_chain_similarity_check

```
Lnz= !ones(16,ndump13);  # flag any non-zero difference
for id=1:ndump13
              for ic=1:7
                            Lnz(ic,id)= any(abs(dump13(id).CCchain(ic).clag-\
                                          dump13(id).CCchain(ic+1).clag) >= 10*eps);
              endfor
endfor

if !any(any(Lnz))
              printf("\nNo differences exceeding the floating point precision limit of this\n\
machine were found between chains for any of the %d dumps; all chains generated\n\
identical lag data.\n\n", ndump13);
else
              printf("\nWARNING: Differences exceeding the floating point precision limit\n\
                of this machine were found between chains.\n\n");
endif
```

3.10.4 tc27_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc27_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
              printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc27_apr23-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;1;0;1;0;1;0;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 27;
applied_dump= [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000];

## ----------------------------------------------------------------------------------------
[frame27,num_frames,desc27,bad_frames,dontcare_frames]= get_lag_frames("lag_frames27.txt");
printf("%d frames read for test case 27.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc27,"# ver: v");
printf("test case version: %s\n", desc27((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
              printf("Checksum is OK for all frames and corrupt data were not\n");
              printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb27 = get_mcb_data("mcb_reg27.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain27, config27, unchain]= make_mcb_chains(mcb27);


## print chip configuration
nchains27= columns(Cchain27);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains27);
for ic=1:4:nchains27
              printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
              Cchain27(ic).Xinput,Cchain27(ic).CCC(1),\
              Cchain27(ic).CCC(end),Cchain27(ic).Yinput,\
              Cchain27(ic+1).Xinput,Cchain27(ic+1).CCC(1),\
              Cchain27(ic+1).CCC(end),Cchain27(ic+1).Yinput,\
              Cchain27(ic+2).Xinput,Cchain27(ic+2).CCC(1),\
              Cchain27(ic+2).CCC(end),Cchain27(ic+2).Yinput,\
```

```
              Cchain27(ic+3).Xinput,Cchain27(ic+3).CCC(1),\
              Cchain27(ic+3).CCC(end),Cchain27(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump27, ndump27, ibadstamp] = make_dump_index(frame27);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

              # form contatenated chains for each dump on the basis of the output frame
              # status bits and add these to the dump structure.
              [dump27, with_chains27] = add_CCchain_dump(dump27, frame27);

              if !(ndump27 == rows(with_chains27))
                            printf("\nADVISORY: NOT ALL dumps contain chains!\n");
              endif
              timestamp27(1)= dump27(1).TIMESTAMP_0;
              printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp27(1));

              # check consistency of STATUS and MCSR register states; data_bias and data captured
              # from data and dump control streams. Check consistency of mcb contents with those
              # of the structure test_mcb.
              [Lwarn, Ladvis] = check_register_capture(dump27,mcb27,Cchain27,applied_dump,"tc27",tc_SID_X,\
              tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

              if !Lwarn & !Ladvis
                            printf("\nCheck OK: no inconsistencies reported in captured data\n");
                            printf("          or register contents; no unexpected register reports;\n");
                            printf("          details in log file.\n");
              else
                            if Lwarn
                                          printf("\nWARNINGS encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
                            if Ladvis
                                          printf("\nADVISORIES encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
              endif

              # Check that DESSR bits are all set as expected.
              if all(mcb27.DESSR == test_DESSR)
                            printf("\nCheck OK: DESSR bits are set as expected.\n");
              else
                            printf("\nWARNING: DESSR bits are set NOT as expected.\n");
              endif
endif
printf("***--------------------------------------------------------------------------***\n");
```

3.10.5 tc27_list_dumped_chains

```
ndumped_chains= columns(dump27(1).CCchain);
for ich=1:ndumped_chains
              printf("(%d:%d)  ",dump27(1).CCchain(ich).CCC(1),dump27(1).CCchain(ich).CCC(end));
endfor
printf("\n");
```

3.10.6 tc27_find_NaN

```
for ic= 1:columns(dump27(1).CCchain)
              if all(isnan(dump27(1).CCchain(ic).clag))
                            printf("chain (%d:%d) all
NaN\n",dump27(1).CCchain(ic).CCC(1),dump27(1).CCchain(ic).CCC(end));
              elseif any(isnan(dump27(1).CCchain(ic).clag))
                            printf("chain (%d:%d) contains
NaN\n",dump27(1).CCchain(ic).CCC(1),dump27(1).CCchain(ic).CCC(end));
              else
                            printf("chain (%d:%d) contain lag
data\n",dump27(1).CCchain(ic).CCC(1),dump27(1).CCchain(ic).CCC(end));
              endif
endfor
```

3.10.7 tc35_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc35_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc35_apr23-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;0;1;0;1;0;1];
test_mcb.Y_SE_CLK= [0;1;0;1;0;1;0;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 35;
applied_dump= [10000, 10000, 0, 0, 0, 0, 0, 0];

## ----------------------------------------------------------------------------------------
[frame35,num_frames,desc35,bad_frames,dontcare_frames]= get_lag_frames("lag_frames35.txt");
printf("%d frames read for test case 35.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc35,"# ver: v");
printf("test case version: %s\n", desc35((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb35 = get_mcb_data("mcb_reg35.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain35, config35, unchain]= make_mcb_chains(mcb35);


## print chip configuration
nchains35= columns(Cchain35);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains35);
for ic=1:4:nchains35
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain35(ic).Xinput,Cchain35(ic).CCC(1),\
               Cchain35(ic).CCC(end),Cchain35(ic).Yinput,\
               Cchain35(ic+1).Xinput,Cchain35(ic+1).CCC(1),\
               Cchain35(ic+1).CCC(end),Cchain35(ic+1).Yinput,\
               Cchain35(ic+2).Xinput,Cchain35(ic+2).CCC(1),\
               Cchain35(ic+2).CCC(end),Cchain35(ic+2).Yinput,\
               Cchain35(ic+3).Xinput,Cchain35(ic+3).CCC(1),\
               Cchain35(ic+3).CCC(end),Cchain35(ic+3).Yinput);
```

```
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump35, ndump35, ibadstamp] = make_dump_index(frame35);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump35, with_chains35] = add_CCchain_dump(dump35, frame35);

               if !(ndump35 == rows(with_chains35))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp35(1)= dump35(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp35(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump35,mcb35,Cchain35,applied_dump,"tc35",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb35.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif
printf("***--------------------------------------------------------------------------***\n");
```

3.10.8 tc35_list_dumped_chains

```
ndumped_chains= columns(dump35(1).CCchain);
for ich=1:ndumped_chains
               printf("(%d:%d)  ",dump35(1).CCchain(ich).CCC(1),dump35(1).CCchain(ich).CCC(end));
endfor
printf("\n");
```

3.10.9 tc35_find_NaN

```
for ic= 1:columns(dump35(1).CCchain)
               if all(isnan(dump35(1).CCchain(ic).clag))
                              printf("chain (%d:%d) all
NaN\n",dump35(1).CCchain(ic).CCC(1),dump35(1).CCchain(ic).CCC(end));
               elseif any(isnan(dump35(1).CCchain(ic).clag))
                              printf("chain (%d:%d) contains
NaN\n",dump35(1).CCchain(ic).CCC(1),dump35(1).CCchain(ic).CCC(end));
               else
                              printf("chain (%d:%d) contain lag
data\n",dump35(1).CCchain(ic).CCC(1),dump35(1).CCchain(ic).CCC(end));
               endif
endfor
```

3.10.10 tc36_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc36_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc36_apr23-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;1;0;1;0;1;0;1];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 1;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 36;
applied_dump= [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000];

## ----------------------------------------------------------------------------------------
[frame36,num_frames,desc36,bad_frames,dontcare_frames]= get_lag_frames("lag_frames36.txt");
printf("%d frames read for test case 36.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc36,"# ver: v");
printf("test case version: %s\n", desc36((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb36 = get_mcb_data("mcb_reg36.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain36, config36, unchain]= make_mcb_chains(mcb36);


## print chip configuration
nchains36= columns(Cchain36);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains36);
for ic=1:4:nchains36
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain36(ic).Xinput,Cchain36(ic).CCC(1),\
               Cchain36(ic).CCC(end),Cchain36(ic).Yinput,\
               Cchain36(ic+1).Xinput,Cchain36(ic+1).CCC(1),\
               Cchain36(ic+1).CCC(end),Cchain36(ic+1).Yinput,\
               Cchain36(ic+2).Xinput,Cchain36(ic+2).CCC(1),\
               Cchain36(ic+2).CCC(end),Cchain36(ic+2).Yinput,\
               Cchain36(ic+3).Xinput,Cchain36(ic+3).CCC(1),\
               Cchain36(ic+3).CCC(end),Cchain36(ic+3).Yinput);
endfor
```

```
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump36, ndump36, ibadstamp] = make_dump_index(frame36);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump36, with_chains36] = add_CCchain_dump(dump36, frame36);

               if !(ndump36 == rows(with_chains36))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp36(1)= dump36(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp36(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump36,mcb36,Cchain36,applied_dump,"tc36",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb36.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif
printf("***--------------------------------------------------------------------------***\n");
```

3.10.11 tc50_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc50_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc50_apr24-05_v1.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
```

```
test_mcb.X_DATA= [1;1;1;1;1;1;1;1];
test_mcb.Y_DATA= [1;1;1;1;1;1;1;1];
test_mcb.X_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.Y_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];  # even bits set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 136;
tc_SID_Y= 68;

tc= 50;
applied_dump= [5000, 5000, 5000,5000, 5000, 5000, 5000, 5000];

## ----------------------------------------------------------------------------------
[frame50,num_frames,desc50,bad_frames,dontcare_frames]= get_lag_frames("lag_frames50.txt");
printf("%d frames read for test case 50.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc50,"# ver: v");
printf("test case version: %s\n", desc50((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb50 = get_mcb_data("mcb_reg50.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain50, config50, unchain]= make_mcb_chains(mcb50);

## print chip configuration
nchains50= columns(Cchain50);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains50);
for ic=1:4:nchains50
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain50(ic).Xinput,Cchain50(ic).CCC(1),\
               Cchain50(ic).CCC(end),Cchain50(ic).Yinput,\
               Cchain50(ic+1).Xinput,Cchain50(ic+1).CCC(1),\
               Cchain50(ic+1).CCC(end),Cchain50(ic+1).Yinput,\
               Cchain50(ic+2).Xinput,Cchain50(ic+2).CCC(1),\
               Cchain50(ic+2).CCC(end),Cchain50(ic+2).Yinput,\
               Cchain50(ic+3).Xinput,Cchain50(ic+3).CCC(1),\
               Cchain50(ic+3).CCC(end),Cchain50(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump50, ndump50, ibadstamp] = make_dump_index(frame50);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump50, with_chains50] = add_CCchain_dump(dump50, frame50);

               if !(ndump50 == rows(with_chains50))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp50(1)= dump50(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp50(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump50,mcb50,Cchain50,applied_dump,"tc50",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
```

```
                              if Ladvis
                                        printf("\nADVISORIES encountered during register status and data capture
                                                  checks;\n\
                                        details in log file.\n");
                              endif
                    endif

                    # Check that DESSR bits are all set as expected.
                    if all(mcb50.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
                    else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                    endif
endif

printf("***--------------------------------------------------------------------------***\n");
```

3.10.12 tc50_check_captured_data

```
## Check captured data with  reset lines: test cases 50

SBID_X_ERROR= [0;0;0;0;2;2;2;2;0;0;0;0;2;2;2;2];
SBID_Y_ERROR= [1;0;1;0;1;0;1;0;5;4;5;4;5;4;5;4];
SID_X_ERROR= 136;
SID_Y_ERROR= 68;

Lcapee= !1;
for ic=1:16
                    if (dump50(1).CCchain(ic).SBID_X != SBID_X_ERROR(ic) || dump50(1).CCchain(ic).SBID_Y !=
SBID_Y_ERROR(ic)\
                    || dump50(1).CCchain(ic).SID_X != SID_X_ERROR || dump50(1).CCchain(ic).SID_Y != SID_Y_ERROR)
                              printf("\nWARNING: Unexpected data captured on streams with selected\n\
data lines reset. Check cell %d.\n",ic);
                              Lcapee= !0;
                    endif
endfor
if !Lcapee
                    printf("\nCheck OK: Expected data are captured on streams with reset data lines.\n");
endif
```

3.10.13 tc50_check_status_bits

```
sbt= [0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1];
if any(any(mcb50.X_DATA != sbt)) || any(any(mcb50.X_PHASE != [sbt])) || \
any(any(mcb50.Y_DATA != [!sbt])) || any(any(mcb50.Y_PHASE != [!sbt]))
                    printf("\n WARNING: Unexpected X|Y-DATA or X|Y-PHASE STATUS register\n\
bit settings.\n");
else
                              printf("\n Check OK: X|Y-DATA and X|Y-PHASE STATUS register bits are set\n\
as expected.\n");
endif
```

3.10.14 tc51_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc51_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
                    printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc51_apr24-05_v1.v
```

```
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [1;1;1;1;1;1;1;1];
test_mcb.Y_DATA= [1;1;1;1;1;1;1;1];
test_mcb.X_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.Y_PHASE= [1;1;1;1;1;1;1;1];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];  # even bits set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 64;
tc_SID_Y= 32;
tc=51;

applied_dump= [5000, 5000, 5000,5000, 5000, 5000, 5000, 5000];

## ----------------------------------------------------------------------------------------
[frame51,num_frames,desc51,bad_frames,dontcare_frames]= get_lag_frames("lag_frames51.txt");
printf("%d frames read for test case 51.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc51,"# ver: v");
printf("test case version: %s\n", desc51((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb51 = get_mcb_data("mcb_reg51.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain51, config51, unchain]= make_mcb_chains(mcb51);


## print chip configuration
nchains51= columns(Cchain51);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains51);
for ic=1:4:nchains51
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain51(ic).Xinput,Cchain51(ic).CCC(1),\
               Cchain51(ic).CCC(end),Cchain51(ic).Yinput,\
               Cchain51(ic+1).Xinput,Cchain51(ic+1).CCC(1),\
               Cchain51(ic+1).CCC(end),Cchain51(ic+1).Yinput,\
               Cchain51(ic+2).Xinput,Cchain51(ic+2).CCC(1),\
               Cchain51(ic+2).CCC(end),Cchain51(ic+2).Yinput,\
               Cchain51(ic+3).Xinput,Cchain51(ic+3).CCC(1),\
               Cchain51(ic+3).CCC(end),Cchain51(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump51, ndump51, ibadstamp] = make_dump_index(frame51);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump51, with_chains51] = add_CCchain_dump(dump51, frame51);

               if !(ndump51 == rows(with_chains51))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp51(1)= dump51(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp51(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
```

```
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump51,mcb51,Cchain51,applied_dump,"tc51",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb51.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

printf("***--------------------------------------------------------------------------***\n");
```

3.10.15 tc51_check_captured_data

```
## Check captured data with  reset lines: test cases 50

SBID_X_ERROR= [1;1;0;0;1;1;0;0;5;5;4;4;5;5;4;4];
SBID_Y_ERROR= [0;0;0;0;2;2;2;2;0;0;0;0;2;2;2;2];
SID_X_ERROR= 64;
SID_Y_ERROR= 32;

Lcapee= !1;
for ic=1:16
                if (dump51(1).CCchain(ic).SBID_X != SBID_X_ERROR(ic) || dump51(1).CCchain(ic).SBID_Y !=
SBID_Y_ERROR(ic)\
                || dump51(1).CCchain(ic).SID_X != SID_X_ERROR || dump51(1).CCchain(ic).SID_Y != SID_Y_ERROR)
                                printf("\nWARNING: Unexpected data captured on streams with selected\n\
data lines reset. Check cell %d.\n",ic);
                                Lcapee= !0;
                endif
endfor
if !Lcapee
                printf("\nCheck OK: Expected data are captured on streams with reset data lines.\n");
endif
```

3.10.16 tc51_check_status_bits

```
sbt= ![0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1;0,1,0,1];
if any(any(mcb51.X_DATA != sbt)) || any(any(mcb51.X_PHASE != [sbt])) || \
any(any(mcb51.Y_DATA != [!sbt])) || any(any(mcb51.Y_PHASE != [!sbt]))
                printf("\n WARNING: Unexpected X|Y-DATA or X|Y-PHASE STATUS register\n\
bit settings.\n");
else
                                printf("\n Check OK: X|Y-DATA and X|Y-PHASE STATUS register bits are set\n\
as expected.\n");
endif
```

3.10.17 tc69_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.
```

```
name_version= "tc69_postsim.m : V1.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc69_dec09-05_v1.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 1;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 69;
applied_dump= [1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000];

## ----------------------------------------------------------------------------------------
[frame69,num_frames,desc69,bad_frames,dontcare_frames]= get_lag_frames("lag_frames69.txt");
printf("%d frames read for test case 69.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc69,"# ver: v");
printf("test case version: %s\n", desc69((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb69 = get_mcb_data("mcb_reg69.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain69, config69, unchain]= make_mcb_chains(mcb69);


## print chip configuration
nchains69= columns(Cchain69);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains69);
for ic=1:4:nchains69
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain69(ic).Xinput,Cchain69(ic).CCC(1),\
               Cchain69(ic).CCC(end),Cchain69(ic).Yinput,\
               Cchain69(ic+1).Xinput,Cchain69(ic+1).CCC(1),\
               Cchain69(ic+1).CCC(end),Cchain69(ic+1).Yinput,\
               Cchain69(ic+2).Xinput,Cchain69(ic+2).CCC(1),\
               Cchain69(ic+2).CCC(end),Cchain69(ic+2).Yinput,\
               Cchain69(ic+3).Xinput,Cchain69(ic+3).CCC(1),\
               Cchain69(ic+3).CCC(end),Cchain69(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump69, ndump69, ibadstamp] = make_dump_index(frame69);


if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps
```

```
                printf("NOTE: Accumulator rollover at centre lag of chain 7 fixed by adding 2^23.\n");
                # fix the real part of acc. overflow for the centre lag of the last chain
                frame69(dump69(ndump69).index(11)).clag(65)=frame69(dump69(ndump69).index(11)).clag(65)+2^23;
                frame69(dump69(ndump69).index(12)).clag(65)=frame69(dump69(ndump69).index(12)).clag(65)+2^23;

                # fix the imaginary part of acc. overflow for the off centre lag of the last chain
                frame69(dump69(ndump69).index(11)).clag(64)=frame69(dump69(ndump69).index(11)).clag(64)+...
                (2^23)*1i;
                frame69(dump69(ndump69).index(12)).clag(64)=frame69(dump69(ndump69).index(12)).clag(64)+...
                (2^23)*1i;

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump69, with_chains69] = add_CCchain_dump(dump69, frame69);

                if !(ndump69 == rows(with_chains69))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp69=zeros(ndump69,1);
                for id=1:ndump69; timestamp69(id)= dump69(id).TIMESTAMP_0; endfor;
                if timestamp69(1) < 0
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp69(1)+2^32);
                else
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp69(1));
                endif
                if all(diff(timestamp69) == 1)
                                printf("Timestamps are sequential and increment by 1.\n");
                else
                                printf("Missing timestamps (not increment by 1).\n");
                endif

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump69,mcb69,Cchain69,applied_dump,"tc69",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb69.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

[bmod69] = get_bmodel_chains("test70", Cchain69, 16);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc69";
lag_peak_thold= 3;
fd_peak_thold= 10;

# generate model difference statistics excluding chains 1 through 6 (dump overruns)
icstart= 7;
icstop= 16;

printf("Generating tables of model lag and frequency domain\n\
difference statistics for chains %d through %d; using name: %s\n", icstart, icstop, tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [0.8333;0.8333;0.1667;0.1667;0.3333;0.3333;0.6667;0.6667;1.3333;1.3333;2.667;2.667;...
5.333;5.333;10.667;10.667]; # (number of samples per dump) / 12000
```

```
dmod=  [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # SE_CLK modulusm (1 -> always set)
make_stat_tables(Cchain69,dump69,bmod69,tname,icstart,icstop,dmult,dmod,\
lag_peak_thold,fd_peak_thold,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.10.18 tc69_make_dump_ovr_list

```
printf("\nNumber of dumps: %d\n",ndump69);

rate1_cells= [0;1];
rate2_cells= [2;3];
rate3_cells= [4;5];
rate4_cells= [6;7];
rate5_cells= [8;9];
rate6_cells= [10;11];
rate7_cells= [12;13];
rate8_cells= [14;15];

## get dump overrun information from the output frames
## the first dump does not occur; supply timestamp of initial dump
## Assure that version 1.10 or greater of list_dump_ovr is available!
name_version_ldo= list_dump_ovr;
printf("\nUsing list_dump_ovr: %s\n",name_version_ldo);
tstart= 222333444;

[out1, nrate1dump, tsout1, cellout1, OVRout1, idump1, icell1, icell_OVR1]= ...
                                        list_dump_ovr(frame69,dump69,1, rate1_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate: %d\n", nrate1dump);

[out2, nrate2dump, tsout2, cellout2, OVRout2, idump2, icell2, icell_OVR2]= ...
                                       list_dump_ovr(frame69,dump69,2, rate2_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/2: %d\n", nrate2dump);

[out3, nrate2dump, tsout3, cellout3, OVRout3, idump3, icell3, icell_OVR3]= ...
                                       list_dump_ovr(frame69,dump69,4, rate3_cells,tstart);

printf("Requested number of dumps of cells dumping at the highest rate/4: %d\n", nrate2dump);

[out4, nrate4dump, tsout4, cellout4, OVRout4, idump4, icell4, icell_OVR4]= ...
                                       list_dump_ovr(frame69,dump69,8, rate4_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/8: %d\n", nrate4dump);

%% [out5, nrate5dump, tsout5, cellout5, OVRout5, idump5, icell5, icell_OVR5]= ...
%%                                       list_dump_ovr(frame69,dump69,16, rate5_cells,tstart);
%% printf("Requested number of dumps of cells dumping at the highest rate/16: %d\n", nrate5dump);
%%
%% [out6, nrate6dump, tsout6, cellout6, OVRout6, idump6, icell6, icell_OVR6]= ...
%%                                       list_dump_ovr(frame69,dump69,32, rate6_cells,tstart);
%% printf("Requested number of dumps of cells dumping at the highest rate/32: %d\n", nrate6dump);
%%
%% [out7, nrate7dump, tsout7, cellout7, OVRout7, idump7, icell7, icell_OVR7]= ...
%%                                       list_dump_ovr(frame69,dump69,64, rate7_cells,tstart);
%% printf("Requested number of dumps of cells dumping at the highest rate/64: %d\n", nrate7dump);
%%
%% [out8, nrate8dump, tsout8, cellout8, OVRout8, idump8, icell8, icell_OVR8]= ...
%%                                       list_dump_ovr(frame69,dump69,128, rate8_cells,tstart);
%% printf("Requested number of dumps of cells dumping at the highest rate/128: %d\n", nrate8dump);
%% printf("\nWrite dump overrun tables.\n");
```

3.10.19 tc69_dump_overrun_tables

```
## make dump overrun tables

CS="|";

alltsout= [tsout1;tsout2;tsout3;tsout4];
allcellout=[cellout1;cellout2;cellout3;cellout4];
allOVRout= [OVRout1;OVRout2;OVRout3;OVRout4];

[ss,is]=sort(str2num(alltsout));

fdu2= fopen("tc69_dump_overrun-f2.txt","w"); # table format 2

fprintf(fdu2,"timestamp%s0   1       2   3       4   5       6  7\n",CS);


idt=1;
for id=1:ndump69
               if strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
```

```
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),allOVRout(is(idt++),:));
               elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:))&\
               !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))
                              fprintf(fdu2,"%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                              allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                                        allOVRout(is(idt++),:),allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif !strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:)) &\
                                        fprintf(fdu2,"%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:));
               endif
endfor

fclose(fdu2);
```

3.10.20 tc69_chain_similarity_check

```
Lnz= !ones(16,ndump69)  # flag any non-zero difference
printf("\n");
for id=1:ndump69
               for ic=1:2:columns(dump69(id).CCchain)
                              if (any(ic==[0;2;4;6] && any(ic==[1;3;5;7]) && ...
                                 (dump69(id).CCchain(ic).OVR == dump69(id).CCchain(ic+1).OVR)
                                        Lnz(ic,id)=any(abs(dump69(id).CCchain(ic).clag-
dump(id).CCchain(ic+1).clag) > eps)
                              endif
               endfor
endfor
if !any(any(Lnz))
               printf("\nNOTICE: No differences exceeding the floating point precision limit of\n\
               this machine were found between cells having similar OVR bit states and identical
               identical streams for any of the %d dumps.\n\n", ndump69);
else
               printf("\nWARNING: Differences exceeding the floating point precision limit\n\
                 of this machine were found between cells having similar OVR bit states and
                 identical streams. Check dump %d, chain %d.\n\n", id, ic);
endif
```

3.10.21 tc69_ACC_OV_scan

```
Lnacc= !ones(16,ndump69);  # flag any set ACC_OV bits
printf("\n");
for id=1:ndump69
               for ic=1:(columns(dump69(id).CCchain))
                              if (dump69(id).CCchain(ic).ACC_OV)
                                        printf("Bit ACC_OV set in ouput frame of cell %d, dump %d.\n",\
                                        dump69(id).CCchain(ic).CCC, id);
                                        Lnacc(ic,id)=!0;
                              endif
               endfor
endfor
if !any(any(Lnacc))
               printf("\nNOTICE: Accumulator overflow bit ACC_OV was not set\n\
               in any output frame.\n");
endif
printf("\n");
```

3.10.22 tc70_postsim

```
printf("***------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.
```

```
name_version= "tc70_postsim.m : V1.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc70_dec09-05_v1.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 1;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 70;
applied_dump= [1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000];

## ----------------------------------------------------------------------------------------
[frame70,num_frames,desc70,bad_frames,dontcare_frames]= get_lag_frames("lag_frames70.txt");
printf("%d frames read for test case 70.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc70,"# ver: v");
printf("test case version: %s\n", desc70((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb70 = get_mcb_data("mcb_reg70.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain70, config70, unchain]= make_mcb_chains(mcb70);


## print chip configuration
nchains70= columns(Cchain70);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains70);
for ic=1:4:nchains70
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain70(ic).Xinput,Cchain70(ic).CCC(1),\
               Cchain70(ic).CCC(end),Cchain70(ic).Yinput,\
               Cchain70(ic+1).Xinput,Cchain70(ic+1).CCC(1),\
               Cchain70(ic+1).CCC(end),Cchain70(ic+1).Yinput,\
               Cchain70(ic+2).Xinput,Cchain70(ic+2).CCC(1),\
               Cchain70(ic+2).CCC(end),Cchain70(ic+2).Yinput,\
               Cchain70(ic+3).Xinput,Cchain70(ic+3).CCC(1),\
               Cchain70(ic+3).CCC(end),Cchain70(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump70, ndump70, ibadstamp] = make_dump_index(frame70);


if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps
```

```
                printf("NOTE: Accumulator rollover at centre lag of chain 7 fixed by adding 2^23.\n");
                # fix the real part of acc. overflow for the centre lag of the last chain
                frame70(dump70(ndump70).index(11)).clag(65)=frame70(dump70(ndump70).index(11)).clag(65)+2^23;
                frame70(dump70(ndump70).index(12)).clag(65)=frame70(dump70(ndump70).index(12)).clag(65)+2^23;

                # fix the imaginary part of acc. overflow for the off centre lag of the last chain
                frame70(dump70(ndump70).index(11)).clag(64)=frame70(dump70(ndump70).index(11)).clag(64)+...
                (2^23)*1i;
                frame70(dump70(ndump70).index(12)).clag(64)=frame70(dump70(ndump70).index(12)).clag(64)+...
                (2^23)*1i;

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump70, with_chains70] = add_CCchain_dump(dump70, frame70);

                if !(ndump70 == rows(with_chains70))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp70=zeros(ndump70,1);
                for id=1:ndump70; timestamp70(id)= dump70(id).TIMESTAMP_0; endfor;
                if timestamp70(1) < 0
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp70(1)+2^32);
                else
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp70(1));
                endif
                if all(diff(timestamp70) == 1)
                                printf("Timestamps are sequential and increment by 1.\n");
                else
                                printf("Missing timestamps (not increment by 1).\n");
                endif

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump70,mcb70,Cchain70,applied_dump,"tc70",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb70.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

[bmod70] = get_bmodel_chains("test70", Cchain70, 16);
name_version_mdc= model_diff_chains;

printf("\nUsing model_diff_chains: %s\n",name_version_mdc);
name_version_mst= make_stat_tables;
printf("Using make_stat_tables: %s\n",name_version_mst);

tname= "tc70";
lag_peak_thold= 3;
fd_peak_thold= 10;

# generate model difference statistics excluding chains 1 through 6 (dump overruns)
icstart= 7;
icstop= 16;

printf("Generating tables of model lag and frequency domain\n\
difference statistics for chains %d through %d; using name: %s\n", icstart, icstop, tname);

printf("Using peak detection threshold of %4.2f std deviation\n\
 over lags and %4.2f std. dev. over the frequency domain\n\
 for a given chain.\n\n", lag_peak_thold, fd_peak_thold);
dmult= [0.8333;0.8333;0.1667;0.1667;0.3333;0.3333;0.6667;0.6667;1.3333;1.3333;2.667;2.667;...
5.333;5.333;10.667;10.667]; # (number of samples per dump) / 12000
dmod=  [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]; # SE_CLK modulusm (1 -> always set)
```

```
make_stat_tables(Cchain70,dump70,bmod70,tname,icstart,icstop,dmult,dmod,\
lag_peak_thold,fd_peak_thold,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.10.23 tc70_make_dump_ovr_list

```
printf("\nNumber of dumps: %d\n",ndump70);

rate1_cells= [0;1];
rate2_cells= [2;3];
rate3_cells= [4;5];
rate4_cells= [6;7];
rate5_cells= [8;9];
rate6_cells= [10;11];
rate7_cells= [12;13];
rate8_cells= [14;15];

## get dump overrun information from the output frames
## the first dump does not occur; supply timestamp of initial dump
## Assure that version 1.10 or greater of list_dump_ovr is available!
name_version_ldo= list_dump_ovr;
printf("\nUsing list_dump_ovr: %s\n",name_version_ldo);
tstart= -1;

[out1, nrate1dump, tsout1, cellout1, OVRout1, idump1, icell1, icell_OVR1]= ...
                                        list_dump_ovr(frame70,dump70,1, rate1_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate: %d\n", nrate1dump);

[out2, nrate2dump, tsout2, cellout2, OVRout2, idump2, icell2, icell_OVR2]= ...
                                        list_dump_ovr(frame70,dump70,2, rate2_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/2: %d\n", nrate2dump);

[out3, nrate2dump, tsout3, cellout3, OVRout3, idump3, icell3, icell_OVR3]= ...
                                        list_dump_ovr(frame70,dump70,4, rate3_cells,tstart);

printf("Requested number of dumps of cells dumping at the highest rate/4: %d\n", nrate2dump);

[out4, nrate4dump, tsout4, cellout4, OVRout4, idump4, icell4, icell_OVR4]= ...
                                        list_dump_ovr(frame70,dump70,8, rate4_cells,tstart);
printf("Requested number of dumps of cells dumping at the highest rate/8: %d\n", nrate4dump);

%% [out5, nrate5dump, tsout5, cellout5, OVRout5, idump5, icell5, icell_OVR5]= ...
%%                                         list_dump_ovr(frame70,dump70,16, rate5_cells,tstart);
%% printf("Requested number of dumps of cells dumping at the highest rate/16: %d\n", nrate5dump);
%%
%% [out6, nrate6dump, tsout6, cellout6, OVRout6, idump6, icell6, icell_OVR6]= ...
%%                                         list_dump_ovr(frame70,dump70,32, rate6_cells,tstart);
%% printf("Requested number of dumps of cells dumping at the highest rate/32: %d\n", nrate6dump);
%%
%% [out7, nrate7dump, tsout7, cellout7, OVRout7, idump7, icell7, icell_OVR7]= ...
%%                                         list_dump_ovr(frame70,dump70,64, rate7_cells,tstart);
%% printf("Requested number of dumps of cells dumping at the highest rate/64: %d\n", nrate7dump);
%%
%% [out8, nrate8dump, tsout8, cellout8, OVRout8, idump8, icell8, icell_OVR8]= ...
%%                                         list_dump_ovr(frame70,dump70,128, rate8_cells,tstart);
%% printf("Requested number of dumps of cells dumping at the highest rate/128: %d\n", nrate8dump);
%% printf("\nWrite dump overrun tables.\n");
```

3.10.24 tc70_dump_overrun_tables

```
## make dump overrun tables

CS="|";

alltsout= [tsout1;tsout2;tsout3;tsout4];
allcellout=[cellout1;cellout2;cellout3;cellout4];
allOVRout= [OVRout1;OVRout2;OVRout3;OVRout4];

nallsort= str2num(alltsout);
nallsort(1)= -1;

[ss,is]=sort(nallsort);

fdu2= fopen("tc70_dump_overrun-f2.txt","w"); # table format 2

fprintf(fdu2,"timestamp%s0   1       2   3       4   5       6  7\n",CS);
```

```
idt=1;
for id=1:ndump70
               if strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),allOVRout(is(idt++),:));
               elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:))&\
               !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))
                              fprintf(fdu2,"%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                              allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:))

fprintf(fdu2,"%s%s%s%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:),\
                                             allOVRout(is(idt++),:),allOVRout(is(idt++),:),allOVRout(is(idt++),:));
                              elseif !strcmp(alltsout(is(idt),:),alltsout(is(idt+1),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+2),:)) &\
                              !strcmp(alltsout(is(idt),:),alltsout(is(idt+3),:)) &\
                                             fprintf(fdu2,"%s%s%s\n",alltsout(is(idt),:),CS,allOVRout(is(idt++),:));
               endif
endfor

fclose(fdu2);
```

3.10.25 tc70_chain_similarity_check

```
Lnz= !ones(16,ndump70);  # flag any non-zero difference
printf("\n");
for id=1:ndump70
               for ic=1:(columns(dump70(id).CCchain)-1)
                              if (any(dump70(id).CCchain(ic).CCC==[0;2;4;6;8;10;12;14] &...
                               dump70(id).CCchain(ic+1).CCC==[1;3;5;7;9;11;13;15])) && ...
                                 (dump70(id).CCchain(ic).OVR == dump70(id).CCchain(ic+1).OVR)
                                             Lnz(ic,id)=any(abs(dump70(id).CCchain(ic).clag-...
                                             dump70(id).CCchain(ic+1).clag) > eps);
                              endif
#                             printf("\ndump %d, cell %d\n",id,dump70(id).CCchain(ic).CCC);
               endfor
endfor
if !any(any(Lnz))
               printf("\nNOTICE: No differences exceeding the floating point precision \n\
               limit of this machine were found between cells having similar \n\
               OVR bit states and identical streams for any of the %d dumps.\n\n", ndump70);
else
               printf("\nWARNING: Differences exceeding the floating point precision \n\
                 limit of this machine were found between cells having \n\
                 similar OVR bit states and identical streams.\n\n");
endif
```

3.10.26 tc70_ACC_OV_scan

```
Lnacc= !ones(16,ndump70);  # flag any set ACC_OV bits
printf("\n");
for id=1:ndump70
               for ic=1:(columns(dump70(id).CCchain))
                              if (dump70(id).CCchain(ic).ACC_OV)
                                             printf("Bit ACC_OV set in ouput frame of cell %d, dump %d.\n",\
                                             dump70(id).CCchain(ic).CCC, id);
                                             Lnacc(ic,id)=!0;
                              endif
               endfor
endfor
if !any(any(Lnacc))
               printf("\nNOTICE: Accumulator overflow bit ACC_OV was not set\n\
               in any output frame.\n");
endif
printf("\n");
```

3.10.27 tc69_70_chain_compare

```
Lnz= !ones(16,ndump70);  # flag any non-zero difference
printf("\n");
for id=1:ndump70
               for ic=1:(columns(dump70(id).CCchain)-1)
                              if (any(dump70(id).CCchain(ic).CCC==[0;2;4;6;8;10;12;14] &...
                               dump70(id).CCchain(ic+1).CCC==[1;3;5;7;9;11;13;15])) && ...
                                 (dump70(id).CCchain(ic).OVR == dump70(id).CCchain(ic+1).OVR)
                                             Lnz(ic,id)=any(abs(dump70(id).CCchain(ic).clag-...
                                             dump69(id).CCchain(ic).clag) > eps);
                              endif
#                             printf("\ndump %d, cell %d\n",id,dump70(id).CCchain(ic).CCC);
               endfor
endfor
if !any(any(Lnz))
               printf("\nNOTICE: No differences exceeding the floating point precision \n\
               limit of this machine were found between identically \n\
               configured cells of test cases 69 and 70 having similar \n\
               OVR bit states. \n\n");
else
               printf("\nWARNING: Differences exceeding the floating point precision \n\
                 limit of this machine were found between identically \n\
                 configured cells of test cases 69 and 70 having similar \n\
                 OVR bit states. \n\n");
endif
```

3.11 X-Y dump enable offset detection

3.11.1 tc48_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc48_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc48_aug07-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1];  # even bits set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 85;
tc_SID_Y= 170;
tc= 48;
applied_dump= [500, 1000, 0, 0, 0, 0, 0, 0];
```

```
## ----------------------------------------------------------------------------
[frame48,num_frames,desc48,bad_frames,dontcare_frames]= get_lag_frames("lag_frames48.txt");
printf("%d frames read for test case 48.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc48,"# ver: v");
printf("test case version: %s\n", desc48((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb48 = get_mcb_data("mcb_reg48.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain48, config48, unchain]= make_mcb_chains(mcb48);


printf("\nInput configuration from MCSR regsister settings:\n");
if columns(config48) > 72
               printf("%s\n", config48(1:72));
               printf("%s\n", config48(73:end));
else
               printf("%s\n", config48);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump48, ndump48, ibadstamp] = make_dump_index(frame48);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump48, with_chains48] = add_CCchain_dump(dump48, frame48);

               if !(ndump48 == rows(with_chains48))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump48,mcb48,Cchain48,applied_dump,"tc48",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb48.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

printf("***--------------------------------------------------------------------------***\n");
```

3.11.2 tc48_captured_table

```
CS="|";
fdump1=fopen("tc48_captured_table.txt","w");

for id=1:2;
               fprintf(fdump1,"dump %d %s cell number %s",id,CS,CS);
```

```
                for ic=1:columns(dump48(id).CCchain)
                                fprintf(fdump1,"(%d) %s ",dump48(id).CCchain(ic).CCC(1),CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s Phase bin %s",id,CS,CS);
                for ic=1:columns(dump48(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump48(id).CCchain(ic).Phase_bin,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s cmmd %s",id,CS,CS);
                for ic=1:columns(dump48(id).CCchain)
                                fprintf(fdump1,"%s%s%s %s ",dump48(id).CCchain(ic).cmmd,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s SBID_X %s",id,CS,CS);
                for ic=1:columns(dump48(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump48(id).CCchain(ic).SBID_X,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s SBID_Y %s",id,CS,CS);
                for ic=1:columns(dump48(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump48(id).CCchain(ic).SBID_Y,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s BBID_X %s",id,CS,CS);
                for ic=1:columns(dump48(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump48(id).CCchain(ic).BBID_X,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s BBID_Y %s",id,CS,CS);
                for ic=1:columns(dump48(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump48(id).CCchain(ic).BBID_Y,CS);
                endfor
                fprintf(fdump1,"\n");
endfor
fclose(fdump1);
```

3.11.3 tc48_data_ID_capture

```
CS="|";
fdd=fopen("tc48_data_ID_capture_table.txt","w");
fprintf(fdd,"dump %s chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS,CS);
for id=1:2
                for ic=1:8
                                fprintf(fdd,"%d %s %d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n",id,CS,\
                                Cchain48(ic).Xinput,dump48(id).CCchain(ic).CCC(1),dump48(id).CCchain(ic).CCC(1),\

Cchain48(ic).Yinput,CS,dump48(id).CCchain(ic).SID_X(1),CS,dump48(id).CCchain(ic).SID_Y(1),CS,\
                                dump48(id).CCchain(ic).SBID_X(1),CS,dump48(id).CCchain(ic).SBID_Y(1),CS,\
                                dump48(id).CCchain(ic).BBID_X(1),CS,dump48(id).CCchain(ic).BBID_Y(1));
                endfor
endfor
fclose(fdd);
```

3.11.4 tc49_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc49_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
                printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc49_aug07-05_v3.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
```

```
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0];  # even bits set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 85;
tc_SID_Y= 170;
tc= 49;
applied_dump= [500, 1000, 0, 0, 0, 0, 0, 0];

## --------------------------------------------------------------------------------
[frame49,num_frames,desc49,bad_frames,dontcare_frames]= get_lag_frames("lag_frames49.txt");
printf("%d frames read for test case 49.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc49,"# ver: v");
printf("test case version: %s\n", desc49((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb49 = get_mcb_data("mcb_reg49.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain49, config49, unchain]= make_mcb_chains(mcb49);


printf("\nInput configuration from MCSR regsister settings:\n");
if columns(config49) > 72
               printf("%s\n", config49(1:72));
               printf("%s\n", config49(73:end));
else
               printf("%s\n", config49);
endif
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump49, ndump49, ibadstamp] = make_dump_index(frame49);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump49, with_chains49] = add_CCchain_dump(dump49, frame49);

               if !(ndump49 == rows(with_chains49))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump49,mcb49,Cchain49,applied_dump,"tc49",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif
```

```
                # Check that DESSR bits are all set as expected.
                if all(mcb49.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

printf("***-------------------------------------------------------------------------***\n");
```

3.11.5 tc49_captured_table

```
CS="|";
fdump1=fopen("tc49_captured_table.txt","w");

for id=1:2;
                fprintf(fdump1,"dump %d %s cell number %s",id,CS,CS);
                for ic=1:columns(dump49(id).CCchain)
                                fprintf(fdump1,"(%d) %s ",dump49(id).CCchain(ic).CCC(1),CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s Phase bin %s",id,CS,CS);
                for ic=1:columns(dump49(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump49(id).CCchain(ic).Phase_bin,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s cmmd %s",id,CS,CS);
                for ic=1:columns(dump49(id).CCchain)
                                fprintf(fdump1,"%s%s%s %s ",dump49(id).CCchain(ic).cmmd,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s SBID_X %s",id,CS,CS);
                for ic=1:columns(dump49(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump49(id).CCchain(ic).SBID_X,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s SBID_Y %s",id,CS,CS);
                for ic=1:columns(dump49(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump49(id).CCchain(ic).SBID_Y,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s BBID_X %s",id,CS,CS);
                for ic=1:columns(dump49(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump49(id).CCchain(ic).BBID_X,CS);
                endfor;
                fprintf(fdump1,"\ndump %d %s BBID_Y %s",id,CS,CS);
                for ic=1:columns(dump49(id).CCchain)
                                fprintf(fdump1,"%d %s ",dump49(id).CCchain(ic).BBID_Y,CS);
                endfor
                fprintf(fdump1,"\n");
endfor
fclose(fdump1);
```

3.11.6 tc49_data_ID_capture

```
CS="|";
fdd=fopen("tc49_data_ID_capture_table.txt","w");
fprintf(fdd,"dump %s chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS,CS);
for id=1:2
                for ic=1:8
                                fprintf(fdd,"%d %s %d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n",id,CS,\
                                Cchain49(ic).Xinput,dump49(id).CCchain(ic).CCC(1),dump49(id).CCchain(ic).CCC(1),\

Cchain49(ic).Yinput,CS,dump49(id).CCchain(ic).SID_X(1),CS,dump49(id).CCchain(ic).SID_Y(1),CS,\
                                dump49(id).CCchain(ic).SBID_X(1),CS,dump49(id).CCchain(ic).SBID_Y(1),CS,\
                                dump49(id).CCchain(ic).BBID_X(1),CS,dump49(id).CCchain(ic).BBID_Y(1));
                endfor
endfor
fclose(fdd);
```

3.12 off-line detection and reporting of errors

Post simulation scripts or functions are not required.

3.13 MCSR status flags

Post simulation scripts or functions are not required.

3.14 correlation hold off

3.14.1 tc42_postsim

```
printf("***------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc42_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc42_apr26-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
tc=42;
applied_dump= [10000, 10000,10000, 10000, 10000, 10000, 10000, 10000];

## ----------------------------------------------------------------------------------------
[frame42,num_frames,desc42,bad_frames,dontcare_frames]= get_lag_frames("lag_frames42.txt");
printf("%d frames read for test case 42.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc42,"# ver: v");
printf("test case version: %s\n", desc42((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb42 = get_mcb_data("mcb_reg42.txt");
```

```
## Form chains and input configuration based on the contents of the MCB register file.
[Cchain42, config42, unchain]= make_mcb_chains(mcb42);


## print chip configuration
nchains42= columns(Cchain42);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains42);
for ic=1:4:nchains42
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain42(ic).Xinput,Cchain42(ic).CCC(1),\
               Cchain42(ic).CCC(end),Cchain42(ic).Yinput,\
               Cchain42(ic+1).Xinput,Cchain42(ic+1).CCC(1),\
               Cchain42(ic+1).CCC(end),Cchain42(ic+1).Yinput,\
               Cchain42(ic+2).Xinput,Cchain42(ic+2).CCC(1),\
               Cchain42(ic+2).CCC(end),Cchain42(ic+2).Yinput,\
               Cchain42(ic+3).Xinput,Cchain42(ic+3).CCC(1),\
               Cchain42(ic+3).CCC(end),Cchain42(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump42, ndump42, ibadstamp] = make_dump_index(frame42);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump42, with_chains42] = add_CCchain_dump(dump42, frame42);

               if !(ndump42 == rows(with_chains42))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp42(1)= dump42(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp42(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump42,mcb42,Cchain42,applied_dump,"tc42",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb42.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

printf("\nWriting dump capture table.\n");
tc42_dump_capture

printf("Writing data ID capture table.\n");
tc42_data_ID_capture

printf("***--------------------------------------------------------------------------***\n");
```

3.15 frame abort

3.15.1 tc43_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc43_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc43_apr24-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;1;0;1;0;1;0;1];
test_mcb.Y_SE_CLK= [0;1;0;1;0;1;0;1];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 43;
applied_dump= [10000, 0, 10000, 0, 20000, 0, 40000, 0];

## ----------------------------------------------------------------------------------------
[frame43,num_frames,desc43,bad_frames,dontcare_frames]= get_lag_frames("lag_frames43x.txt");
printf("%d frames read for test case 43.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc43,"# ver: v");
printf("test case version: %s\n", desc43((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb43 = get_mcb_data("mcb_reg43.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain43, config43, unchain]= make_mcb_chains(mcb43);


## print chip configuration
nchains43= columns(Cchain43);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains43);
for ic=1:4:nchains43
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain43(ic).Xinput,Cchain43(ic).CCC(1),\
               Cchain43(ic).CCC(end),Cchain43(ic).Yinput,\
               Cchain43(ic+1).Xinput,Cchain43(ic+1).CCC(1),\
               Cchain43(ic+1).CCC(end),Cchain43(ic+1).Yinput,\
               Cchain43(ic+2).Xinput,Cchain43(ic+2).CCC(1),\
               Cchain43(ic+2).CCC(end),Cchain43(ic+2).Yinput,\
```

```
                Cchain43(ic+3).Xinput,Cchain43(ic+3).CCC(1),\
                Cchain43(ic+3).CCC(end),Cchain43(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump43, ndump43, ibadstamp] = make_dump_index(frame43);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump43, with_chains43] = add_CCchain_dump(dump43, frame43);

                if !(ndump43 == rows(with_chains43))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif

                timestamp43=zeros(ndump43,1);
                for id=1:ndump43; timestamp43(id)= dump43(id).TIMESTAMP_0; endfor;
                if timestamp43(1) < 0
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp43(1)+2^32);
                else
                                printf("\nInitial (dump= 1) timestamp (word 0): %d\n",timestamp43(1));
                endif
                if all(diff(timestamp43) == 1)
                                printf("Timestamps are sequential and increment by 1.\n");
                else
                                printf("Missing timestamps (not increment by 1).\n");
                endif

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump43,mcb43,Cchain43,applied_dump,"tc43",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb43.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

printf("***--------------------------------------------------------------------------***\n");
```

3.15.2 tc43_print_timestamps

```
for id=1:ndump43
                printf("%d: ", dump43(id).TIMESTAMP_0);
                for ic=1:columns(dump43(id).CCchain)
                                printf("%d(%d:%d)%d  ",Cchain43(ic).Xinput,\
                                dump43(id).CCchain(ic).CCC(1), dump43(id).CCchain(ic).CCC(end),\
                                Cchain43(ic).Yinput);
                endfor
                printf("\n");
endfor
```

3.16 dump command capture

3.16.1 tc41_postsim

```
printf("***------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc41_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc41_apr26-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 24;
tc_SID_X= 0;
tc_SID_Y= 255;
tc=41;
applied_dump= [10000, 10000,10000, 10000, 10000, 10000, 10000, 10000];

## ----------------------------------------------------------------------------------------
[frame41,num_frames,desc41,bad_frames,dontcare_frames]= get_lag_frames("lag_frames41.txt");
printf("%d frames read for test case 41.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc41,"# ver: v");
printf("test case version: %s\n", desc41((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb41 = get_mcb_data("mcb_reg41.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain41, config41, unchain]= make_mcb_chains(mcb41);


## print chip configuration
nchains41= columns(Cchain41);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains41);
for ic=1:4:nchains41
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain41(ic).Xinput,Cchain41(ic).CCC(1),\
               Cchain41(ic).CCC(end),Cchain41(ic).Yinput,\
               Cchain41(ic+1).Xinput,Cchain41(ic+1).CCC(1),\
               Cchain41(ic+1).CCC(end),Cchain41(ic+1).Yinput,\
               Cchain41(ic+2).Xinput,Cchain41(ic+2).CCC(1),\
               Cchain41(ic+2).CCC(end),Cchain41(ic+2).Yinput,\
               Cchain41(ic+3).Xinput,Cchain41(ic+3).CCC(1),\
```

```
              Cchain41(ic+3).CCC(end),Cchain41(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump41, ndump41, ibadstamp] = make_dump_index(frame41);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

              # form contatenated chains for each dump on the basis of the output frame
              # status bits and add these to the dump structure.
              [dump41, with_chains41] = add_CCchain_dump(dump41, frame41);

              if !(ndump41 == rows(with_chains41))
                            printf("\nADVISORY: NOT ALL dumps contain chains!\n");
              endif
              timestamp41(1)= dump41(1).TIMESTAMP_0;
              printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp41(1));

              # check consistency of STATUS and MCSR register states; data_bias and data captured
              # from data and dump control streams. Check consistency of mcb contents with those
              # of the structure test_mcb.
              [Lwarn, Ladvis] = check_register_capture(dump41,mcb41,Cchain41,applied_dump,"tc41",tc_SID_X,\
              tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

              if !Lwarn & !Ladvis
                            printf("\nCheck OK: no inconsistencies reported in captured data\n");
                            printf("          or register contents; no unexpected register reports;\n");
                            printf("          details in log file.\n");
              else
                            if Lwarn
                                          printf("\nWARNINGS encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
                            if Ladvis
                                          printf("\nADVISORIES encountered during register status and data capture
                                                        checks;\n\
                                          details in log file.\n");
                            endif
              endif

              # Check that DESSR bits are all set as expected.
              if all(mcb41.DESSR == test_DESSR)
                            printf("\nCheck OK: DESSR bits are set as expected.\n");
              else
                            printf("\nWARNING: DESSR bits are set NOT as expected.\n");
              endif
endif

printf("\nWriting dump capture table.\n");
tc41_dump_capture

printf("Writing data ID capture table.\n");
tc41_data_ID_capture

printf("***--------------------------------------------------------------------------***\n");
```

3.16.2 tc41_dump_capture

```
CS="|";
fdc=fopen("tc41_dump_capture_table.txt","w");
fprintf(fdc,"chain %s cmmd %s Phase_bin %s RECIRC_BLK_X %s RECIRC_BLK_Y \n",CS,CS,CS,CS);
for ic=1:8
              fprintf(fdc,"%d(%d:%d)%d %s %s %s %d %s %d %s %d\n", \
              Cchain41(ic).Xinput,dump41(1).CCchain(ic).CCC(1),dump41(1).CCchain(ic).CCC(2),Cchain41(ic).Yinput,CS,\
              dump41(1).CCchain(ic).cmmd(1,:),CS,dump41(1).CCchain(ic).Phase_bin(1),CS,\
              dump41(1).CCchain(ic).RECIRC_BLK_X(1),CS,dump41(1).CCchain(ic).RECIRC_BLK_Y(1));
endfor
fclose(fdc);
```

3.16.3 tc42_dump_capture

```
CS="|";
fdc=fopen("tc42_dump_capture_table.txt","w");
fprintf(fdc,"chain %s cmmd %s Phase_bin %s RECIRC_BLK_X %s RECIRC_BLK_Y \n",CS,CS,CS,CS);
for ic=1:8
              fprintf(fdc,"%d(%d:%d)%d %s %s %s %d %s %d %s %d\n", \
```

```
              Cchain42(ic).Xinput,dump42(1).CCchain(ic).CCC(1),dump42(1).CCchain(ic).CCC(2),Cchain42(ic).Yinput,CS,\
              dump42(1).CCchain(ic).cmmd(1,:),CS,dump42(1).CCchain(ic).Phase_bin(1),CS,\
              dump42(1).CCchain(ic).RECIRC_BLK_X(1),CS,dump42(1).CCchain(ic).RECIRC_BLK_Y(1));
endfor
fclose(fdc);
```

3.16.4 tc46_postsim

```
rintf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc46_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
              printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc46_apr24-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;1;0;1;0;1;0;1];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;

tc= 46;
applied_dump= [10000, 10000,10000, 10000, 10000, 10000, 10000, 10000];

## ----------------------------------------------------------------------------------------
[frame46,num_frames,desc46,bad_frames,dontcare_frames]= get_lag_frames("lag_frames46.txt");
printf("%d frames read for test case 46.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc46,"# ver: v");
printf("test case version: %s\n", desc46((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
              printf("Checksum is OK for all frames and corrupt data were not\n");
              printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb46 = get_mcb_data("mcb_reg46.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain46, config46, unchain]= make_mcb_chains(mcb46);


## print chip configuration
nchains46= columns(Cchain46);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains46);
for ic=1:4:nchains46
```

```
                printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
                Cchain46(ic).Xinput,Cchain46(ic).CCC(1),\
                Cchain46(ic).CCC(end),Cchain46(ic).Yinput,\
                Cchain46(ic+1).Xinput,Cchain46(ic+1).CCC(1),\
                Cchain46(ic+1).CCC(end),Cchain46(ic+1).Yinput,\
                Cchain46(ic+2).Xinput,Cchain46(ic+2).CCC(1),\
                Cchain46(ic+2).CCC(end),Cchain46(ic+2).Yinput,\
                Cchain46(ic+3).Xinput,Cchain46(ic+3).CCC(1),\
                Cchain46(ic+3).CCC(end),Cchain46(ic+3).Yinput);
endfor

printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump46, ndump46, ibadstamp] = make_dump_index(frame46);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump46, with_chains46] = add_CCchain_dump(dump46, frame46);

                if !(ndump46 == rows(with_chains46))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif
                timestamp46(1)= dump46(1).TIMESTAMP_0;
                printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp46(1));

                # check consistency of STATUS and MCSR register states; data_bias and data captured
                # from data and dump control streams. Check consistency of mcb contents with those
                # of the structure test_mcb.
                [Lwarn, Ladvis] = check_register_capture(dump46,mcb46,Cchain46,applied_dump,"tc46",tc_SID_X,\
                tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

                if !Lwarn & !Ladvis
                                printf("\nCheck OK: no inconsistencies reported in captured data\n");
                                printf("          or register contents; no unexpected register reports;\n");
                                printf("          details in log file.\n");
                else
                                if Lwarn
                                                printf("\nWARNINGS encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                                if Ladvis
                                                printf("\nADVISORIES encountered during register status and data capture
                                                                checks;\n\
                                                details in log file.\n");
                                endif
                endif

                # Check that DESSR bits are all set as expected.
                if all(mcb46.DESSR == test_DESSR)
                                printf("\nCheck OK: DESSR bits are set as expected.\n");
                else
                                printf("\nWARNING: DESSR bits are set NOT as expected.\n");
                endif
endif

printf("\nWriting dump capture table.\n");
tc46_dump_capture

printf("Writing data ID capture table.\n");
tc46_data_ID_capture

printf("***--------------------------------------------------------------------------***\n");
```

3.16.5 tc46_dump_capture

```
CS="|";
fdc=fopen("tc46_dump_capture_table.txt","w");
fprintf(fdc,"chain %s cmmd %s Phase_bin %s RECIRC_BLK_X %s RECIRC_BLK_Y \n",CS,CS,CS,CS);
for ic=1:8
                fprintf(fdc,"%d(%d:%d)%d %s %s %s %d %s %d %s %d\n", \
                Cchain46(ic).Xinput,dump46(1).CCchain(ic).CCC(1),dump46(1).CCchain(ic).CCC(2),Cchain46(ic).Yinput,CS,\
                dump46(1).CCchain(ic).cmmd(1,:),CS,dump46(1).CCchain(ic).Phase_bin(1),CS,\
                dump46(1).CCchain(ic).RECIRC_BLK_X(1),CS,dump46(1).CCchain(ic).RECIRC_BLK_Y(1));
endfor
fclose(fdc);
```

3.16.6 tc47_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc47_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc47_apr24-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 1;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [1;0;1;0;1;0;1;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 200;
tc_SID_Y= 100;
tc=47;

applied_dump= [10000, 10000,10000, 10000, 10000, 10000, 10000, 10000];

## ----------------------------------------------------------------------------------------
[frame47,num_frames,desc47,bad_frames,dontcare_frames]= get_lag_frames("lag_frames47.txt");
printf("%d frames read for test case 47.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc47,"# ver: v");
printf("test case version: %s\n", desc47((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb47 = get_mcb_data("mcb_reg47.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain47, config47, unchain]= make_mcb_chains(mcb47);


## print chip configuration
nchains47= columns(Cchain47);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains47);
for ic=1:4:nchains47
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain47(ic).Xinput,Cchain47(ic).CCC(1),\
               Cchain47(ic).CCC(end),Cchain47(ic).Yinput,\
               Cchain47(ic+1).Xinput,Cchain47(ic+1).CCC(1),\
               Cchain47(ic+1).CCC(end),Cchain47(ic+1).Yinput,\
               Cchain47(ic+2).Xinput,Cchain47(ic+2).CCC(1),\
               Cchain47(ic+2).CCC(end),Cchain47(ic+2).Yinput,\
               Cchain47(ic+3).Xinput,Cchain47(ic+3).CCC(1),\
               Cchain47(ic+3).CCC(end),Cchain47(ic+3).Yinput);
endfor
```

```
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump47, ndump47, ibadstamp] = make_dump_index(frame47);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump47, with_chains47] = add_CCchain_dump(dump47, frame47);

               if !(ndump47 == rows(with_chains47))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp47(1)= dump47(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp47(1));

               # check consistency of STATUS and MCSR register states; data_bias and data captured
               # from data and dump control streams. Check consistency of mcb contents with those
               # of the structure test_mcb.
               [Lwarn, Ladvis] = check_register_capture(dump47,mcb47,Cchain47,applied_dump,"tc47",tc_SID_X,\
               tc_SID_Y, base_SBID_X,base_SBID_Y,tc,test_mcb);

               if !Lwarn & !Ladvis
                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
                              printf("          or register contents; no unexpected register reports;\n");
                              printf("          details in log file.\n");
               else
                              if Lwarn
                                             printf("\nWARNINGS encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
                              if Ladvis
                                             printf("\nADVISORIES encountered during register status and data capture
                                                            checks;\n\
                                             details in log file.\n");
                              endif
               endif

               # Check that DESSR bits are all set as expected.
               if all(mcb47.DESSR == test_DESSR)
                              printf("\nCheck OK: DESSR bits are set as expected.\n");
               else
                              printf("\nWARNING: DESSR bits are set NOT as expected.\n");
               endif
endif

printf("\nWriting dump capture table.\n");
tc47_dump_capture

printf("Writing data ID capture table.\n");
tc47_data_ID_capture

printf("***--------------------------------------------------------------------------***\n");
```

3.16.7 tc47_dump_capture

```
CS="|";
fdc=fopen("tc47_dump_capture_table.txt","w");
fprintf(fdc,"chain %s cmmd %s Phase_bin %s RECIRC_BLK_X %s RECIRC_BLK_Y \n",CS,CS,CS,CS);
for ic=1:8
               fprintf(fdc,"%d(%d:%d)%d %s %s %s %d %s %d %s %d\n", \
               Cchain47(ic).Xinput,dump47(1).CCchain(ic).CCC(1),dump47(1).CCchain(ic).CCC(2),Cchain47(ic).Yinput,CS,\
               dump47(1).CCchain(ic).cmmd(1,:),CS,dump47(1).CCchain(ic).Phase_bin(1),CS,\
               dump47(1).CCchain(ic).RECIRC_BLK_X(1),CS,dump47(1).CCchain(ic).RECIRC_BLK_Y(1));
endfor
fclose(fdc);
```

3.17 timestamp capture

3.17.1 tc17_timestamp_capture

```
CS="|";
fdd=fopen("tc17_timestamp_capture_table.txt","w");
fprintf(fdd,"dump %s TIMESTAMP word1 %s TIMESTAMP word0\n",CS,CS);
for id=1:ndump17
               fprintf(fdd,"%d %s %d (%s) %s %d (%s)\n",\
               id,CS,dump17(id).TIMESTAMP_1,dec2hex(dump17(id).TIMESTAMP_1),CS,\
               dump17(id).TIMESTAMP_0,dec2hex(dump17(id).TIMESTAMP_0));
endfor
fclose(fdd);
```

3.17.2 tc18_timestamp_capture

```
CS="|";
fdd=fopen("tc18_timestamp_capture_table.txt","w");
fprintf(fdd,"dump %s TIMESTAMP word1 %s TIMESTAMP word0\n",CS,CS);
for id=1:ndump18
               fprintf(fdd,"%d %s %d (%s) %s %d (%s)\n",\
               id,CS,dump18(id).TIMESTAMP_1,dec2hex(dump18(id).TIMESTAMP_1),CS,\
               dump18(id).TIMESTAMP_0,dec2hex(dump18(id).TIMESTAMP_0));
endfor
fclose(fdd);
```

3.17.3 tc19_timestamp_capture

```
CS="|";
fdd=fopen("tc19_timestamp_capture_table.txt","w");
fprintf(fdd,"dump %s TIMESTAMP word1 %s TIMESTAMP word0\n",CS,CS);
for id=1:ndump19
               fprintf(fdd,"%d %s %d (%s) %s %d (%s)\n",\
               id,CS,dump19(id).TIMESTAMP_1,dec2hex(dump19(id).TIMESTAMP_1),CS,\
               dump19(id).TIMESTAMP_0,dec2hex(dump19(id).TIMESTAMP_0));
endfor
fclose(fdd);
```

3.17.4 tc20_timestamp_capture

```
CS="|";
fdd=fopen("tc20_timestamp_capture_table.txt","w");
fprintf(fdd,"dump %s TIMESTAMP word1 %s TIMESTAMP word0\n",CS,CS);
for id=1:ndump20
               fprintf(fdd,"%d %s %d (%s) %s %d (%s)\n",\
               id,CS,dump20(id).TIMESTAMP_1,dec2hex(dump20(id).TIMESTAMP_1),CS,\
               dump20(id).TIMESTAMP_0,dec2hex(dump20(id).TIMESTAMP_0));
endfor
fclose(fdd);
```

3.18 data stream ID capture

3.18.1 tc17_data_ID_capture

```
CS="|";
fdd=fopen("tc17_data_ID_capture_table.txt","w");
fprintf(fdd,"chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS);
for ic=1:4
                fprintf(fdd,"%d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n", \
                Cchain17(ic).Xinput,dump17(1).CCchain(ic).CCC(1),dump17(1).CCchain(ic).CCC(2),Cchain17(ic).Yinput,CS,
                dump17(1).CCchain(ic).SID_X(1),CS,dump17(1).CCchain(ic).SID_Y(1),CS,\
                dump17(1).CCchain(ic).SBID_X(1),CS,dump17(1).CCchain(ic).SBID_Y(1),CS,\
                dump17(1).CCchain(ic).BBID_X(1),CS,dump17(1).CCchain(ic).BBID_Y(1));
endfor
fclose(fdd);
```

3.18.2 tc18_data_ID_capture

```
CS="|";
fdd=fopen("tc18_data_ID_capture_table.txt","w");
fprintf(fdd,"chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS);
for ic=1:4
                fprintf(fdd,"%d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n", \
                Cchain18(ic).Xinput,dump18(1).CCchain(ic).CCC(1),dump18(1).CCchain(ic).CCC(2),Cchain18(ic).Yinput,CS,
                dump18(1).CCchain(ic).SID_X(1),CS,dump18(1).CCchain(ic).SID_Y(1),CS,\
                dump18(1).CCchain(ic).SBID_X(1),CS,dump18(1).CCchain(ic).SBID_Y(1),CS,\
                dump18(1).CCchain(ic).BBID_X(1),CS,dump18(1).CCchain(ic).BBID_Y(1));
endfor
fclose(fdd);
```

3.18.3 tc19_data_ID_capture

```
CS="|";
fdd=fopen("tc19_data_ID_capture_table.txt","w");
fprintf(fdd,"chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS);
for ic=1:4
                fprintf(fdd,"%d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n", \
                Cchain19(ic).Xinput,dump19(1).CCchain(ic).CCC(1),dump19(1).CCchain(ic).CCC(2),Cchain19(ic).Yinput,CS,
                dump19(1).CCchain(ic).SID_X(1),CS,dump19(1).CCchain(ic).SID_Y(1),CS,\
                dump19(1).CCchain(ic).SBID_X(1),CS,dump19(1).CCchain(ic).SBID_Y(1),CS,\
                dump19(1).CCchain(ic).BBID_X(1),CS,dump19(1).CCchain(ic).BBID_Y(1));
endfor
fclose(fdd);
```

3.18.4 tc20_data_ID_capture

```
CS="|";
fdd=fopen("tc20_data_ID_capture_table.txt","w");
fprintf(fdd,"chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS);
for ic=1:4
                fprintf(fdd,"%d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n", \
                Cchain20(ic).Xinput,dump20(1).CCchain(ic).CCC(1),dump20(1).CCchain(ic).CCC(2),Cchain20(ic).Yinput,CS,
                dump20(1).CCchain(ic).SID_X(1),CS,dump20(1).CCchain(ic).SID_Y(1),CS,\
                dump20(1).CCchain(ic).SBID_X(1),CS,dump20(1).CCchain(ic).SBID_Y(1),CS,\
                dump20(1).CCchain(ic).BBID_X(1),CS,dump20(1).CCchain(ic).BBID_Y(1));
endfor
fclose(fdd);
```

3.18.5 tc41_data_ID_capture

```
CS="|";
fdd=fopen("tc41_data_ID_capture_table.txt","w");
fprintf(fdd,"chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS);
for ic=1:8
                fprintf(fdd,"%d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n", \
                Cchain41(ic).Xinput,dump41(1).CCchain(ic).CCC(1),dump41(1).CCchain(ic).CCC(2),Cchain41(ic).Yinput,CS,
```

```
                dump41(1).CCchain(ic).SID_X(1),CS,dump41(1).CCchain(ic).SID_Y(1),CS,\
                dump41(1).CCchain(ic).SBID_X(1),CS,dump41(1).CCchain(ic).SBID_Y(1),CS,\
                dump41(1).CCchain(ic).BBID_X(1),CS,dump41(1).CCchain(ic).BBID_Y(1));
endfor
fclose(fdd);
```

3.18.6 tc42_data_ID_capture

```
CS="|";
fdd=fopen("tc42_data_ID_capture_table.txt","w");
fprintf(fdd,"chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS);
for ic=1:8
                fprintf(fdd,"%d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n", \
                Cchain42(ic).Xinput,dump42(1).CCchain(ic).CCC(1),dump42(1).CCchain(ic).CCC(2),Cchain42(ic).Yinput,CS,
                dump42(1).CCchain(ic).SID_X(1),CS,dump42(1).CCchain(ic).SID_Y(1),CS,\
                dump42(1).CCchain(ic).SBID_X(1),CS,dump42(1).CCchain(ic).SBID_Y(1),CS,\
                dump42(1).CCchain(ic).BBID_X(1),CS,dump42(1).CCchain(ic).BBID_Y(1));
endfor
fclose(fdd);
```

3.18.7 tc46_data_ID_capture

```
CS="|";
fdd=fopen("tc46_data_ID_capture_table.txt","w");
fprintf(fdd,"chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS);
for ic=1:8
                fprintf(fdd,"%d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n", \
                Cchain46(ic).Xinput,dump46(1).CCchain(ic).CCC(1),dump46(1).CCchain(ic).CCC(2),Cchain46(ic).Yinput,CS,
                dump46(1).CCchain(ic).SID_X(1),CS,dump46(1).CCchain(ic).SID_Y(1),CS,\
                dump46(1).CCchain(ic).SBID_X(1),CS,dump46(1).CCchain(ic).SBID_Y(1),CS,\
                dump46(1).CCchain(ic).BBID_X(1),CS,dump46(1).CCchain(ic).BBID_Y(1));
endfor
fclose(fdd);
```

3.18.8 tc47_data_ID_capture

```
CS="|";
fdd=fopen("tc47_data_ID_capture_table.txt","w");
fprintf(fdd,"chain %s SID_X %s SID_Y %s SBID_X %s SBID_Y %s BBID_X %s BBID_Y \n",CS,CS,CS,CS,CS,CS);
for ic=1:8
                fprintf(fdd,"%d(%d:%d)%d %s %d %s %d %s %d %s %d %s %d %s %d\n", \
                Cchain47(ic).Xinput,dump47(1).CCchain(ic).CCC(1),dump47(1).CCchain(ic).CCC(2),Cchain47(ic).Yinput,CS,
                dump47(1).CCchain(ic).SID_X(1),CS,dump47(1).CCchain(ic).SID_Y(1),CS,\
                dump47(1).CCchain(ic).SBID_X(1),CS,dump47(1).CCchain(ic).SBID_Y(1),CS,\
                dump47(1).CCchain(ic).BBID_X(1),CS,dump47(1).CCchain(ic).BBID_Y(1));
endfor
fclose(fdd);
```

3.19 enable phase rotation

3.19.1 tc44_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc44_postsim.m : V2.00";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.40
               printf("WARNING: %s requires postsim_lib 1.40 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
# create structure of EXPECTED mcb states based on the test case tc44_apr24-05_v2.v
# along with an array containing the specified dump intervals in samples.
test_mcb.MCSR.AOV= 0;
test_mcb.MCSR.OVR= 0;
test_mcb.MCSR.SyncER= 0;
test_mcb.MCSR.TvER= 0;
test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0];
test_mcb.X_DUMP_SYNC= 0;
test_mcb.Y_DUMP_SYNC= 0;
test_mcb.X_SCHID_SYNC= 0;
test_mcb.Y_SCHID_SYNC= 0;
test_mcb.X_TIMESTAMP= 0;
test_mcb.Y_TIMESTAMP= 0;
test_mcb.X_DATA= [0;0;0;0;0;0;0;0];
test_mcb.Y_DATA= [0;0;0;0;0;0;0;0];
test_mcb.X_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0];
test_mcb.X_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0];
test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0];
test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0];

test_DESSR= [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]; # expected to be all set

base_SBID_X= 0;
base_SBID_Y= 0;
tc_SID_X= 85;
tc_SID_Y= 170;

tc= 44;
applied_dump= [40000, 0, 0, 0, 0, 0, 0, 0];

## ----------------------------------------------------------------------------------------
[frame44,num_frames,desc44,bad_frames,dontcare_frames]= get_lag_frames("lag_frames44.txt");
printf("%d frames read for test case 44.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc44,"# ver: v");
printf("test case version: %s\n", desc44((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb44 = get_mcb_data("mcb_reg44.txt");

## Form chains and input configuration based on the contents of the MCB register file.
[Cchain44, config44, unchain]= make_mcb_chains(mcb44);


## print chip configuration
nchains44= columns(Cchain44);
printf("\nChip configuration obtained from MCSR regsister settings.\n");
printf("List chains from 1 to %d.\n", nchains44);
for ic=1:4:nchains44
               printf("%d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d  %d(%d:%d)%d\n",\
               Cchain44(ic).Xinput,Cchain44(ic).CCC(1),\
               Cchain44(ic).CCC(end),Cchain44(ic).Yinput,\
               Cchain44(ic+1).Xinput,Cchain44(ic+1).CCC(1),\
               Cchain44(ic+1).CCC(end),Cchain44(ic+1).Yinput,\
               Cchain44(ic+2).Xinput,Cchain44(ic+2).CCC(1),\
               Cchain44(ic+2).CCC(end),Cchain44(ic+2).Yinput,\
```

```
              Cchain44(ic+3).Xinput,Cchain44(ic+3).CCC(1),\
              Cchain44(ic+3).CCC(end),Cchain44(ic+3).Yinput);
endfor
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump44, ndump44, ibadstamp] = make_dump_index(frame44);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

              # form contatenated chains for each dump on the basis of the output frame
              # status bits and add these to the dump structure.
              [dump44, with_chains44] = add_CCchain_dump(dump44, frame44);

              if !(ndump44 == rows(with_chains44))
                            printf("\nADVISORY: NOT ALL dumps contain chains!\n");
              endif
              timestamp44(1)= dump44(1).TIMESTAMP_0;
              printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp44(1));

# no point in running check_register_capture since there are only two chains and
# a single dump; 16 chains were intended in the original test case design.

              # Check that DESSR bits are all set as expected.
              if all(mcb44.DESSR == test_DESSR)
                            printf("\nCheck OK: DESSR bits are set as expected.\n");
              else
                            printf("\nWARNING: DESSR bits are set NOT as expected.\n");
              endif
endif

tname= "tc44";

printf("***--------------------------------------------------------------------------***\n");
```

3.19.2 tc44_similarity_check

```
Lnz= !ones(1,1);  # flag any non-zero difference
id=1;
ic1=[1];  # 8 chain pairs were intended in the original test case design
ic2=[2];
for ip=1:1
              Lnz(ic,id)= any(abs(dump44(id).CCchain(ic1(ip)).clag-\
              dump44(id).CCchain(ic2(ip)).clag) >= 10*eps);
endfor

if !any(any(Lnz))
              printf("\nNo differences exceeding the floating point precision limit of this\n\
              machine were found between similar chains of test case 44.\n\n");
else
              printf("\nADVISORY: Differences exceeding the floating point precision limit\n\
                of this machine were found between lag data of similar chains\n\
                within test case 44.\n\n");
endif
```

3.20 enable correlation

Post simulation scripts or functions are not required.

3.21 autocorrelation

3.21.1 tc56_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc56_postsim.m : V1.10";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
tname= "tc56";
[frame56,num_frames,desc56,bad_frames,dontcare_frames]= get_lag_frames("lag_frames56.txt");
printf("%d frames read for test case 56.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc56,"# ver: v");
printf("test case version: %s\n", desc56((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb56 = get_mcb_data("mcb_reg56.txt");
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump56, ndump56, ibadstamp] = make_dump_index(frame56);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump56, with_chains56] = add_CCchain_dump(dump56, frame56);

               if !(ndump56 == rows(with_chains56))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp56(1)= dump56(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp56(1));
endif

# create a 'fake' Cchain structure for plotting (make_mcb_chains does not yet support
# autocorr mode)
Cchain56(1).length= 16; # the length in cells of this chain;
Cchain56(1).CCC= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # the array of cells in this chain
Cchain56(1).Xinput= 0;  # the selected X input to this chain.
Cchain56(1).Yinput= -1; # set to -1 for autocorrelation

printf("\nWriting plot files\n");
write_fmod_fd_plots(1,1,dump56,Cchain56,tname,-40,5,1,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.21.2 tc57_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.
```

```
name_version= "tc57_postsim.m : V1.10";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
tname= "tc57";
[frame57,num_frames,desc57,bad_frames,dontcare_frames]= get_lag_frames("lag_frames57.txt");
printf("%d frames read for test case 57.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc57,"# ver: v");
printf("test case version: %s\n", desc57((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb57 = get_mcb_data("mcb_reg57.txt");
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump57, ndump57, ibadstamp] = make_dump_index(frame57);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump57, with_chains57] = add_CCchain_dump(dump57, frame57);

               if !(ndump57 == rows(with_chains57))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp57(1)= dump57(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp57(1));
endif

# create a 'fake' Cchain structure for plotting (make_mcb_chains does not yet support
# autocorr mode)
Cchain57(1).length= 16; # the length in cells of this chain;
Cchain57(1).CCC= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # the array of cells in this chain
Cchain57(1).Xinput= 1;  # the selected X input to this chain.
Cchain57(1).Yinput= -1; # set to -1 for autocorrelation

printf("\nWriting plot files\n");
write_fmod_fd_plots(1,1,dump57,Cchain57,tname,-40,5,1,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.21.3 tc58_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc58_postsim.m : V1.10";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
tname= "tc58";
[frame58,num_frames,desc58,bad_frames,dontcare_frames]= get_lag_frames("lag_frames58.txt");
printf("%d frames read for test case 58.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc58,"# ver: v");
printf("test case version: %s\n", desc58((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
```

```
                printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb58 = get_mcb_data("mcb_reg58.txt");
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump58, ndump58, ibadstamp] = make_dump_index(frame58);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump58, with_chains58] = add_CCchain_dump(dump58, frame58);

                if !(ndump58 == rows(with_chains58))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif
                timestamp58(1)= dump58(1).TIMESTAMP_0;
                printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp58(1));
endif

# create a 'fake' Cchain structure for plotting (make_mcb_chains does not yet support
# autocorr mode)
Cchain58(1).length= 16; # the length in cells of this chain;
Cchain58(1).CCC= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # the array of cells in this chain
Cchain58(1).Xinput= 2;  # the selected X input to this chain.
Cchain58(1).Yinput= -1; # set to -1 for autocorrelation

printf("\nWriting plot files\n");
write_fmod_fd_plots(1,1,dump58,Cchain58,tname,-40,5,1,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.21.4 tc59_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc59_postsim.m : V1.10";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
                printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
tname= "tc59";
[frame59,num_frames,desc59,bad_frames,dontcare_frames]= get_lag_frames("lag_frames59.txt");
printf("%d frames read for test case 59.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc59,"# ver: v");
printf("test case version: %s\n", desc59((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
                printf("Checksum is OK for all frames and corrupt data were not\n");
                printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb59 = get_mcb_data("mcb_reg59.txt");
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump59, ndump59, ibadstamp] = make_dump_index(frame59);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump59, with_chains59] = add_CCchain_dump(dump59, frame59);

                if !(ndump59 == rows(with_chains59))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif
```

```
                timestamp59(1)= dump59(1).TIMESTAMP_0;
                printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp59(1));
endif

# create a 'fake' Cchain structure for plotting (make_mcb_chains does not yet support
# autocorr mode)
Cchain59(1).length= 16; # the length in cells of this chain;
Cchain59(1).CCC= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # the array of cells in this chain
Cchain59(1).Xinput= 3;  # the selected X input to this chain.
Cchain59(1).Yinput= -1; # set to -1 for autocorrelation

printf("\nWriting plot files\n");
write_fmod_fd_plots(1,1,dump59,Cchain59,tname,-40,5,1,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.21.5 tc60_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc60_postsim.m : V1.10";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
                printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
tname= "tc60";
[frame60,num_frames,desc60,bad_frames,dontcare_frames]= get_lag_frames("lag_frames60.txt");
printf("%d frames read for test case 60.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc60,"# ver: v");
printf("test case version: %s\n", desc60((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
                printf("Checksum is OK for all frames and corrupt data were not\n");
                printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb60 = get_mcb_data("mcb_reg60.txt");
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump60, ndump60, ibadstamp] = make_dump_index(frame60);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump60, with_chains60] = add_CCchain_dump(dump60, frame60);

                if !(ndump60 == rows(with_chains60))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif
                timestamp60(1)= dump60(1).TIMESTAMP_0;
                printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp60(1));
endif

# create a 'fake' Cchain structure for plotting (make_mcb_chains does not yet support
# autocorr mode)
Cchain60(1).length= 16; # the length in cells of this chain;
Cchain60(1).CCC= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # the array of cells in this chain
Cchain60(1).Xinput= 4;  # the selected X input to this chain.
Cchain60(1).Yinput= -1; # set to -1 for autocorrelation

printf("\nWriting plot files\n");
write_fmod_fd_plots(1,1,dump60,Cchain60,tname,-40,5,1,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.21.6 tc61_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc61_postsim.m : V1.10";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
tname= "tc61";
[frame61,num_frames,desc61,bad_frames,dontcare_frames]= get_lag_frames("lag_frames61.txt");
printf("%d frames read for test case 61.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc61,"# ver: v");
printf("test case version: %s\n", desc61((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb61 = get_mcb_data("mcb_reg61.txt");
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump61, ndump61, ibadstamp] = make_dump_index(frame61);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump61, with_chains61] = add_CCchain_dump(dump61, frame61);

               if !(ndump61 == rows(with_chains61))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp61(1)= dump61(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp61(1));
endif

# create a 'fake' Cchain structure for plotting (make_mcb_chains does not yet support
# autocorr mode)
Cchain61(1).length= 16; # the length in cells of this chain;
Cchain61(1).CCC= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # the array of cells in this chain
Cchain61(1).Xinput= 5;  # the selected X input to this chain.
Cchain61(1).Yinput= -1; # set to -1 for autocorrelation

printf("\nWriting plot files\n");
write_fmod_fd_plots(1,1,dump61,Cchain61,tname,-40,5,1,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.21.7 tc62_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc62_postsim.m : V1.10";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------------
tname= "tc62";
[frame62,num_frames,desc62,bad_frames,dontcare_frames]= get_lag_frames("lag_frames62.txt");
```

```
printf("%d frames read for test case 62.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc62,"# ver: v");
printf("test case version: %s\n", desc62((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb62 = get_mcb_data("mcb_reg62.txt");
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump62, ndump62, ibadstamp] = make_dump_index(frame62);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps

               # form contatenated chains for each dump on the basis of the output frame
               # status bits and add these to the dump structure.
               [dump62, with_chains62] = add_CCchain_dump(dump62, frame62);

               if !(ndump62 == rows(with_chains62))
                              printf("\nADVISORY: NOT ALL dumps contain chains!\n");
               endif
               timestamp62(1)= dump62(1).TIMESTAMP_0;
               printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp62(1));
endif

# create a 'fake' Cchain structure for plotting (make_mcb_chains does not yet support
# autocorr mode)
Cchain62(1).length= 16; # the length in cells of this chain;
Cchain62(1).CCC= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # the array of cells in this chain
Cchain62(1).Xinput= 6;  # the selected X input to this chain.
Cchain62(1).Yinput= -1; # set to -1 for autocorrelation

printf("\nWriting plot files\n");
write_fmod_fd_plots(1,1,dump62,Cchain62,tname,-40,5,1,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.21.8 tc63_postsim

```
printf("***--------------------------------------------------------------------------***\n");
## Check and report version.
## There are no version checks on individual postsim_lib modules or for the presence of
## such modules in the current working directory.
## Be aware that such modules will be loaded before those contained within postsim_lib.

name_version= "tc63_postsim.m : V1.10";
printf("%s run time: %s",name_version, asctime(localtime(time())));
printf("Using postsim_lib: %s: %s\n\n",POSTSIM_LIB_VERSION, POSTSIM_LIB_VDATE);

if str2num(POSTSIM_LIB_VERSION) < 1.20
               printf("WARNING: %s requires postsim_lib 1.20 or greater!\n\n", name_version);
endif

## ----------------------------------------------------------------------------------
## ----------------------------------------------------------------------------------
tname= "tc63";
[frame63,num_frames,desc63,bad_frames,dontcare_frames]= get_lag_frames("lag_frames63.txt");
printf("%d frames read for test case 63.\n", num_frames);

# Report the version of the test case as read from the description field in the frame.
verdate_start=findstr(desc63,"# ver: v");
printf("test case version: %s\n", desc63((verdate_start+7):(verdate_start+16)));

if isempty(bad_frames) & isempty(dontcare_frames)
               printf("Checksum is OK for all frames and corrupt data were not\n");
               printf("detected (no 'x' or 'X' hex digits were found).\n");
endif

## Read the MCB register data from the register contents file.
mcb63 = get_mcb_data("mcb_reg63.txt");
printf("\n");

## create an array of dump structures, one element per dump, each of which contains an
## index of frames corresponding to a given dump.
[dump63, ndump63, ibadstamp] = make_dump_index(frame63);

if isempty(ibadstamp)  # continue processing only if there are no corrupt timestamps
```

```
                # form contatenated chains for each dump on the basis of the output frame
                # status bits and add these to the dump structure.
                [dump63, with_chains63] = add_CCchain_dump(dump63, frame63);

                if !(ndump63 == rows(with_chains63))
                                printf("\nADVISORY: NOT ALL dumps contain chains!\n");
                endif
                timestamp63(1)= dump63(1).TIMESTAMP_0;
                printf("\nInitial (single dump) timestamp (word 0): %d\n",timestamp63(1));endif

# create a 'fake' Cchain structure for plotting
Cchain63(1).length= 16; # the length in cells of this chain;
Cchain63(1).CCC= [0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15]; # the array of cells in this chain
Cchain63(1).Xinput= 7;  # the selected X input to this chain.
Cchain63(1).Yinput= -1; # set to -1 for autocorrelation

printf("\nWriting plot files\n");
write_fmod_fd_plots(1,1,dump63,Cchain63,tname,-40,5,1,1);
printf("***--------------------------------------------------------------------------***\n");
```

3.22 ancillary functions

3.22.1 function make_stat_tables

```
## desc.: make model difference lag and frequency domain
##        statistic tables for multiple dump test cases.
##
## usage: make_stat_tables(Cchain, dump, bmodel, tname,
##        ic_start, ic_stop, dmult, dmod, lag_peak_thold,
##        fd_peak_thold, Ldumpall)
##
## dmult: col. vector; multiples of 12000 sample dumps
##        for each chain
##  dmod: col. vector; clock modulus for each chain (1 is SE_CLK high)
##
##
## [Ver_date]= make_stat_tables();
## Ver_date: version number: date this version was created

## ********************* Revision history *********************
## (version; date created: description of changes)
## 1.00: July 22, 2005: first working version
## 1.10: July 23, 2005: fix to allow chains with no lag peaks
## 1.20: Aug   2, 2005: fix to skip chains with no bmodel data
## 1.30: Aug   4, 2005: remove redundant entries for dump all
##
## ************************************************************

function [Version_date]= make_stat_tables(Cchain, dump, bmodel, tname, ic_start,\
  ic_stop, dmult, dmod, std_threshold_L1, fd_std_threshold_L1, Ldumpall)

Version_date= "1.30: 05-08-04";
if nargin == 0
               return;
endif
if nargin < 9
               std_threshold_L1= 3;           # threshold for low level noise peak detection
               fd_std_threshold_L1= 10;  # threshold for residual freq. domain line detection
               Ldumpall= !1;                  # do not table all dumps; table all dumps if set true
endif
if nargin < 10
               fd_std_threshold_L1= 10;
               Ldumpall= !1;
endif
if nargin < 11
               Ldumpall= !1;
endif

snorm= 0.020./sqrt(dmult./dmod);
snormfd= 6e-4./sqrt(dmult./dmod);
bw = 1;  # do not scale with frequency (keep in units of power per bin)
CS="|";  # column delimiter for table data; one row per line
Lwin=1;

nchains= columns(Cchain);  # number of chains of this configuration
maxdumps= columns(dump);   # maximum number of dumps
Lplot= !ones(nchains,maxdumps);    # default plot flags
Lentry= !ones(maxdumps,nchains);   # diff table entry flags
LDentry= !ones(maxdumps,nchains);  # dump to dump table entry flags
LFDentry= !ones(maxdumps,nchains); # freq. diff table entry flags

Ltablef= fopen(sprintf("%s_diff_table.txt",tname),"w");
Dtablef= fopen(sprintf("%s_dump_diff_table.txt",tname),"w");
LFDtablef= fopen(sprintf("%s_diff_fd_table.txt",tname),"w");

fprintf(Ltablef,"chain %s dump %s mean_re %s mean_im %s range_re %s range_im %s\
 std_re %s std_im %s norm_std_re %s norm_std_im %s peak count %s description %s plot\n",\
 CS,CS,CS,CS,CS,CS,CS,CS,CS,CS,CS,CS);

fprintf(Dtablef,"chain %s dump %s mean_re %s mean_im %s range_re %s range_im %s\
 std_re %s std_im %s F/B std_re %s F/B std_im %s co-dump peak %s description %s plot\n",
CS,CS,CS,CS,CS,CS,CS,CS,CS,CS,CS,CS);

fprintf(LFDtablef,"chain %s dump %s mean_re %s mean_im %s range_re %s range_im %s\
 std_re %s std_im %s norm_std_re %s norm_std_im %s peak count %s description %s plot\n",\
 CS,CS,CS,CS,CS,CS,CS,CS,CS,CS,CS,CS);

# Compare and display the behavioural and functional model results for each chain in turn.
# Begin by finding and extracting the lag data for all dumps associated with each chain.
#
for ic=ic_start:ic_stop
               if isempty(bmodel(ic).length)
                              continue; # handle chains where there is no bmodel data; likely no fmodel dumps
               endif
               [fb,ff,bb,idump]= model_diff_chains(Cchain(ic), dump, bmodel(ic), bw, Lwin);
```

```
## generate statistics over all dumps
## not all stats will be used in these tables; keep for possible expansion
fb_std_re= std(real(fb.clag));
fb_std_im= std(imag(fb.clag));
fbfd_std_re= std(real(fb.fd));
fbfd_std_im= std(imag(fb.fd));

fb_range_re= range(real(fb.clag));
fb_range_im= range(imag(fb.clag));
fbfd_range_re= range(real(fb.fd));
fbfd_range_im= range(imag(fb.fd));

fb_mean= mean(fb.clag);
fbfd_mean= mean(fb.fd);

[max_fb_std_re, imax_fb_std_re]= max(fb_std_re);
[max_fb_std_im, imax_fb_std_im]= max(fb_std_im);
[max_fbfd_std_re, imax_fbfd_std_re]= max(fbfd_std_re);
[max_fbfd_std_im, imax_fbfd_std_im]= max(fbfd_std_im);
[min_fb_std_re, imin_fb_std_re]= min(fb_std_re);
[min_fb_std_im, imin_fb_std_im]= min(fb_std_im);
[min_fbfd_std_re, imin_fbfd_std_re]= min(fbfd_std_re);
[min_fbfd_std_im, imin_fbfd_std_im]= min(fbfd_std_im);

[max_fb_range_re, imax_fb_range_re]= max(fb_range_re);
[max_fb_range_im, imax_fb_range_im]= max(fb_range_im);
[max_fbfd_range_re, imax_fbfd_range_re]= max(fbfd_range_re);
[max_fbfd_range_im, imax_fbfd_range_im]= max(fbfd_range_im);
[min_fb_range_re, imin_fb_range_re]= min(fb_range_re);
[min_fb_range_im, imin_fb_range_im]= min(fb_range_im);
[min_fbfd_range_re, imin_fbfd_range_re]= min(fbfd_range_re);
[min_fbfd_range_im, imin_fbfd_range_im]= min(fbfd_range_im);

[max_fb_mean, imax_fb_mean]= max(fb_mean);
[max_fbfd_mean, imax_fbfd_mean]= max(fbfd_mean);
[min_fb_mean, imin_fb_mean]= min(fb_mean);
[min_fbfd_mean, imin_fbfd_mean]= min(fbfd_mean);

if !isempty(ff.clag) & !isempty(bb.clag) # quick fix to skip dump to dump statistics for single dump
                                         data
               bb_std_re= std(real(bb.clag));
               bb_std_im= std(imag(bb.clag));
               ff_std_re= std(real(ff.clag));
               ff_std_im= std(imag(ff.clag));
               bb_range_re= range(real(bb.clag));
               bb_range_im= range(imag(bb.clag));
               ff_range_re= range(real(ff.clag));
               ff_range_im= range(imag(ff.clag));
               bb_mean= mean(bb.clag);
               ff_mean= mean(ff.clag);

               [max_bb_std_re, imax_bb_std_re]= max(bb_std_re);
               [max_bb_std_im, imax_bb_std_im]= max(bb_std_im);
               [max_ff_std_re, imax_ff_std_re]= max(ff_std_re);
               [max_ff_std_im, imax_ff_std_im]= max(ff_std_im);

               [max_bb_range_re, imax_bb_range_re]= max(bb_range_re);
               [max_bb_range_im, imax_bb_range_im]= max(bb_range_im);
               [max_ff_range_re, imax_ff_range_re]= max(ff_range_re);
               [max_ff_range_im, imax_ff_range_im]= max(ff_range_im);

               [max_bb_mean, imax_bb_mean]= max(bb_mean);
               [max_ff_mean, imax_ff_mean]= max(ff_mean);

               [min_bb_std_re, imin_bb_std_re]= min(bb_std_re);
               [min_bb_std_im, imin_bb_std_im]= min(bb_std_im);
               [min_ff_std_re, imin_ff_std_re]= min(ff_std_re);
               [min_ff_std_im, imin_ff_std_im]= min(ff_std_im);

               [min_bb_range_re, imin_bb_range_re]= min(bb_range_re);
               [min_bb_range_im, imin_bb_range_im]= min(bb_range_im);
               [min_ff_range_re, imin_ff_range_re]= min(ff_range_re);
               [min_ff_range_im, imin_ff_range_im]= min(ff_range_im);

               [min_bb_mean, imin_bb_mean]= min(bb_mean);
               [min_ff_mean, imin_ff_mean]= min(ff_mean);

endif

ndumps= columns(idump);
peaks_L1= !ones(Cchain(ic).length*128, ndumps);
peak_count_L1= zeros(ndumps,1);

# determine level 2 spike detection threshold based on the rms value
# of the lag data for the behavioural model. Quantities above this value
# indicate a significant departure of the functional model from the behavioural
```

```
                std_threshold_L2= min([std(real(bmodel(ic).blags));std(imag(bmodel(ic).blags))]);

                for id=1:ndumps
                                temp= find(abs(real(fb.clag(:,id))) > std_threshold_L1*fb_std_re(id)\
                                | abs(imag(fb.clag(:,id))) > std_threshold_L1*fb_std_im(id));
                                if !isempty(temp)
                                                peaks_L1(temp,id)= !0;  # check for multiple occurences of lags > 3 std
                                endif
                                peak_count_L1(id)= rows(temp);                  # number of  L1 level peaks for this
                                                                                        dump
                                if (peak_count_L1(id) > Cchain(ic).length*128/100) && !Ldumpall
                                                tcode= "high peak count"; # peaks > 3 sigma for at least 1% of lags
                                                fprintf(Ltablef,\
                                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(id),CS,real(fb_mean(id)),CS,imag(fb_mean(id)),CS,fb_range_re(id),CS,\
                                                fb_range_im(id),CS,fb_std_re(id),CS,fb_std_im(id),CS,\
                                                fb_std_re(id)/snorm(ic),CS,fb_std_im(id)/snorm(ic),CS,\
                                                peak_count_L1(id),CS,tcode,CS,Lplot(ic,idump(id)));
                                endif
                                if Ldumpall
                                                tcode= "all dumps";
                                                if !Lentry(id,ic)
                                                                fprintf(Ltablef,\
                                                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s
%6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(id),CS,real(fb_mean(id)),CS,imag(fb_mean(id)),CS,fb_range_re(id),CS,\
                                                                fb_range_im(id),CS,fb_std_re(id),CS,fb_std_im(id),CS,\
                                                                fb_std_re(id)/snorm(ic),CS,fb_std_im(id)/snorm(ic),CS,\
                                                                peak_count_L1(id),CS,tcode,CS,Lplot(ic,idump(id)));
                                                                Lentry(id,ic)= 1;
                                                endif
                                endif

                                # process frequency domain difference; analagous quantities to lag domain above
                                temp= find(abs(real(fb.fd(:,id))) > fd_std_threshold_L1*fbfd_std_re(id)\
                                | abs(imag(fb.fd(:,id))) > fd_std_threshold_L1*fbfd_std_im(id));
                                if !isempty(temp)
                                                fd_peaks_L1(temp,id)= !0;
                                endif
                                fd_peak_count_L1(id)= rows(temp);                   # number of  L1 level peaks for
                                                                                        this dump
                                if (fd_peak_count_L1(id) > 0) && !Ldumpall
                                                tcode= "high peak count"; # any peaks > threshold
                                                fprintf(LFDtablef,\
                                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\

ic,CS,idump(id),CS,real(fbfd_mean(id)),CS,imag(fbfd_mean(id)),CS,fbfd_range_re(id),CS,\
                                                fbfd_range_im(id),CS,fbfd_std_re(id),CS,fbfd_std_im(id),CS,\
                                                fbfd_std_re(id)/snormfd(ic),CS,fbfd_std_im(id)/snormfd(ic),CS,\
                                                fd_peak_count_L1(id),CS,tcode,CS,Lplot(ic,idump(id)));
                                endif
                                if Ldumpall
                                                tcode= "all dumps";
                                                if !LFDentry(id,ic)
                                                                fprintf(LFDtablef,\
                                                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s
%4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\

ic,CS,idump(id),CS,real(fbfd_mean(id)),CS,imag(fbfd_mean(id)),\
                                                                CS,fbfd_range_re(id),CS,\

fbfd_range_im(id),CS,fbfd_std_re(id),CS,fbfd_std_im(id),CS,\

fbfd_std_re(id)/snormfd(ic),CS,fbfd_std_im(id)/snormfd(ic),CS,\
                                                                fd_peak_count_L1(id),CS,tcode,CS,Lplot(ic,idump(id)));
                                                                LFDentry(id,ic)= 1;
                                                endif
                                endif
                endfor
                if !Ldumpall
                                tcode= "max. re. stdev";  # max. real lag stdev for this chain
                                imax= imax_fb_std_re;
                                fprintf(Ltablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(imax),CS,real(fb_mean(imax)),CS,imag(fb_mean(imax)),CS,fb_range_re(imax),CS,\
                                fb_range_im(imax),CS,max_fb_std_re,CS,fb_std_im(imax),CS,\
                                fb_std_re(imax)/snorm(ic),CS,fb_std_im(imax)/snorm(ic),CS,\
                                peak_count_L1(imax),CS,tcode,CS,Lplot(ic,idump(imax)));
```

```
                              tcode= "max. im. stdev";  # max. imag. lag stdev for this chain
                              imax= imax_fb_std_im;
                              fprintf(Ltablef,\
                              "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(imax),CS,real(fb_mean(imax)),CS,imag(fb_mean(imax)),CS,fb_range_re(imax),CS,\
                              fb_range_im(imax),CS,fb_std_re(imax),CS,max_fb_std_im,CS,\
                              fb_std_re(imax)/snorm(ic),CS,fb_std_im(imax)/snorm(ic),CS,\
                              peak_count_L1(imax),CS,tcode,CS,Lplot(ic,idump(imax)));

                              tcode= "min. re. stdev";  # min. real lag stdev for this chain
                              imin= imin_fb_std_re;
                              fprintf(Ltablef,\
                              "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(imin),CS,real(fb_mean(imin)),CS,imag(fb_mean(imin)),CS,fb_range_re(imin),CS,\
                              fb_range_im(imin),CS,min_fb_std_re,CS,fb_std_im(imin),CS,\
                              fb_std_re(imin)/snorm(ic),CS,fb_std_im(imin)/snorm(ic),CS,\
                              peak_count_L1(imin),CS,tcode,CS,Lplot(ic,idump(imax)));

                              tcode= "min. im. stdev";  # min. imag. lag stdev for this chain
                              imin= imin_fb_std_im;
                              fprintf(Ltablef,\
                              "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(imin),CS,real(fb_mean(imin)),CS,imag(fb_mean(imin)),CS,fb_range_re(imin),CS,\
                              fb_range_im(imin),CS,fb_std_re(imin),CS,min_fb_std_im,CS,\
                              fb_std_re(imin)/snorm(ic),CS,fb_std_im(imin)/snorm(ic),CS,\
                              peak_count_L1(imin),CS,tcode,CS,Lplot(ic,idump(imax)));

                              tcode= "max. re. range";  # max. real lag stdev for this chain
                              imax= imax_fb_range_re;
                              fprintf(Ltablef,\
                              "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(imax),CS,real(fb_mean(imax)),CS,imag(fb_mean(imax)),CS,fb_range_re(imax),CS,\
                              fb_range_im(imax),CS,fb_std_re(imax),CS,fb_std_im(imax),CS,\
                              fb_std_re(imax)/snorm(ic),CS,fb_std_im(imax)/snorm(ic),CS,\
                              peak_count_L1(imax),CS,tcode,CS,Lplot(ic,idump(imax)));

                              tcode= "max. im. range";  # max. imag. lag stdev for this chain
                              imax= imax_fb_range_im;
                              fprintf(Ltablef,\
                              "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(imax),CS,real(fb_mean(imax)),CS,imag(fb_mean(imax)),CS,fb_range_re(imax),CS,\
                              fb_range_im(imax),CS,fb_std_re(imax),CS,fb_std_im(imax),CS,\
                              fb_std_re(imax)/snorm(ic),CS,fb_std_im(imax)/snorm(ic),CS,\
                              peak_count_L1(imax),CS,tcode,CS,Lplot(ic,idump(imax)));

                              tcode= "min. re. range";  # min. real lag stdev for this chain
                              imin= imin_fb_range_re;
                              fprintf(Ltablef,\
                    "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(imin),CS,real(fb_mean(imin)),CS,imag(fb_mean(imin)),CS,fb_range_re(imin),CS,\
                              fb_range_im(imin),CS,fb_std_re(imin),CS,fb_std_im(imin),CS,\
                              fb_std_re(imin)/snorm(ic),CS,fb_std_im(imin)/snorm(ic),CS,\
                              peak_count_L1(imin),CS,tcode,CS,Lplot(ic,idump(imax)));

                              tcode= "max. im. range";  # min. imag. lag stdev for this chain
                              imin= imin_fb_range_im;
                              fprintf(Ltablef,\
                              "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %d %s %s %s %d\n",\

ic,CS,idump(imin),CS,real(fb_mean(imin)),CS,imag(fb_mean(imin)),CS,fb_range_re(imin),CS,\
                              fb_range_im(imin),CS,fb_std_re(imin),CS,fb_std_im(imin),CS,\
                              fb_std_re(imin)/snorm(ic),CS,fb_std_im(imin)/snorm(ic),CS,\
                              peak_count_L1(imin),CS,tcode,CS,Lplot(ic,idump(imax)));

###  frequency domain  tables entries
                              tcode= "max. re. stdev";
                              im= imax_fbfd_std_re;
                              fprintf(LFDtablef,\
                              "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\
                              ic,CS,idump(im),CS,real(fbfd_mean(im)),CS,imag(fbfd_mean(im)),CS,fbfd_range_re(im),CS,\
                              fbfd_range_im(im),CS,fbfd_std_re(im),CS,fbfd_std_im(im),CS,\
                              fbfd_std_re(im)/snormfd(ic),CS,fbfd_std_im(im)/snormfd(ic),CS,\
                              fd_peak_count_L1(im),CS,tcode,CS,Lplot(ic,idump(im)));
```

```
                                tcode= "max. im. stdev";
                                im= imax_fbfd_std_im;
                                fprintf(LFDtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\
                                ic,CS,idump(im),CS,real(fbfd_mean(im)),CS,imag(fbfd_mean(im)),CS,fbfd_range_re(im),CS,\
                                fbfd_range_im(im),CS,fbfd_std_re(im),CS,fbfd_std_im(im),CS,\
                                fbfd_std_re(im)/snormfd(ic),CS,fbfd_std_im(im)/snormfd(ic),CS,\
                                fd_peak_count_L1(im),CS,tcode,CS,Lplot(ic,idump(im)));

                                tcode= "min. re. stdev";
                                im= imin_fbfd_std_re;
                                fprintf(LFDtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\
                                ic,CS,idump(im),CS,real(fbfd_mean(im)),CS,imag(fbfd_mean(im)),CS,fbfd_range_re(im),CS,\
                                fbfd_range_im(im),CS,fbfd_std_re(im),CS,fbfd_std_im(im),CS,\
                                fbfd_std_re(im)/snormfd(ic),CS,fbfd_std_im(im)/snormfd(ic),CS,\
                                fd_peak_count_L1(im),CS,tcode,CS,Lplot(ic,idump(im)));

                                tcode= "min. im. stdev";
                                im= imin_fbfd_std_im;
                                fprintf(LFDtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\
                                ic,CS,idump(im),CS,real(fbfd_mean(im)),CS,imag(fbfd_mean(im)),CS,fbfd_range_re(im),CS,\
                                fbfd_range_im(im),CS,fbfd_std_re(im),CS,fbfd_std_im(im),CS,\
                                fbfd_std_re(im)/snormfd(ic),CS,fbfd_std_im(im)/snormfd(ic),CS,\
                                fd_peak_count_L1(im),CS,tcode,CS,Lplot(ic,idump(im)));

                                tcode= "max. re. range";
                                im= imax_fbfd_range_re;
                                fprintf(LFDtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\
                                ic,CS,idump(im),CS,real(fbfd_mean(im)),CS,imag(fbfd_mean(im)),CS,fbfd_range_re(im),CS,\
                                fbfd_range_im(im),CS,fbfd_std_re(im),CS,fbfd_std_im(im),CS,\
                                fbfd_std_re(im)/snormfd(ic),CS,fbfd_std_im(im)/snormfd(ic),CS,\
                                fd_peak_count_L1(im),CS,tcode,CS,Lplot(ic,idump(im)));

                                tcode= "max. im. range";
                                im= imax_fbfd_range_im;
                                fprintf(LFDtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\
                                ic,CS,idump(im),CS,real(fbfd_mean(im)),CS,imag(fbfd_mean(im)),CS,fbfd_range_re(im),CS,\
                                fbfd_range_im(im),CS,fbfd_std_re(im),CS,fbfd_std_im(im),CS,\
                                fbfd_std_re(im)/snormfd(ic),CS,fbfd_std_im(im)/snormfd(ic),CS,\
                                fd_peak_count_L1(im),CS,tcode,CS,Lplot(ic,idump(im)));

                                tcode= "min. re. range";
                                im= imin_fbfd_range_re;
                                fprintf(LFDtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\
                                ic,CS,idump(im),CS,real(fbfd_mean(im)),CS,imag(fbfd_mean(im)),CS,fbfd_range_re(im),CS,\
                                fbfd_range_im(im),CS,fbfd_std_re(im),CS,fbfd_std_im(im),CS,\
                                fbfd_std_re(im)/snormfd(ic),CS,fbfd_std_im(im)/snormfd(ic),CS,\
                                fd_peak_count_L1(im),CS,tcode,CS,Lplot(ic,idump(im)));

                                tcode= "min. im. range";
                                im= imin_fbfd_range_im;
                                fprintf(LFDtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %4.2e %s %4.2e %s \
%3.1f %s %3.1f %s %d %s %s %s %d\n",\
                                ic,CS,idump(im),CS,real(fbfd_mean(im)),CS,imag(fbfd_mean(im)),CS,fbfd_range_re(im),CS,\
                                fbfd_range_im(im),CS,fbfd_std_re(im),CS,fbfd_std_im(im),CS,\
                                fbfd_std_re(im)/snormfd(ic),CS,fbfd_std_im(im)/snormfd(ic),CS,\
                                fd_peak_count_L1(im),CS,tcode,CS,Lplot(ic,idump(im)));
                endif

                ipeaks_L1= find(peaks_L1');
                if isempty(ipeaks_L1)
                                ilag_cp= ipeaks_L1;
                                ilag_dump= ipeaks_L1;
                else
                                [ip,jp]= find(peaks_L1');
                                ilag_cp= jp(find(!diff(jp)));    # lag indices of peaks that occur in more than one
                                                                                                dump
                                ilag_dump= ip(find(!diff(jp)));  # dump indices with peaks in more than one dump; first
                                                                                                index
                endif
## lag domain dump to dump statistics table
                if !Ldumpall
                                if isempty(ff.clag) || isempty(bb.clag)
                                                continue; # quick fix to skip dump to dump statistics for single dump
```

```
                                endif
                                tcode= "max. re. stdev";  # max. real lag stdev for this chain
                                imax= imax_ff_std_re;
                                pog= find(imax==ilag_dump);
                                if isempty(pog)
                                        spogg= "null";
                                else
                                        spogg= sprintf(" %d",ilag_cp(pog)-1);
                                endif
                                fprintf(Dtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
                        %3.2f %s %3.2f %s %s %s %s %s %d\n",\

        ic,CS,idump(imax),CS,real(ff_mean(imax)),CS,imag(ff_mean(imax)),CS,ff_range_re(imax),CS,\
                                ff_range_im(imax),CS,max_ff_std_re,CS,ff_std_im(imax),CS,\
                                ff_std_re(imax)/bb_std_re(imax),CS,ff_std_im(imax)/bb_std_im(imax),CS,\
                                spogg,CS,tcode,CS,Lplot(ic,idump(imax)));

                                tcode= "max. im. stdev";  # max. imag. lag stdev for this chain
                                imax= imax_ff_std_im;
                                pog= find(idump(imax)==ilag_dump);
                                if isempty(pog)
                                        spogg= "null";
                                else
                                        spogg= sprintf(" %d",ilag_cp(pog)-1);
                                endif
                                fprintf(Dtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
                        %3.2f %s %3.2f %s %s %s %s %s %d\n",\

        ic,CS,idump(imax),CS,real(ff_mean(imax)),CS,imag(ff_mean(imax)),CS,ff_range_re(imax),CS,\
                                ff_range_im(imax),CS,ff_std_re(imax),CS,max_ff_std_im,CS,\
                                ff_std_re(imax)/bb_std_re(imax),CS,ff_std_im(imax)/bb_std_im(imax),CS,\
                                spogg,CS,tcode,CS,Lplot(ic,idump(imax)));

                                tcode= "min. re. stdev";  # min. real lag stdev for this chain
                                imin= imin_ff_std_re;
                                pog= find(idump(imin)==ilag_dump);
                                if isempty(pog)
                                        spogg= "null";
                                else
                                        spogg= sprintf(" %d",ilag_cp(pog)-1);
                                endif
                                fprintf(Dtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
                        %3.2f %s %3.2f %s %s %s %s %s %d\n",\

        ic,CS,idump(imin),CS,real(ff_mean(imin)),CS,imag(ff_mean(imin)),CS,ff_range_re(imin),CS,\
                                ff_range_im(imin),CS,min_ff_std_re,CS,ff_std_im(imin),CS,\
                                ff_std_re(imin)/bb_std_re(imin),CS,ff_std_im(imin)/bb_std_im(imin),CS,\
                                spogg,CS,tcode,CS,Lplot(ic,idump(imin)));

                                tcode= "min. im. stdev";  # min. imag. lag stdev for this chain
                                imin= imin_ff_std_im;
                                pog= find(idump(imin)==ilag_dump);
                                if isempty(pog)
                                        spogg= "null";
                                else
                                        spogg= sprintf(" %d",ilag_cp(pog)-1);
                                endif
                                fprintf(Dtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
                        %3.2f %s %3.2f %s %s %s %s %s %d\n",\

        ic,CS,idump(imin),CS,real(ff_mean(imin)),CS,imag(ff_mean(imin)),CS,ff_range_re(imin),CS,\
                                ff_range_im(imin),CS,ff_std_re(imin),CS,min_ff_std_im,CS,\
                                ff_std_re(imin)/bb_std_re(imin),CS,ff_std_im(imin)/bb_std_im(imin),CS,\
                                spogg,CS,tcode,CS,Lplot(ic,idump(imin)));

                                tcode= "max. re. range";  # max. real lag stdev for this chain
                                imax= imax_ff_range_re;
                                pog= find(idump(imax)==ilag_dump);
                                if isempty(pog)
                                        spogg= "null";
                                else
                                        spogg= sprintf(" %d",ilag_cp(pog)-1);
                                endif
                                fprintf(Dtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
                        %3.2f %s %3.2f %s %s %s %s %s %d\n",\

        ic,CS,idump(imax),CS,real(ff_mean(imax)),CS,imag(ff_mean(imax)),CS,ff_range_re(imax),CS,\
                                ff_range_im(imax),CS,ff_std_re(imax),CS,ff_std_im(imax),CS,\
                                ff_std_re(imax)/bb_std_re(imax),CS,ff_std_im(imax)/bb_std_im(imax),CS,\
                                spogg,CS,tcode,CS,Lplot(ic,idump(imax)));
```

```
                                tcode= "max. im. range";  # max. imag. lag stdev for this chain
                                imax= imax_ff_range_im;
                                pog= find(idump(imax)==ilag_dump);
                                if isempty(pog)
                                                spogg= "null";
                                else
                                                spogg= sprintf(" %d",ilag_cp(pog)-1);
                                endif
                                fprintf(Dtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
                        %3.2f %s %3.2f %s %s %s %s %s %d\n",\

ic,CS,idump(imax),CS,real(ff_mean(imax)),CS,imag(ff_mean(imax)),CS,ff_range_re(imax),CS,\
                                ff_range_im(imax),CS,ff_std_re(imax),CS,ff_std_im(imax),CS,\
                                ff_std_re(imax)/bb_std_re(imax),CS,ff_std_im(imax)/bb_std_im(imax),CS,\
                                spogg,CS,tcode,CS,Lplot(ic,idump(imax)));

                                tcode= "min. re. range";  # min. real lag stdev for this chain
                                imin= imin_ff_range_re;
                                pog= find(idump(imin)==ilag_dump);
                                if isempty(pog)
                                                spogg= "null";
                                else
                                                spogg= sprintf(" %d",ilag_cp(pog)-1);
                                endif
                                fprintf(Dtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
                        %3.2f %s %3.2f %s %s %s %s %s %d\n",\

ic,CS,idump(imin),CS,real(ff_mean(imin)),CS,imag(ff_mean(imin)),CS,ff_range_re(imin),CS,\
                                ff_range_im(imin),CS,ff_std_re(imin),CS,ff_std_im(imin),CS,\
                                ff_std_re(imin)/bb_std_re(imin),CS,ff_std_im(imin)/bb_std_im(imin),CS,\
                                spogg,CS,tcode,CS,Lplot(ic,idump(imin)));

                                tcode= "min. im. range";  # min. imag. lag stdev for this chain
                                imin= imin_ff_range_im;
                                pog= find(idump(imin)==ilag_dump);
                                if isempty(pog)
                                                spogg= "null";
                                else
                                                spogg= sprintf(" %d",ilag_cp(pog)-1);
                                endif
                                fprintf(Dtablef,\
                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
                        %3.2f %s %3.2f %s %s %s %s %s %d\n",\

ic,CS,idump(imin),CS,real(ff_mean(imin)),CS,imag(ff_mean(imin)),CS,ff_range_re(imin),CS,\
                                ff_range_im(imin),CS,ff_std_re(imin),CS,ff_std_im(imin),CS,\
                                ff_std_re(imin)/bb_std_re(imin),CS,ff_std_im(imin)/bb_std_im(imin),CS,\
                                spogg,CS,tcode,CS,Lplot(ic,idump(imin)));
                        endif

                        for id=1:(ndumps-1)
                                pog= find(idump(id)==ilag_dump);
                                if !isempty(pog)
                                # n-1 dumps will be tabled where n is the number of dumps in which
                                # this lag peak occurs; the highest lag number is excluded.
                                # NOTE: a peak is either +/-real or +/-imag.; hence this process is
                                # is not very discriminating!
                                                tcode= "coincident dump peaks";
                                                spogg= sprintf(" %d",ilag_cp(pog)-1);
                                                if !Ldumpall
                                                                fprintf(Dtablef,\
                                                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s
%6.4f %s \
%3.2f %s %3.2f %s %s %s %s %s %d\n",\

ic,CS,idump(id),CS,real(ff_mean(id)),CS,imag(ff_mean(id)),CS,ff_range_re(id),CS,\
                                                                ff_range_im(id),CS,ff_std_re(id),CS,ff_std_im(id),CS,\

ff_std_re(id)/bb_std_re(id),CS,ff_std_im(id)/bb_std_im(id),CS,\
                                                                spogg,CS,tcode,CS,Lplot(ic,idump(id)));
                                                endif
                                else
                                                spogg= sprintf(" null");
                                endif
                                if Ldumpall
                                                tcode= "all dumps";
                                                if !LDentry(id,ic)
                                                                fprintf(Dtablef,\
                                                                "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s
%6.4f %s \
%3.2f %s %3.2f %s %s %s %s %s %d\n",\

ic,CS,idump(id),CS,real(ff_mean(id)),CS,imag(ff_mean(id)),CS,ff_range_re(id),CS,\
                                                                ff_range_im(id),CS,ff_std_re(id),CS,ff_std_im(id),CS,\
```

```
ff_std_re(id)/bb_std_re(id),CS,ff_std_im(id)/bb_std_im(id),CS,\
                                                        spogg,CS,tcode,CS,Lplot(ic,idump(id)));
                                                        LDentry(id,ic)= 1;
                                            endif
                                endif
                                if ((ff_std_re(id)/bb_std_re(id)) < 0.9 | 1.1 < (ff_std_re(id)/bb_std_re(id))) && !
Ldumpall
                                            tcode= "F/B stdev not unity";
                                            fprintf(Dtablef,\
                                            "%d %s %d %s %4.2e %s %4.2e %s %6.4f %s %6.4f %s %6.4f %s %6.4f %s \
%3.2f %s %3.2f %s %s %s %s %s %d\n",\

ic,CS,idump(id),CS,real(ff_mean(id)),CS,imag(ff_mean(id)),CS,ff_range_re(id),CS,\
                                            ff_range_im(id),CS,ff_std_re(id),CS,ff_std_im(id),CS,\
                                            ff_std_re(id)/bb_std_re(id),CS,ff_std_im(id)/bb_std_im(id),CS,\
                                            spogg,CS,tcode,CS,Lplot(ic,idump(id)));
                                endif
                    endfor
endfor
fclose(Ltablef);
fclose(Dtablef);
fclose(LFDtablef);
endfunction
```

3.22.2 model_diff_chains

```
## desc: provides the lag and frequency domain
##       difference between functional and behavioural models
##       on a single chain basis.
##
## usage: [fb,ff,bb,idump]= model_diff_chains(Cchain(ic),
##                             dump, bmodel(ic), bw, Lwin);
##
## Cchain(ic): one element of an mcb chain structure array.
##       dump: an array of dump stuctures.
## bmodel(ic): one element of a behavioural model structure
##             array.
##    bw,Lwin: bandwidth and window arguments for
##             function Lag2fd.
##    fb.clag: complex lag difference array, size
##             nlags x ndumps where ndumps is the
##             number of dumps for this chain.
##      fb.fd: complex freq. domain difference array, size
##             nlags x ndumps where ndumps is the
##             number of dumps for this chain.
##      ff.**: as above but dump to dump difference of
##             functional model, size nlags x ndumps-1.
##      bb.**: as above but dump to dump difference of
##             behavioural model, size nlags x ndumps-1.
##     idump: element index of dump for this chain.
##       NOTE: dump indices of the behavioural model
##             are consecutive.
##
##
## [Ver_date]= model_diff_chains();
## Ver_date: version number: date this version was created

## ********************** Revision history **********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.10: add fix for configs having chains of different length
##
## ************************************************************

function [fb,ff,bb,idump]= model_diff_chains(Cchain, dump, bmodel, bw, Lwin)

Version_date= "1.10: 05-10-02";
if nargin == 0
              fb= Version_date;
              return;
endif

if !(Cchain(1).Xinput == bmodel(1).Xin && Cchain(1).Yinput == bmodel(1).Yin &&\
Cchain(1).length == bmodel(1).length)
              printf("\nError from model_diff_chains: chain mismatch.\n");
              return;
endif
ndumps= columns(dump);
Ldump= !ones(1,ndumps);
clag= zeros(Cchain(1).length*128, ndumps);
fd= zeros(Cchain(1).length*128, ndumps);
ff= zeros(Cchain(1).length*128, ndumps-1);
bb= zeros(Cchain(1).length*128, ndumps-1);
```

```
fffd= zeros(Cchain(1).length*128, ndumps-1);
bbfd= zeros(Cchain(1).length*128, ndumps-1);

ibd=0;  # initialize bmodel dump counter
for id=1:ndumps
               nchains=columns(dump(id).CCchain(:));
               for ic=1:nchains   # compare cell numbers of each chain; don't assume a particular chain dump order
                              if rows(dump(id).CCchain(ic).CCC) != Cchain(1).length;continue;endif;
                              if all(dump(id).CCchain(ic).CCC == Cchain(1).CCC)
                                             clag(:,id)= dump(id).CCchain(ic).clag(:) - bmodel(1).blags(:,++ibd);
                                             if ibd > 1
                                                            ff(:,id)= dump(id).CCchain(ic).clag(:) - fx;
                                                            bb(:,id)= bmodel(1).blags(:,ibd) - bx;
                                             endif
                                             fx= dump(id).CCchain(ic).clag(:);
                                             bx= bmodel(1).blags(:,ibd);

                                             fd_temp= lag2fd(dump(id).CCchain(ic).clag(:),bw,Lwin);
                                             bfd_temp= lag2fd(bmodel(1).blags(:,ibd),bw,Lwin);
                                             fd(:,id)=  fd_temp - bfd_temp;
                                             if ibd > 1
                                                            fffd(:,id)= fd_temp - fxfd;
                                                            bbfd(:,id)= bfd_temp - bxfd;
                                             endif
                                             fxfd= fd_temp;
                                             bxfd= bfd_temp;
                                             Ldump(id)= !0;
                              endif
               endfor
endfor

idump= find(Ldump);
fb.clag= clag(:,idump);
fb.fd= fd(:,idump);
ff.clag= ff(:,idump(2:end));
ff.fd= fffd(:,idump(2:end));
bb.clag= bb(:,idump(2:end));
bb.fd= bbfd(:,idump(2:end));

if ibd != columns(idump)
               printf("\nError from model_diff_chains: Number of dumps differs\n\
               between models.\n");
               printf("\nnumber of bmodel dumps: %d\n",ibd);
               return
endif
endfunction
```

3.22.3 check_timestamp

```
## Compare timestamp word 0 against the first dump
## in dump structure; timestamp supplied value in numeric
## or string hex fromat.
##
## usage: check_timestamp(dump, timestamp0)
##

function check_timestamp(dump, timestamp0)

if !isnumeric(timestamp0)
               timestamp0= hex2dec(timestamp0);
endif

if !(dump(1).TIMESTAMP_0 == timestamp0)
               printf("\nWARNING: Unexpected timestamp word 0.\n");
else
               printf("\nCheck OK: Expected timestamp word 0.\n");
endif
```

3.22.4 write_lag_plots

```
## desc: generate lag plot files
## usage: write_lag_plots(ic,ida,idump,dump,fb,nlag,tname,
##                                 bmin,bmax,rlim, Cchain)
##     ic: chain number (first chain is 1)
##    ida: dump number (out of total dumps)
##  idump: index array of chip dump for a given chain dump
##   dump: full dump structure: bmod(ic).blags(:,id)
##     fb: full fb structure for chain ic: fb.clag(:,id)
##   nlag: number of lags
##  tname: file identifier
##   bmin: complex B model minimum axis
##   bmax: complex B model maximum axis
##   rlim: complex maximum of re, im difference axis
## Cchain: mcb_reg structure
##
##  NOTE: "a" for bmin,bmax,rlim --> y autoscaling

## version: 2.11; 05-08-26

function write_lag_plots(ic,ida,idump,dump,fb,nlag,tname,bmin,bmax,rlim,Cchain)

id= find(idump == ida);

if isempty(id)
               printf("\nERROR from write_lag_plots: Can't find dump index.\n");
               printf("Chain %d may not have participated in dump %d.\n",ic,ida);
               return
end

fmod= dump(ida).CCchain(ic).clag;
if !isnumeric(bmin)
               bmin= min(real(fmod))+min(imag(fmod))*1i;
endif
if !isnumeric(bmax)
               bmax= max(real(fmod))+max(imag(fmod))*1i;
endif
if !isnumeric(rlim)
               rlim= max(abs(real(fb.clag(:,id))))+max(abs(imag(fb.clag(:,id))))*1i;
endif

chainst= sprintf("%d(%d:%d)%d %d(%d:%d)%d %d(%d:%d)%d %d(%d:%d)%d\n",\
Cchain(ic).Xinput,Cchain(ic).CCC(1), Cchain(ic).CCC(end),Cchain(ic).Yinput);

plotfext="eps";

lagnum= 0:(nlag-1);
# gset term X11
# gset output
gset term postscript eps 14
gset nokey
eval(sprintf("gset xtics %d, %d, %d", lagnum(1), nlag/4, lagnum(nlag)));
gset mxtics 2; # every second labelled tick
gset size ratio 0.3

axis ([0, nlag-1, real(bmin),real(bmax)]);
xlabel("lag number");
eval(sprintf("title(""%s RTL model, chain %d: %s, chip dump: %d, chain dump: %d, real."")",\
tname, ic, chainst, ida, id));
eval(sprintf("gset output ""%s_c%dd%d_lag_rmod_re.%s"";",tname,ic,ida, plotfext));
plot(lagnum, real(fmod));

axis ([0, nlag-1, imag(bmin),imag(bmax)]);
eval(sprintf("title(""%s RTL model, chain %d: %s, chip dump: %d, chain dump: %d, imag."")",\
tname, ic, chainst,ida, id));
eval(sprintf("gset output ""%s_c%dd%d_lag_rmod_im.%s"";",tname,ic,ida, plotfext));
plot(lagnum, imag(fmod));

axis ([0, nlag-1, -real(rlim), real(rlim)]);
eval(sprintf("title(""%s-behavioural, chain %d: %s, chip dump: %d, chain dump: %d, real."")",\
tname, ic, chainst, ida, id));
eval(sprintf("gset output ""%s_c%dd%d_lag_rdif_re.%s"";",tname, ic, ida, plotfext));
plot(lagnum, real(fb.clag(:,id)))

eval(sprintf("title(""%s-behavioural, chain %d: %s, chip dump: %d, chain dump: %d,  imag."")",\
tname, ic, chainst, ida, id));
axis ([0, nlag-1, -imag(rlim), imag(rlim)]);
eval(sprintf("gset output ""%s_c%dd%d_lag_rdif_im.%s"";",tname, ic, ida, plotfext));
plot(lagnum, imag(fb.clag(:,id)))

# Generate histogram plots of the functional - behavioural model difference
nbins= 20; # number of bins
nbintics= 10; # number of bins to label with tics
xh=-real(rlim):(2*real(rlim))/nbins:real(rlim);
axis;
```

```
eval(sprintf("gset xtic %6.3f,%6.3f,%6.3f",-real(rlim),2*real(rlim)/nbintics,real(rlim)));
gset nomxtics
xlabel("");
eval(sprintf("title(""%s-behavioural, chain %d: %s, chip dump: %d, chain dump: %d, real"")",\
tname, ic, chainst, ida, id));
[yh,xh]=hist(real(fb.clag(:,id)),xh);
pdfsc=(xh(2)-xh(1))*nlag;
xpdf= normal_pdf(xh, mean(real(fb.clag(:,id))),(std(real(fb.clag(:,id)))**2))*pdfsc;
axis([-real(rlim), real(rlim), 0, max(yh)*1.1]);
gset term X11
gset output
figure(0);
bar(xh,yh);
hold on
plot(xh,xpdf)
gset term postscript eps 14
eval(sprintf("gset output ""%s_c%dd%d_lag_hist_re.%s"";",tname,ic,ida, plotfext));
replot
hold off

xh=-imag(rlim):(2*imag(rlim))/nbins:imag(rlim);
axis;
eval(sprintf("gset xtic %6.3f,%6.3f,%6.3f",-imag(rlim),2*imag(rlim)/nbintics,imag(rlim)));
gset nomxtics
eval(sprintf("title(""%s-behavioural, chain %d: %s, chip dump: %d, chain dump %d, imag."")",\
tname, ic, chainst, ida, id));
[yh,xh]=hist(imag(fb.clag(:,id)),xh);
pdfsc=(xh(2)-xh(1))*nlag;
xpdf= normal_pdf(xh, mean(imag(fb.clag(:,id))),(std(imag(fb.clag(:,id)))**2))*pdfsc;
axis([-imag(rlim), imag(rlim), 0, max(yh)*1.1]);
gset term X11
gset output
figure(1);
bar(xh,yh);
hold on
plot(xh,xpdf)
gset term postscript eps 14
eval(sprintf("gset output ""%s_c%dd%d_lag_hist_im.%s"";",tname,ic,ida, plotfext));
replot
hold off
gset term X11
gset output

endfunction
```

3.22.5 write_fd_plots

```
## desc: generate frequency domain plot files
## usage: write_fd_plots(ic,ida,idump,dump,fb,nlag,tname,
##                   bmin,bmax,rlim,rlimh,bw,Lwin,Cchain)
##     ic: chain number (first chain is 1)
##    ida: dump number (out of total dumps)
##  idump: index of chip dump for a given chain dump
##   dump: full dump structure
##     fb: full fb structure for chain ic: fb.clag(:,id)
##   nlag: number of lags
##  tname: file identifier
##   bmin: complex B model minimum axis
##   bmax: complex B model maximum axis
##   rlim: complex maximum of re, im difference axis
##  rlimh: complex maximum of re, im difference
##         histogram axis
## Cchain: mcb_reg structure
##   NOTE: "a" for bmin,bmax,rlim --> y autoscaling

## version: 2.11; 05-08-26

function write_fd_plots(ic,ida,idump,dump,fb,nlag,tname,bmin,bmax,rlim,rlimh,bw,Lwin, Cchain)

id= find(idump == ida);
if isempty(id)
               printf("\nERROR from write_lag_plots: Can't find dump index.\n");
               printf("Chain %d may not have participated in dump %d.\n",ic,ida);
               return
end

bfd= lag2fd(dump(ida).CCchain(ic).clag(:),bw,Lwin);
abm= max(abs(bfd));

if !isnumeric(bmin)
               bmin= min(10*log10(abs(bfd)));
endif
if !isnumeric(bmax)
```

```
              bmax= (10*log10(abm));
endif
if !isnumeric(rlim)
              rlim= max(abs(real(fb.fd(:,id))))+max(abs(imag(fb.fd(:,id))))*1i;
endif
if !isnumeric(rlimh)
              rlimh= rlim;
endif

chainst= sprintf("%d(%d:%d)%d %d(%d:%d)%d %d(%d:%d)%d %d(%d:%d)%d\n",\
Cchain(ic).Xinput,Cchain(ic).CCC(1), Cchain(ic).CCC(end),Cchain(ic).Yinput);

plotfext="eps";
lagnum= 0:(nlag-1);
# gset term X11
# gset output
gset term postscript eps 14
gset nokey
gset size ratio 0.3

freq= -(nlag/2-1):nlag/2;  # zero bin is zero frequency (d.c.)
eval(sprintf("gset xtics %d, %d, %d", freq(1)-1, nlag/8, freq(nlag)));
gset mxtics 2; # every second labelled tick
axis ([-nlag/2, nlag/2, bmin,bmax]);
xlabel("frequency [bin]");
ylabel("[dB]");
eval(sprintf("title(""%s RTL model, chain %d: %s, chip dump: %d, chain dump: %d, freq. domain mag."")",\
tname, ic, chainst, ida, id));
eval(sprintf("gset output ""%s_c%dd%d_FRE_RMOD_mag.%s"";",tname,ic,ida, plotfext));
plot(freq,10*log10(abs(bfd)/abm));

axis ([-nlag/2, nlag/2, -200,200]);
ylabel("[degrees]");
eval(sprintf("title(""%s RTL model, ch %d: %s, chip dump: %d, chain dump: %d, freq. domain phase."")",\
tname, ic, chainst, ida, id));
eval(sprintf("gset output ""%s_c%dd%d_FRE_RMOD_pha.%s"";",tname,ic,ida, plotfext));
plot(freq,180/pi*angle(bfd));

ylabel("");
axis ([-nlag/2, nlag/2, -real(rlim), real(rlim)]);
eval(sprintf("title(""%s-behavioural, chain %d: %s, chip dump: %d, chain dump: %d, freq. domain real"")",\
tname, ic, chainst, ida, id));
eval(sprintf("gset output ""%s_c%dd%d_FRE_RDIF_re.%s"";",tname,ic,ida, plotfext));
plot(freq, real(fb.fd(:,id)))

eval(sprintf("title(""%s-behavioural, chain %d: %s, chip dump: %d, chain dump: %d, freq. domain imag."")",\
tname, ic, chainst, ida, id));
axis ([-nlag/2, nlag/2, -imag(rlim), imag(rlim)]);
eval(sprintf("gset output ""%s_c%dd%d_FRE_RDIF_im.%s"";",tname,ic,ida, plotfext));
plot(freq, imag(fb.fd(:,id)))

# Generate histogram plots of the functional - behavioural model difference
nbins= 20; # number of bins
nbintics= 10; # number of bins to label with tics
xh=-real(rlimh):(2*real(rlimh))/nbins:real(rlimh);
axis;
eval(sprintf("gset xtic %6.4f,%6.4f,%6.4f",-real(rlimh),2*real(rlimh)/nbintics,real(rlimh)));
gset nomxtics
xlabel("");
ylabel("");
eval(sprintf("title(""%s-behavioural, chain %d: %s, chip dump: %d, chain dump: %d, freq. domain real."")",\
tname, ic, chainst, ida, id));
[yh,xh]=hist(real(fb.fd(:,id)),xh);
pdfsc=(xh(2)-xh(1))*nlag;
xpdf= normal_pdf(xh, mean(real(fb.fd(:,id))),(std(real(fb.fd(:,id)))**2))*pdfsc;
axis([-real(rlimh), real(rlimh), 0, max(yh)*1.1]);
gset term X11
gset output
figure(0);
bar(xh,yh);
hold on
plot(xh,xpdf)
gset term postscript eps 14
eval(sprintf("gset output ""%s_c%dd%d_FRE_HIST_re.%s"";",tname,ic,ida, plotfext));
replot
hold off

xh=-imag(rlimh):(2*imag(rlimh))/nbins:imag(rlimh);
axis;
eval(sprintf("gset xtic %6.4f,%6.4f,%6.4f",-imag(rlimh),2*imag(rlimh)/nbintics,imag(rlimh)));
gset nomxtics
eval(sprintf("title(""%s-behavioural, chain %d: %s, chip dump: %d, chain dump: %d, freq. domain imag."")",\
tname, ic, chainst, ida, id));
[yh,xh]=hist(imag(fb.fd(:,id)),xh);
pdfsc=(xh(2)-xh(1))*nlag;
xpdf= normal_pdf(xh, mean(imag(fb.fd(:,id))),(std(imag(fb.fd(:,id)))**2))*pdfsc;
axis([-imag(rlimh), imag(rlimh), 0, max(yh)*1.1]);
```

```
gset term X11
gset output
figure(1);
bar(xh,yh);
hold on
plot(xh,xpdf)
gset term postscript eps 14
eval(sprintf("gset output ""%s_c%dd%d_FRE_HIST_im.%s"";",tname,ic,ida, plotfext));
replot
hold off
gset term X11
gset output

endfunction
```

3.22.6 function write_fmod_fd_plots

```
## desc: generate frequency domain plot files for the
##       functional model
## usage: write_fmod_fd_plots(ic,id,dump,Cchain,tname,
##                            bmin,bmax,rlim,rlimh,bw,Lwin)
##     ic: chain number in dump (first chain is 1)
##     id: index of chip dump
##   dump: full dump structure
## Cchain: mcb chain structure
##  tname: file identifier
##   bmin: Magnitude axis min.
##   bmax: Magnitude axis max.
##  NOTE: "a" for bmin,bmax,rlim --> y autoscaling
##        -1 for Cchain.Yinput --> autocorr mode

## version: 2.00; 05-08-05

function [ffd]= write_fmod_fd_plots(ic,id,dump,Cchain,tname,bmin,bmax,bw,Lwin)

Version_date= "2.00: 05-08-05";
if nargin == 0
               ffd= Version_date;
               return;
endif

nlag= Cchain(ic).length*128;
ffd= lag2fd(dump(id).CCchain(ic).clag,bw,Lwin);
abm= max(abs(ffd));

if !isnumeric(bmin)
               bmin= min(10*log10(abs(ffd)));
endif
if !isnumeric(bmax)
               bmax= (10*log10(abm)+5);
endif

plotfext="eps";

lagnum= 0:(nlag-1);
# gset term X11
# gset output
gset term postscript eps 14
gset nokey
gset size ratio 0.3

freq= -(nlag/2-1):nlag/2;  # zero bin is zero frequency (d.c.)
eval(sprintf("gset xtics %d, %d, %d", freq(1)-1, nlag/8, freq(nlag)));
gset mxtics 2; # every second labelled tick
axis ([-nlag/2, nlag/2, bmin,bmax]);
xlabel("frequency [bin]");
ylabel("[dB]");

if Cchain(ic).Yinput < 0  # assume autocorr mode
               eval(sprintf("title(""%d lag functional model, freq. domain mag., dump: %d, chain %d(%d:%d)X"")",\
               nlag, id, Cchain(ic).Xinput,dump(id).CCchain(ic).CCC(1),dump(id).CCchain(ic).CCC(end)));
else

               eval(sprintf("title(""%d lag functional model, freq. domain mag., dump: %d, chain %d(%d:%d)%d"")",\
               nlag, id,
Cchain(ic).Xinput,dump(id).CCchain(ic).CCC(1),dump(id).CCchain(ic).CCC(end),Cchain(ic).Yinput));
endif
eval(sprintf("gset output ""%s_fc%dd%d_fd_mag.%s"";",tname,ic,id, plotfext));
plot(freq,10*log10(abs(ffd)/abm));

axis ([-nlag/2, nlag/2, -200,200]);
ylabel("[degrees]");
```

```
if Cchain(ic).Yinput < 0  # assume autocorr mode
               eval(sprintf("title(""%d lag functional model, freq. domain phase., dump: %d, chain %d(%d:%d)X"")",\
               nlag, id, Cchain(ic).Xinput,dump(id).CCchain(ic).CCC(1),dump(id).CCchain(ic).CCC(end)));
else
               eval(sprintf("gset output ""%s_fc%dd%d_fd_pha.%s"";",tname,ic,id, plotfext));
               eval(sprintf("title(""%d lag functional model, freq. domain phase., dump: %d, chain %d(%d:%d)%d"")",\
               nlag, id,
Cchain(ic).Xinput,dump(id).CCchain(ic).CCC(1),dump(id).CCchain(ic).CCC(end),Cchain(ic).Yinput));
endif
eval(sprintf("gset output ""%s_fc%dd%d_fd_pha.%s"";",tname,ic,id, plotfext));
plot(freq,180/pi*angle(ffd));

gset term X11
gset output

endfunction
```

3.22.7 function list_dump_ovr

```
##
##  usage: [out, nrateNdump, tsout, cellout, OVRout,
##                                idump, icell, icell_OVR]=
##           list_dump_ovr(frame, dump, dmult, rateN_cells, tstart)
##
## description: Returns strings and arrays to facilitate
##              inspection and tabulation of data from
##              frames in which dump overruns have occured.
##
## frame: output frame structure array
##        (obtained from get_lag_frames).
##
## dump:  dump structure array (from make_dump_index).
##
## dmult: dump rate factor (every dmult dumps at the
##        highest dump rate); highest dump rate = 1,
##        rate= base_dump_rate x (1/dmult).
##
## rateN_cells: row vector of cell numbers dumping
##              at the rate set by dmult.
##
## out:   single output string containing timestamp and
##        overrun information for the rateN_cells.
##
## nrateNdump: number of dumps at the rate set by dmult.
##
## tsout:   (nrateNdump,:) unterminated string array
##          containing the timestamp for each dump.
##
## cellout: (nrateNdump,:) unterminated string containing
##          dumped cell numbers expected for this timestamp.
##          Cells that were missed in this dump are
##          embedded within parenthesis.
##
## OVRout: (nrateNdump,:) unterminated string containing
##          the overrun (OVR) flags of cells in cellout.
##          Flags of missing cells are designated by 'x'.
##
## idump:  vector of length nrateNdump, containing indices
##         of structure array dump() that correspond to each
##         dump of the rateN_cells.
##
## icell:  (nrateNdump,:) array containing indices of
##         structure array dump.index() that correspond
##         to each dumped rateN_cell(k) with OVR reset.
##         An index of zero indicates that the cell number
##         found in the kth position of rateN_cells was
##         absent in a particular dump.
##
## icell_OVR: (nrateNdump,:) array containing indices of
##            structure array dump.index() that correspond
##            to each dumped rateN_cell(k) with OVR set.
##            An index of zero indicates that the cell
##            number found in the kth position of
##            rateN_cells was absent in this dump.
##
## NOTE: (1) only word 0 of timestamp is used; roll over
##           is not taken into account.
##       (2) this function will abort on ERROR if the
##           timestamps are not identical for all frames
##           of a particular dump.
##
## [out]= get_dump_ovr();
## Ver_date: version number: date this version was created
```

```
## ********************* Revision history *********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01: source code format tweaks; no operational change
## 1.10: add option to set start timestamp
## ************************************************************

%% frame= frame07;
%% dump=dump07;
%% dmult=10;
%% rateN_cells=rate3_cells;

Version_date= "1.10: 05-07-25";
if nargin == 0
              out= Version_date;
              return;
endif

if nargin < 5
              Lfirststamp= !1;  # The first timestamp is not supplied
else
              Lfirststamp= !0;
endif
out=""; # initialize output strings
tsout="";
cellout="";
OVRout="";

nrateN_cells= rows(rateN_cells); # the number of cells dumping at given rate dmult
ndump= columns(dump);
xtimestamp=zeros(ndump,1);

for i=1:ndump xtimestamp(i)=dump(i).TIMESTAMP_0; endfor;  # extract all timestamps dumped

if Lfirststamp
              starttime= tstart;
else
              starttime=min(xtimestamp);
endif
endtime= max(xtimestamp);
rateNts= (starttime+dmult-1):dmult:(endtime);              # vector of expected timestamps at rate dmult

nrateNts= columns(rateNts);                                # get the number of expected dumps for the given rate
icell= zeros(nrateNts, nrateN_cells);       # frame(dump.index(i)) indices of the cells without OVR set
icell_OVR= zeros(nrateNts, nrateN_cells);                  # frame(dump.index(i)) indices of the cells with OVR set
idump= zeros(nrateNts,1);                                  # indices of frame(dump(i).index) involving these cells

ratex=zeros(1,nrateNts);     # initialize an intermediate column vector to dump indices
for i=1:nrateNts ratex(i)= find(rateNts(i) == xtimestamp); endfor;  # locate timestamp, empty--> zero in ratex

r1dumps= ratex(find(ratex));            # extract indices of existing timestamps

idump_count= 0;                                                        # initialize counter for dumps containing
rateN_cells
for idu=1:ndump
              cells="";
              OVR_cells="";
              anycells= !1; # initialize flag that is true if rateN_cells occur within this dump

              if any(idu == r1dumps)
                            nindump= length(dump(idu).index);          # number of cells involved in this dump
                            dumped_cells= zeros(nindump,1);            # initialize all cells involved in this
dump
                            dumped_OVR_flags= !ones(nindump,1);        # initialize OVR flags for the above cells
                            idump(++idump_count)=idu;                  # increment the index of dumps involving
rateN_cells

                            for ikc=1:nindump
                                          dumped_cells(ikc)= frame(dump(idu).index(ikc)).CCC;
                                          dumped_OVR_flags(ikc)= frame(dump(idu).index(ikc)).status.OVR;
                            endfor;
                            dumped_OVR_cells= dumped_cells(dumped_OVR_flags);  # array of cell numbers with OVR
bits set

                            for ikd=1:nrateN_cells
                                          icell(idump_count,ikd)= find((dumped_cells == rateN_cells(ikd)) & !
dumped_OVR_flags);
                                          icell_OVR(idump_count,ikd)= find((dumped_cells == rateN_cells(ikd)) &...

dumped_OVR_flags);
                                          # NOTE: the array icell won't update if find returns empty,
                                          # hence make sure that icell is properly initialized!

                                          missing= (icell(idump_count,ikd) == 0) & (icell_OVR(idump_count,ikd) ==
0);
                                          OVR_set= any(dumped_OVR_cells == rateN_cells(ikd));
```

```
                                        if missing
                                                cells=[cells;"(",num2str(rateN_cells(ikd)),")"];
                                        else
                                                cells=[cells;num2str(rateN_cells(ikd))];
                                        endif
                                        if OVR_set & !missing
                                                OVR_cells=[OVR_cells;"1"];
                                        elseif !OVR_set & !missing
                                                OVR_cells=[OVR_cells;"0"];
                                        else
                                                OVR_cells=[OVR_cells;"x"];
                                        endif
                                endfor
                                anycells= !0;
                endif

                cells=[cells;"    "];  # pad with blanks to make all fields the same length
                OVR_cells=[OVR_cells;"    "];

                scells="";
                orcells="";
                if anycells
                                for ist=1:nrateN_cells
                                        scells=[scells,cells(ist,:)];
                                        orcells=[orcells,OVR_cells(ist,:)];
                                endfor
                                scells=[scells,cells(nrateN_cells+1,:)];       # include the blank field
                                orcells=[orcells,OVR_cells(nrateN_cells+1,:)];

                                timestamp_0= zeros(nindump,1);
                                for itts= 1:nindump timestamp_0(itts)= frame(dump(idu).index(itts)).TIMESTAMP_0;
endfor;

                                if any(diff(timestamp_0))                   # this error is highly unlikely at this
point
                                        printf("ERROR from get_dump_ovr:\n");
                                        printf("Time stamps for all cells of dump %d are NOT the same.
Returning.\n",idu);
                                        return
                                endfor

                                out= [out, sprintf("%d: %s\nset_OVR:   %s\n\n", timestamp_0(1), scells, orcells)];
                                tsout= [tsout; sprintf("%d", timestamp_0(1))];
                                cellout= [cellout; scells];
                                OVRout= [OVRout; orcells];
                endif

endfor

endfunction
```

3.22.8 DS14_diff_phase

```
# examine differential X-Y phase over the integration
# of test case 32.
#
[xamp, xpha, xdva, xnsf, xndva]= get_CCtest_data("X_CCinput0_DS14.vhex");
[yamp, ypha, ydva, ynsf, yndva]= get_CCtest_data("Y_CCinput0_DS14.vhex");
[xsamp, xspha, xsdva, xsnsf, xsndva]= get_CCtest_data("X_CCinput0save32.vhex");
[ysamp, yspha, ysdva, ysnsf, ysndva]= get_CCtest_data("Y_CCinput0save32.vhex");

axis([1,10000,-6,12])
gset xtic 0,1024,9216
figure(0)
plot(xpha(1:10000)-ypha(2:10001))

figure(1)
plot(xspha(1:10000)-yspha(2:10001))
size(find(yamp(1:10000)-ysamp(2:10001)))
```

3.22.9 write_diff_phase_plots_tc32

```
# generate X-Y differential plot file for the datasaver output, stream 0.

plotfext="eps";
tname="tc32";

# gset term X11
# gset output

gset nokey
gset xtics 2500,500, 9000
gset mxtics 2; # every second labelled tick
gset size ratio 0.3

xys_diff=xspha(2500:9000)-yspha(2500:9000);
ineg=find(xys_diff <0);
xys_diff(ineg)=xys_diff(ineg)+16;

axis ([2500, 9000, 0,16]);
xlabel("sample index");
title("X-Y differential phase of DS14-S (X,Y stream 0)");
plot([2500:9000], xys_diff);

hold
plot([2500:9000], xsdva(2500:9000));
eval(sprintf("gset output ""%s_XY_diff_phase.%s"";",tname, plotfext));
gset term postscript eps 14
replot
hold off

gset term X11
gset output
```

3.23 function check_register_capture

```
## desc: This function checks consistentcy:
##       (1) between status reporting within
##           output frames, MCSR and STATUS registers;
##       (2) between cells and dumps of data
##           captured from input streams and control lines;
##       (3) between cells and dumps of data
##           of data bias;
##       (4) between captured and expected stream data;
##       (5) of MCSR and STATUS register settings against a
##           template of expected values.
##
## usage: [Lwarn, Ladvis] = check_register_capture
##        (dump,mcb,Cchain,applied_dump,filename,
##        test_SID_X, test_SID_Y, base_SBID_X,
##        base_SBID_Y, test_case, test_mcb);
##
##         dump: dump structure created by add_CCchain_dump.
##          mcb: mcb structure created by get_mcb_data.
##       Cchain: chain desc. structure from make_mcb_chains.
## applied_dump: row or col. vector of 8 dump intervals
##               (in samples) [stream1,stream2...,stream8].
##     filename: body (no ext.) of output files;
##                               (log and bias tables).
##    test_case: test case number (for BBID checking)
##     test_mcb: optional mcb template; default otherwise.
##  Lwarn: boolean; true if warnings are issued.
## Ladvis: boolean; true if advisories are issued.
##
##  output report is placed in a file named
##                             "filename_register_log.txt"
##  bias table is placed in a file named
##                             "filename_bias_table.txt"
##
##    A WARNING indicates that some tests report
##                   an unexpected result (in most cases);
##
##    An ADVISORY indicates that although a particular
##    test fails, a failure is expected for some test cases.
##
##    check OK: the test was performed with no warning
##              or advisory condition.
##
## NOTE: (1) detection of integration dead time
##           inconsistency only applies to the single dump
##           case; problem detection in multiple dump cases
##           rely on a difference in bias between dumps
##           of a given chain.
##       (2) when a mcb template is supplied, the difference
##           between actual mcb and template flag is changed
##           from advisory to warning.
##       (3) Depending on the test case and data, some
##           warning conditions may preclude some other
##           checks. Hence, this function is not intended
##           as a diagnostic tool; it is a warning system.
##
## default template mcb (all reset):
##   test_mcb.MCSR.AOV= !1;
##   test_mcb.MCSR.OVR= !1;
##   test_mcb.MCSR.SyncER= !1;
##   test_mcb.MCSR.TvER= !1;
##   test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.X_DUMP_SYNC= !1;
##   test_mcb.Y_DUMP_SYNC= !1;
##   test_mcb.X_SCHID_SYNC= !1;
##   test_mcb.Y_SCHID_SYNC= !1;
##   test_mcb.X_TIMESTAMP= !1;
##   test_mcb.Y_TIMESTAMP= !1;
##   test_mcb.X_DATA= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.Y_DATA= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.X_PHASE= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.X_DVALID= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0] & !0;
##   test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0] & !0;
##
##
## [Ver_date]= check_register_capture();
## Ver_date: version number: date this version was created

## ********************** Revision history **********************
## (version; date created: description of changes)
## 1.00: first working version
```

```
## 2.00: July 01, 2005: added check for expected contents of data
##                      captured from streams.
## 3.10: Aug. 26, 2005: made checking of captured BBID compatible
##                      with latest test bench; fixed stream
##                      number reporting in output log.
## 3.20; Sep. 20, 2005: added missing advisory flag setting line
##
## ************************************************************

function [Lwarn, Ladvis] = check_register_capture(dump, mcb, Cchain, applied_dump, filename,\
test_SID_X, test_SID_Y, base_SBID_X, base_SBID_Y, tc, test_mcb)

Version_date= "3.20: 05-09-20";
if nargin == 0
               Lwarn= Version_date;
               return;
endif

if nargin < 11  # default template mcb (all reset):
               test_mcb.MCSR.AOV= !1;
               test_mcb.MCSR.OVR= !1;
               test_mcb.MCSR.SyncER= !1;
               test_mcb.MCSR.TvER= !1;
               test_mcb.X_DUMP_EN= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.Y_DUMP_EN= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.X_DUMP_SYNC= !1;
               test_mcb.Y_DUMP_SYNC= !1;
               test_mcb.X_SCHID_SYNC= !1;
               test_mcb.Y_SCHID_SYNC= !1;
               test_mcb.X_TIMESTAMP= !1;
               test_mcb.Y_TIMESTAMP= !1;
               test_mcb.X_DATA= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.Y_DATA= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.X_PHASE= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.Y_PHASE= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.X_DVALID= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.Y_DVALID= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.X_SE_CLK= [0;0;0;0;0;0;0;0] & !0;
               test_mcb.Y_SE_CLK= [0;0;0;0;0;0;0;0] & !0;
               flgstr= "ADVISORY"; LWARN = !1;  # level of advice in message; set LWARN for warning level
else
               if !(isstruct(test_mcb) && isfield(test_mcb,"MCSR") & isfield(test_mcb,"X_DATA"))
                              printf("\nERROR from check_register_capture: unidentified\n\
                               input structure for test_mcb\n");
                               return
               endif
               flgstr= "WARNING"; LWARN = !0;
endif
if !isnumeric(tc)
               printf("\nERROR from check_register_capture: non numeric\n\
               input for test_case; perhaps a call to an older version of\n\
               this function.\n");
               return
endif
if !(isstruct(mcb) && isfield(mcb,"MCSR") & isfield(mcb,"X_DATA"))
               printf("\nERROR from check_register_capture: unidentified\n\
               input structure for mcb\n");
               return
endif
if !(isstruct(Cchain) && isfield(Cchain,"CCC") & isfield(Cchain,"Xinput") & isfield(Cchain,"Yinput"))
               printf("\nERROR from check_register_capture: unidentified\n\
               input structure for Cchain\n");
               return
endif
if !(all(size(applied_dump) == [8,1]) | all(size(applied_dump) == [1,8]))
               printf("\nERROR from check_register_capture: problem with the input\n\
               list. Check size of applied_dump.\n");
               return;
endif
if all(size(applied_dump) == [1,8])
               applied_dump= applied_dump';
endif

# filename= "tcxx";
# applied_dump= [8000;8000;0;0;0;0;0;0];

CS= "|";
BIAS_PER_SAMPLE= 64;
iactive= find(applied_dump);
base_holdoff= 0;  # number of samples to be used as a base holdoff within table.
Lbias_table= !0;
Ladvis= !1;
Lwarn= !1;
logfspec= [filename,"_register_log.txt"];
biasfspec= [filename,"_bias_table.txt"];

## Check the input data structure and determine if either a dump or frame,
```

```
## return in the case of an unidentified structure.
if isstruct(dump) & isfield(dump,"CCchain") & isfield(dump,"TIMESTAMP_0") & isfield(dump,"TIMESTAMP_1")
               Ldump_chain= !0;

elseif isstruct(dump) & !isfield(dump,"CCchain") & isfield(dump,"TIMESTAMP_0") &...
            isfield(dump,"TIMESTAMP_1") & isfield(dump,"CCC") & isfield(dump,"bad_checksum")
               Ldump_chain= !1;

elseif isstruct(dump) & !isfield(dump,"CCchain") & isfield(dump,"TIMESTAMP_0") &...
          isfield(dump,"TIMESTAMP_1") & isfield(dump,"index")
               Ldump_chain= [];
               printf("\nERROR from check_register_capture: Chains not appended to dump stucture.\n");
               printf("Invoke function add_CCchain_dump.\n");
               printf("Returning.\n");
               return
else
               Ldump_chain=[];
               printf("\nERROR from check_register_capture: unidentified input \n dump or frame structure.\n");
               return
endif

if Ldump_chain # dump structure with appended chains
               ndump=columns(dump);
               maxchains= columns(Cchain);
               AOVchk= !ones(ndump,maxchains);  # 1 -> AOV of ANY cell in ANY chain disagrees with the MCSR AOV
               OVRchk= !ones(ndump,maxchains);  # 1 -> OVR of ANY cell in ANY chain disagrees with the MCSR OVR
               AOVchk_chain= !ones(ndump,maxchains);  # 1 -> some but NOT ALL AOV of cells in ANY chain are set
               OVRchk_chain= !ones(ndump,maxchains);  # 1 -> some but NOT ALL OVR of cells in ANY chain are set
               XSync_statchk= !ones(ndump,maxchains); # 1 -> setting of an XSTATUS bit disagrees with the output
                                                         frame
               YSync_statchk= !ones(ndump,maxchains); # 1 -> setting of an YSTATUS bit disagrees with the output
                                                         frame
               XSync_err= !ones(ndump,maxchains);
               YSync_err= !ones(ndump,maxchains);
               XSyncerrchk_chain= !ones(ndump,maxchains); # 1 -> XSyncerr of cells in ANY chain are not identically
                                                         set
               YSyncerrchk_chain= !ones(ndump,maxchains); # 1 -> YSyncerr of cells in ANY chain are not identically
                                                         set
               input_bias_dump= zeros(ndump, 8); # input bias of cells (no overrun) associated with each input stream
               input_bias_OVR_dump= zeros(ndump,8); # input bias of cells (overrun) associated with each input stream
               input_bias_OVR= zeros(8,1); # input bias associated with each stream; consistent between cells of
                                                         chain
               sid_X= zeros(ndump, maxchains);      # SID-X data associated with a given chain
               sid_Y= zeros(ndump, maxchains);      # SID-Y data associated with a given chain
               bbid_X= zeros(ndump, maxchains);     # BBID-X data associated with a given chain
               bbid_Y= zeros(ndump, maxchains);     # BBID-Y data associated with a given chain
               sbid_X= zeros(ndump, maxchains);     # SBID-X data associated with a given chain
               sbid_Y= zeros(ndump, maxchains);     # SBID-Y data associated with a given chain
               rcblk_X= zeros(ndump, maxchains);    # RECIRC_BLK_X data associated with a given chain
               rcblk_Y= zeros(ndump, maxchains);    # RECIRC_BLK_Y data associated with a given chain
               pbin= zeros(ndump, maxchains);       # Phase bin data associated with a given chain
               cmmd= zeros(ndump, maxchains);       # decimal form dump command associated with a given chain
               idumped= !ones(ndump,maxchains);     # flag chains dumped in a given dump

               inX= !ones(8,maxchains);             # association matrix: X-input data streams with chains
               inY= !ones(8,maxchains);             # association matrix: Y-input data streams with chains
               for id=1:ndump
                              nchains= columns(dump(id).CCchain);
                              for ich= 1:nchains
                                             # identify the inputs for this chain
                                             thchain= zeros(columns(Cchain),1);
                                             for ik=1:columns(Cchain)
                                                            thchain(ik)= (Cchain(ik).CCC(1) ==
dump(id).CCchain(ich).CCC(1))...
                                                             && all(Cchain(ik).CCC == dump(id).CCchain(ich).CCC);
                                             endfor

                                             # ich is the index of a chain within a given dump; ithchain is the index
                                                            of
                                             # a chain for the chip. Not all chains are represented in every dump.
                                             ithchain= find(thchain);
                                             Xinput= Cchain(ithchain).Xinput;
                                             Yinput= Cchain(ithchain).Yinput;

                                             # This section checks for consistency between the MCSR register and
                                             # corresponding status bits within the output lag frames.
                                             AOVchk(id,ithchain)= any(dump(id).CCchain(ich).ACC_OV);
                                             OVRchk(id,ithchain)= any(dump(id).CCchain(ich).OVR);
                                             Syncerrchk(id, ithchain)= (any(dump(id).CCchain(ich).XSyncerr)...
                                                            | any(dump(id).CCchain(ich).YSyncerr));

                                             # This section checks if output frame status bits are set identically
                                             # between each cell of a chain, for all chains and all dumps.
                                             AOVchk_chain(id,ithchain)= any(dump(id).CCchain(ich).ACC_OV) ...
                                                            & !all(dump(id).CCchain(ich).ACC_OV);
                                             OVRchk_chain(id,ithchain)= any(dump(id).CCchain(ich).OVR) ...
                                                            & !all(dump(id).CCchain(ich).OVR);
```

```
                                        XSyncerrchk_chain(id,ithchain)= any(dump(id).CCchain(ich).XSyncerr)...
                                                & !all(dump(id).CCchain(ich).XSyncerr);
                                        YSyncerrchk_chain(id,ithchain)= any(dump(id).CCchain(ich).YSyncerr)...
                                                & !all(dump(id).CCchain(ich).YSyncerr);

                                        # This section checks for consistency between X|Y-STATUS registers and
                                        # corresponding status bits within the output lag frames.
                                        # NOTE: X-Y offset on SCHID_FRAME and DUMP_SYNC are NOT reported
                                        # within the output frame (according to the RFS document).
                                        XSync_err(id,ithchain)= any(dump(id).CCchain(ich).XSyncerr);
                                        XSync_statchk(id,ithchain)= (any(mcb.X_DATA(Xinput+1,:))|...
                                        any(mcb.X_PHASE(Xinput+1,:))|...
                                        mcb.X_DVALID(Xinput+1)|...
                                        mcb.X_DUMP_EN(Xinput+1)|...
                                        mcb.X_SE_CLK(Xinput+1)|...
                                        mcb.X_TIMESTAMP) != XSync_err(id,ithchain);

                                        YSync_err(id,ithchain)= any(dump(id).CCchain(ich).YSyncerr);
                                        YSync_statchk(id,ithchain)= (any(mcb.Y_DATA(Yinput+1,:))|...
                                        any(mcb.Y_PHASE(Yinput+1,:))|...
                                        mcb.Y_DVALID(Yinput+1)|...
                                        mcb.Y_DUMP_EN(Yinput+1)|...
                                        mcb.Y_SE_CLK(Yinput+1)|...
                                        mcb.Y_TIMESTAMP) != YSync_err(id,ithchain);


                                        # This section tabulates the DATA_BIAS for cells with and without
                                        # OVR flags set.

                                        inbias=
dump(id).CCchain(ich).DATA_BIAS(find(!dump(id).CCchain(ich).OVR));
                                        inbias_OVR=
dump(id).CCchain(ich).DATA_BIAS(find(dump(id).CCchain(ich).OVR));

                                        # Check that all data_bias in the chain are the same for a particular
                                                OVR state.
                                        # This check is not really required since scale_lag_data will fail on
                                                error.
                                        # if the above condition occurs.
                                        if !isempty(inbias) && any(inbias(1)-inbias)
                                                inbias= -1;
                                        endif
                                        if !isempty(inbias_OVR) && any(inbias_OVR(1)-inbias_OVR)
                                                inbias_OVR= -1;
                                        endif

                                        if isempty(inbias) inbias=0; endif;
                                        if isempty(inbias_OVR) inbias_OVR=0; endif;
                                        if mcb.MCSR.XYDS  # input selected on the basis of dump control
                                                input_bias_dump(id,Xinput+1)= inbias(1);
                                                input_bias_OVR_dump(id,Xinput+1)= inbias_OVR(1);
                                        else
                                                input_bias_dump(id,Yinput+1)= inbias(1);
                                                input_bias_OVR_dump(id,Yinput+1)= inbias_OVR(1);
                                        endif

                                        # Check captured SID data for consistency between cells of a chain and
                                                store
                                        # in array for further checks.
                                        # Arrays inX and inY associate input streams with chains.
                                        inX(Xinput+1,ithchain)= !0;
                                        inY(Yinput+1,ithchain)= !0;
                                        if all(dump(id).CCchain(ich).SID_X(1) == dump(id).CCchain(ich).SID_X)
                                                sid_X(id,ithchain)= dump(id).CCchain(ich).SID_X(1);
                                        else
                                                sid_X(id,ithchain)= -1; # differ within chain
                                        endif
                                        if all(dump(id).CCchain(ich).SID_Y(1) == dump(id).CCchain(ich).SID_Y)
                                                sid_Y(id,ithchain)= dump(id).CCchain(ich).SID_Y(1);
                                        else
                                                sid_Y(id,ithchain)= -1; # differ within chain
                                        endif
                                        if all(dump(id).CCchain(ich).SBID_X(1) == dump(id).CCchain(ich).SBID_X)
                                                sbid_X(id,ithchain)= dump(id).CCchain(ich).SBID_X(1);
                                        else
                                                sbid_X(id,ithchain)= -1; # differ within chain
                                        endif
                                        if all(dump(id).CCchain(ich).SBID_Y(1) == dump(id).CCchain(ich).SBID_Y)
                                                sbid_Y(id,ithchain)= dump(id).CCchain(ich).SBID_Y(1);
                                        else
                                                sbid_Y(id,ithchain)= -1; # differ within chain
                                        endif
                                        if all(dump(id).CCchain(ich).BBID_X(1) == dump(id).CCchain(ich).BBID_X)
                                                bbid_X(id,ithchain)= dump(id).CCchain(ich).BBID_X(1);
                                        else
                                                bbid_X(id,ithchain)= -1; # differ within chain
                                        endif
```

```
                                        if all(dump(id).CCchain(ich).BBID_Y(1) == dump(id).CCchain(ich).BBID_Y)
                                                bbid_Y(id,ithchain)= dump(id).CCchain(ich).BBID_Y(1);
                                        else
                                                bbid_Y(id,ithchain)= -1; # differ within chain
                                        endif
                                        if all(dump(id).CCchain(ich).RECIRC_BLK_X(1) ==
dump(id).CCchain(ich).RECIRC_BLK_X)
                                                rcblk_X(id,ithchain)=
dump(id).CCchain(ich).RECIRC_BLK_X(1);
                                        else
                                                rcblk_X(id,ithchain)= -1;  # differ within chain
                                        endif
                                        if all(dump(id).CCchain(ich).RECIRC_BLK_Y(1) ==
dump(id).CCchain(ich).RECIRC_BLK_Y)
                                                rcblk_Y(id,ithchain)=
dump(id).CCchain(ich).RECIRC_BLK_Y(1);
                                        else
                                                rcblk_Y(id,ithchain)= -1;  # differ within chain
                                        endif
                                        if all(dump(id).CCchain(ich).Phase_bin(1) ==
dump(id).CCchain(ich).Phase_bin)
                                                pbin(id,ithchain)= dump(id).CCchain(ich).Phase_bin(1);
                                        else
                                                pbin(id,ithchain)= -1;  # differ within chain
                                        endif
                                        cmmddec= bin2dec(dump(id).CCchain(ich).cmmd);
                                        if all(cmmddec == cmmddec(1))
                                                cmmd(id,ithchain)= cmmddec(1);
                                        else
                                                cmmd(id,ithchain)= -1;        # differ within chain
                                        endif
                                        idumped(id,ithchain)= !0;
                        endfor
                endfor

                fidlog= fopen(logfspec,"w");
                if mcb.MCSR.TvER & !mcb.MCSR.TVEN
                        fprintf(fidlog,"\nWARNING from check_register_capture: TvER bit within the MCSR is s
                                                et\n");
                        fprintf(fidlog,"          while test vector generation is disabled (TVEN reset).\n");
                        Lwarn= !0;
                else
                        fprintf(fidlog,"\nCheck OK: TvER bit within the MCSR and test vector enable (TVEN)\n");
                        fprintf(fidlog,"          are both reset.\n");
                endif

                if any(any(AOVchk)) == mcb.MCSR.AOV
                        fprintf(fidlog,"\nCheck OK: Accumulator overflow bit in the MCSR (AOV) is
                                                consistent\n");
                        fprintf(fidlog,"             with status bit settings in output frames.\n");
                else
                        fprintf(fidlog,"\nWARNING from check_register_capture: Accumulator overflow bit\n");
                        fprintf(fidlog,"          in the MCSR (AOV) is inconsistent with output frame\n");
                        fprintf(fidlog,"          status bit settings.\n");
                        Lwarn= !0;
                endif

                if any(any(OVRchk)) == mcb.MCSR.OVR
                        fprintf(fidlog,"\nCheck OK: Dump overrun bit in the MCSR (OVR) is consistent\n");
                        fprintf(fidlog,"             with status bit settings in output frames.\n");
                else
                        fprintf(fidlog,"\nWARNING from check_register_capture: Dump overrun bit\n");
                        fprintf(fidlog,"          in the MCSR (OVR) is inconsistent with output frame\n");
                        fprintf(fidlog,"          status bit settings.\n");
                        Lwarn= !0;
                endif
                if any(any(Syncerrchk))== mcb.MCSR.SyncER   # !any(any(Syncerrchk))
                        fprintf(fidlog,"\nCheck OK: Sync. error bit in the MCSR (SyncER) is consistent\n");
                        fprintf(fidlog,"             with status bit settings in output frames.\n");
                else
                        fprintf(fidlog,"\nWARNING from check_register_capture: Sync. error bit\n");
                        fprintf(fidlog,"          in the MCSR (SyncER) is inconsistent with output frame\n");
                        fprintf(fidlog,"          status bit settings.\n");
                        Lwarn= !0;
                endif
                if !any(any(XSyncerrchk_chain)) & !any(any(YSyncerrchk_chain))
                        fprintf(fidlog,"\nCheck OK: Synch. error status bit (X|YSyncer) settings in output\n");
                        fprintf(fidlog,"              frames are identical between cells of a chain.\n");
                else
                        fprintf(fidlog,"\nADVISORY from check_register_capture: Synch. error status bit\n");
                        fprintf(fidlog,"           (X|YSyncer) settings in an output frame are not
                                                identical\n");
                        fprintf(fidlog,"            between cells of a chain.\n");
                        Ladvis= !0;
                endif

                # Check the data bias for consistency between dumps and report.
```

```
            bias_dump_warn= !1;
            bias_chain_warn= !1;
            for inp=1:8
                        tib= input_bias_dump(find(input_bias_dump(:,inp)),inp);
                        tib_OVR= input_bias_OVR_dump(find(input_bias_OVR_dump(:,inp)),inp);
                        if !isempty(tib) && any(tib(1)-tib(:))
                                    fprintf(fidlog,"\nWARNING from check_register_capture: data bias of
cells with\n");
                                    fprintf(fidlog,"        no overrun differs between dumps.\n");
                                    fprintf(fidlog,"        Inspect chains fed with input stream %d.\n",
inp-1);
                                    Lwarn= !0;
                                    bias_dump_warn= !0;
                        endif
                        if !isempty(tib_OVR) && any(tib_OVR(1)-tib_OVR(:))
                                    fprintf(fidlog,"\nWARNING from check_register_capture: data bias of
cells with\n");
                                    fprintf(fidlog,"        dump overrun differs between dumps.\n");
                                    fprintf(fidlog,"        Inspect chains fed with input stream %d.\n",
inp-1);
                                    Lwarn= !0;
                                    bias_dump_warn= !0;
                        endif
                        if  any(input_bias_dump(:,inp) < 0)
                                    fprintf(fidlog,"\nWARNING from check_register_capture: data bias differs
between\n");
                                    fprintf(fidlog,"        cells of a common chain (same OVR status).\n");
                                    fprintf(fidlog,"        Inspect chains fed with input stream %d.\n",
inp-1);
                                    Lwarn= !0;
                                    bias_chain_warn= !0;
                        endif
                        if  any(input_bias_dump(:,inp) < 0) & mcb.MCSR.AOV
                                    fprintf(fidlog,"\nADVISORY from check_register_capture: data bias
differs between\n");
                                    fprintf(fidlog,"        cells of a common chain (same OVR status).\n");
                                    fprintf(fidlog,"        Inspect chains fed with input stream %d.\n",
inp-1);
                                    Ladvis= !0;
                                    bias_chain_warn= !0;
                        endif
                        if !isempty(tib)
                                    input_bias(inp)= tib(1)/BIAS_PER_SAMPLE;
                        else
                                    input_bias(inp)= 0;
                        endif
                        if !isempty(tib_OVR)
                                    input_bias_OVR(inp)= tib_OVR(1)/BIAS_PER_SAMPLE;
                        else
                                    input_bias_OVR(inp)= 0;
                        endif
            endfor
            if !bias_dump_warn
                        fprintf(fidlog,"\nCheck OK: data bias is consistent between dumps.\n");
            endif
            if !bias_chain_warn
                        fprintf(fidlog,"\nCheck OK: data bias is consistent over chains within a dump.\n");
            endif

            # Check and report consistency of integration dead time between cells fed by different data streams
            # Hold off will result in an advisory being generated.
            if any(input_bias)
                        streams_noovr= find(input_bias);   # streams with same bias on all cells; no OVR
                        if any((applied_dump(streams_noovr(1))-input_bias(streams_noovr(1)))-
(applied_dump(streams_noovr)-\
                        input_bias(streams_noovr)))
                                    fprintf(fidlog,"\nADVISORY from check_register_capture: integration dead
time differs\n");
                                    fprintf(fidlog,"        between streams; on cells reporting no
overrun.\n");
                                    Ladvis= !0;
                        else
                                    fprintf(fidlog,"\nCheck OK: integration dead time is consistent between
dump streams;\n");
                                    fprintf(fidlog,"          on cells reporting no overrun.\n");
                        endif
            endif
            if any(input_bias_OVR)
                        streams_ovr= find(input_bias_OVR); # streams with same bias on all cells; OVR set
                        if any((applied_dump(streams_ovr(1))-input_bias(streams_ovr(1)))-
(applied_dump(streams_ovr)-\
                        input_bias(streams_ovr)))
                                    fprintf(fidlog,"\nADVISORY from check_register_capture: integration dead
time differs\n");
                                    fprintf(fidlog,"        between streams; on cells reporting
overrun.\n");
                                    Ladvis= !0;
```

```
                            else
                                        fprintf(fidlog,"\nCheck OK: integration dead time is consistent between
dump streams;\n");
                                        fprintf(fidlog,"          on cells reporting overrun.\n");
                            endif
                endif

                # Check and report that data captured from the data streams and control lines
                # are identical between cells sharing the same stream.
                Lcapture_cells= !1;        # true if captured data from sdata differs between cells of a chain
                Lcapture_dumps= !1;        # true if captured data from sdata differs between dumps
                Lcapture_dccells= !1;      # true if captured data from dump lines differs between cells of a chain
                Lcapture_dcdumps= !1;      # true if captured data from from dump lines sdata differs between dumps
                Linputs= !1;               # true if there is at least one chain connected to X and Y inputs
                for ithchain= 1:maxchains   # check if the data for each chain is consistent between dumps
                            if any(sid_X(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured X-
Station ID (SID-X)\n");
                                        fprintf(fidlog,"      differs between cells of chain %d.\n", ithchain);
                                        Lwarn= !0;
                                        Lcapture_cells= !0;
                            endif
                            if any(sid_Y(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured Y-
Station ID (SID-Y)\n");
                                        fprintf(fidlog,"      differs between cells of chain %d.\n", ithchain);
                                        Lwarn= !0;
                                        Lcapture_cells= !0;
                            endif
                            if any(sbid_X(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured X-
Station subband\n");
                                        fprintf(fidlog,"      ID (SBID-X) differs between cells of chain %d.\n",
ithchain);
                                        Lwarn= !0;
                                        Lcapture_cells= !0;
                            endif
                            if any(sbid_Y(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured Y-
Station subband\n");
                                        fprintf(fidlog,"     ID (SBID-Y) differs between cells of chain %d.\n",
ithchain);
                                        Lwarn= !0;
                                        Lcapture_cells= !0;
                            endif
                            if any(bbid_X(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured X-
Station baseband\n");
                                        fprintf(fidlog,"      ID (BBID-X) differs between cells of chain %d.\n",
ithchain);
                                        Lwarn= !0;
                                        Lcapture_cells= !0;
                            endif
                            if any(bbid_Y(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured Y-
Station baseband\n");
                                        fprintf(fidlog,"      ID (BBID-Y) differs between cells of chain %d.\n",
ithchain);
                                        Lwarn= !0;
                                        Lcapture_cells= !0;
                            endif
                            if any(rcblk_X(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured X
recirc. block\n");
                                        fprintf(fidlog,"      differs between cells of chain %d.\n", ithchain);
                                        Lwarn= !0;
                                        Lcapture_dccells= !0;
                            endif
                            if any(rcblk_Y(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured Y
recirc. block\n");
                                        fprintf(fidlog,"      differs between cells of chain %d.\n", ithchain);
                                        Lwarn= !0;
                                        Lcapture_dccells= !0;
                            endif
                            if any(pbin(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured phase
bin\n");
                                        fprintf(fidlog,"      differs between cells of chain %d.\n", ithchain);
                                        Lwarn= !0;
                                        Lcapture_dccells= !0;
                            endif
                            if any(cmmd(:,ithchain) < 0)
                                        fprintf(fidlog,"\nWARNING from check_register_capture: Captured dump
command\n");
                                        fprintf(fidlog,"      string differs between cells of chain %d.\n",
ithchain);
```

```
                                Lwarn= !0;
                                Lcapture_dccells= !0;
                        endif

                        icdd= find(idumped(:,ithchain));  # get indices of chains dumped in this dump
                        if isempty(icdd)
                                continue;  # to handle chains that never dump
                        endif
                        if !all(sid_X(icdd(1),ithchain) == sid_X(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: Captured X-
Station ID (SID-X)\n");
                                fprintf(fidlog,"      of chain %d differs between dumps.\n", ithchain);
                                Lwarn= !0;
                                Lcapture_dumps= !0;
                        endif
                        if !all(sid_Y(icdd(1),ithchain) == sid_Y(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: Captured Y-
Station ID (SID-Y)\n");
                                fprintf(fidlog,"      of chain %d differs between dumps.\n", ithchain);
                                Lwarn= !0;
                                Lcapture_dumps= !0;
                        endif
                        if !all(sbid_X(icdd(1),ithchain) == sbid_X(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: Captured X-
Station subband\n");
                                fprintf(fidlog,"      ID (SBID-X) of chain %d differs between dumps.\n",
ithchain);
                                Lwarn= !0;
                                Lcapture_dumps= !0;
                        endif
                        if !all(sbid_Y(icdd(1),ithchain) == sbid_Y(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: Captured Y-
Station subband\n");
                                fprintf(fidlog,"      ID (SBID-Y) of chain %d differs between dumps.\n",
ithchain);
                                Lwarn= !0;
                                Lcapture_dumps= !0;
                        endif
                        if !all(bbid_X(icdd(1),ithchain) == bbid_X(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: X-Station
baseband\n");
                                fprintf(fidlog,"      ID (BBID-X) of chain %d differs between dumps.\n",
ithchain);
                                Lwarn= !0;
                                Lcapture_dumps= !0;
                        endif
                        if !all(bbid_Y(icdd(1),ithchain) == bbid_Y(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: Captured Y-
Station baseband\n");
                                fprintf(fidlog,"      ID (BBID-Y) of chain %d differs between dumps.\n",
ithchain);
                                Lwarn= !0;
                                Lcapture_dumps= !0;
                        endif

                        if !all(rcblk_X(icdd(1),ithchain) == rcblk_X(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: X recirc.
block\n");
                                fprintf(fidlog,"      of chain %d differs between dumps.\n", ithchain);
                                Lwarn= !0;
                                Lcapture_dcdumps= !0;
                        endif
                        if !all(rcblk_Y(icdd(1),ithchain) == rcblk_Y(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: Y recirc.
block\n");
                                fprintf(fidlog,"      of chain %d differs between dumps.\n", ithchain);
                                Lwarn= !0;
                                Lcapture_dcdumps= !0;
                        endif
                        if !all(pbin(icdd(1),ithchain) == pbin(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: phase bin\n");
                                fprintf(fidlog,"      of chain %d differs between dumps.\n", ithchain);
                                Lwarn= !0;
                                Lcapture_dcdumps= !0;
                        endif
                        if !all(cmmd(icdd(1),ithchain) == cmmd(icdd,ithchain))
                                fprintf(fidlog,"\nWARNING from check_register_capture: dump command
string\n");
                                fprintf(fidlog,"      of chain %d differs between dumps.\n", ithchain);
                                Lwarn= !0;
                                Lcapture_dcdumps= !0;
                        endif
                        if any(inX(:,ithchain)) & any(inY(:,ithchain))
                                Linputs = !0;
                        endif
                endfor
```

```
            if (mcb.MCSR.XYDS & any(any(rcblk_Y))) | (!mcb.MCSR.XYDS & any(any(rcblk_X)))
                        fprintf(fidlog,"\nWARNING from check_register_capture: non-zero recirc. block on\n");
                        fprintf(fidlog,"      non-active dump contol line.\n");
                        Lwarn= !0;
            else
                        fprintf(fidlog,"\nCheck OK: non-zero recirc. block is on active dump control.\n");
            endif

            if !Linputs
                        fprintf(fidlog,"\nWARNING from check_register_capture: Some chains are not\n\
                         connected to input streams.\n");
                        Lwarn= !0;
            endif

            if !Lcapture_dumps & !Lcapture_cells & Linputs
                        Lcapture_OK = !0;  # captured data OK thus far
                        fprintf(fidlog,"\nCheck OK: Captured X and Y station, subband and baseband IDs\n");
                        fprintf(fidlog,"          (SID, SBID, BBID) are consistent between dumps and\n");
                        fprintf(fidlog,"          between cells of a chain.\n");
                        for inj=1:maxchains
                                    iddx=find(inX(:,inj));  #
                                    iddy=find(inY(:,inj));  #
                        endfor
                        Lcapture_input= !1;
                        for ix=1:8  # check if data is consistent between chains fed by common input streams
                                    common_Xinput_index= find(inX(ix,:));
                                    common_Yinput_index= find(inY(ix,:));
                                    if !isempty(common_Xinput_index) && ...
                                                !all(sid_X(1,common_Xinput_index(1)) ==
sid_X(1,common_Xinput_index))
                                                fprintf(fidlog,"\nWARNING from check_register_capture:
Captured X-Station ID\n");
                                                fprintf(fidlog,"       (SID-X)differs between chains with
common input data stream.\n");
                                                fprintf(fidlog,"       Check chains associated with data
stream %d.\n", ix-1);
                                                Lwarn= !0;
                                                Lcapture_input= !0;
                                                Lcapture_OK = !1; #
                                    endif
                                    if !isempty(common_Yinput_index) && ...
                                                !all(sid_Y(1,common_Yinput_index(1)) ==
sid_Y(1,common_Yinput_index))
                                                fprintf(fidlog,"\nWARNING from check_register_capture:
Captured Y-Station ID\n");
                                                fprintf(fidlog,"       (SID-Y) differs between chains with
common input data stream.\n");
                                                fprintf(fidlog,"       Check chains associated with Y data
stream %d.\n", ix-1);
                                                Lwarn= !0;
                                                Lcapture_input= !0;
                                                Lcapture_OK = !1; #
                                    endif
                        endfor
                        if !Lcapture_input
                                    fprintf(fidlog,"\nCheck OK: Captured X and Y Station ID (SID) are
consistent\n");
                                    fprintf(fidlog,"          between chains with common input data
streams.\n");
                        endif
                        Lcapture_input= !1;
                        for ix=1:8  # check if data is consistent between chains fed by common input streams
                                    common_Xinput_index= find(inX(ix,:));
                                    common_Yinput_index= find(inY(ix,:));
                                    if !isempty(common_Xinput_index) && ...
                                                !all(sbid_X(1,common_Xinput_index(1)) ==
sbid_X(1,common_Xinput_index))
                                                fprintf(fidlog,"\nWARNING from check_register_capture:
Captured X-Station subband ID\n");
                                                fprintf(fidlog,"       (SBID-X)differs between chains with
common input data stream.\n");
                                                fprintf(fidlog,"       Check chains associated with data
stream %d.\n", ix-1);
                                                Lwarn= !0;
                                                Lcapture_input= !0;
                                                Lcapture_OK = !1; #
                                    endif
                                    if !isempty(common_Yinput_index) && ...
                                                !all(sbid_Y(1,common_Yinput_index(1)) ==
sbid_Y(1,common_Yinput_index))
                                                fprintf(fidlog,"\nWARNING from check_register_capture:
Captured Y-Station subband ID\n");
                                                fprintf(fidlog,"       (SBID-Y) differs between chains
with common input data stream.\n");
                                                fprintf(fidlog,"       Check chains associated with Y data
stream %d.\n", ix-1);
                                                Lwarn= !0;
```

```
                                                Lcapture_input= !0;
                                                Lcapture_OK = !1; #
                                        endif
                                endfor
                                if !Lcapture_input
                                        fprintf(fidlog,"\nCheck OK: Captured X and Y Station subband IDs (SBID)
are consistent\n");
                                        fprintf(fidlog,"          between chains with common input data
streams.\n");
                                endif
                                Lcapture_input= !1;
                                for ix=1:8  # check if data is consistent between chains fed by common input streams
                                        common_Xinput_index= find(inX(ix,:));
                                        common_Yinput_index= find(inY(ix,:));
                                        if !isempty(common_Xinput_index) && ...
                                                !all(bbid_X(1,common_Xinput_index(1)) ==
bbid_X(1,common_Xinput_index))
                                                fprintf(fidlog,"\nWARNING from check_register_capture:
Captured X-Station baseband ID\n");
                                                fprintf(fidlog,"       (BBID-X) differs between chains
with common input data stream.\n");
                                                fprintf(fidlog,"       Check chains associated with data
stream %d.\n", ix-1);
                                                Lwarn= !0;
                                                Lcapture_input= !0;
                                                Lcapture_OK = !1; #
                                        endif
                                        if !isempty(common_Yinput_index) && ...
                                                !all(bbid_Y(1,common_Yinput_index(1)) ==
bbid_Y(1,common_Yinput_index))
                                                fprintf(fidlog,"\nWARNING from check_register_capture:
Captured Y-Station baseband ID\n");
                                                fprintf(fidlog,"       (BBID-Y) differs between chains
with common input data stream.\n");
                                                fprintf(fidlog,"       Check chains associated with Y data
stream %d.\n", ix-1);
                                                Lwarn= !0;
                                                Lcapture_input= !0;
                                                Lcapture_OK = !1; #
                                        endif
                                endfor
                                if !Lcapture_input
                                        fprintf(fidlog,"\nCheck OK: Captured X and Y Station baseband IDs (BBID)
are consistent\n");
                                        fprintf(fidlog,"          between chains with common input data
streams.\n");
                                endif

                                 # check values against expected if consistent over cells and chains
                                if Lcapture_OK
                                        for inj= 1:maxchains  #
                                                iddx= find(inX(:,inj));
                                                iddy= find(inY(:,inj));
                                                if isempty(iddx) | isempty(iddy)
                                                        fprintf(fidlog,"\nADVISORY from
check_register_capture: Chain of\n");
                                                        fprintf(fidlog,"index %d is not dumped.
Skipping check of\n",inj);
                                                        fprintf(fidlog,"captured data for this
chain.\n");
                                                        Ladvis= !0;
                                                else
                                                        if !all(sid_X(find(idumped(:,inj)),inj) ==
test_SID_X) |\
                                                                !
all(sid_Y(find(idumped(:,inj)),inj) == test_SID_Y)
fprintf(fidlog,"\nWARNING from check_register_capture: Captured Station IDs\n");
fprintf(fidlog,"       (SID) differ from expected values.\n");
fprintf(fidlog,"       Check chains associated with data streams X:%d and Y:%d.\n", iddx-1, iddy-1);
                                                                Lwarn= !0;
                                                                Lcapture_OK = !1;
                                                        endif
                                                        if !all(sbid_X(find(idumped(:,inj)),inj) ==
iddx-1+base_SBID_X) |\
                                                                !
all(sbid_Y(find(idumped(:,inj)),inj) == iddy-1+base_SBID_Y)
fprintf(fidlog,"\nWARNING from check_register_capture: Captured Station subband IDs\n");
fprintf(fidlog,"       (SBID) differ from expected values.\n");
fprintf(fidlog,"       Check chains associated with data streams X:%d and Y:%d.\n", iddx-1, iddy-1);
                                                                Lwarn= !0;
                                                                Lcapture_OK = !1;
                                                        endif
                                                        if !all(bbid_X(find(idumped(:,inj)),inj) ==
mod(iddx-1+tc,8)) |\
                                                                !
all(bbid_Y(find(idumped(:,inj)),inj) == mod(iddy-1+tc,8))
fprintf(fidlog,"\nWARNING from check_register_capture: Captured Station baseband IDs\n");
```

```
fprintf(fidlog,"       (BBID) differ from expected values.\n");
fprintf(fidlog,"       Check chains associated with data streams X:%d and Y:%d.\n", iddx-1, iddy-1);
                                                                                    Lwarn= !0;
                                                                                    Lcapture_OK = !1;
                                                                    endif
                                                        endif
                                                endfor
                                                if Lcapture_OK
                                                        fprintf(fidlog,"\nCheck OK: Captured X,Y ID data (SID,
SBID, BBID) are set\n");
                                                        fprintf(fidlog,"          to expected values.\n");
                                                endif
                                        endif
                        endif
                        if !Lcapture_dcdumps & !Lcapture_dccells & Linputs
                                fprintf(fidlog,"\nCheck OK: Captured X,Y reciculation blocks, phase bins and dump\n");
                                fprintf(fidlog,"           command are consistent between dumps and between\n");
                                fprintf(fidlog,"           cells of chains.\n");
                                Lcapture_input= !1;
                                for ix=1:8  # check if data is consistent between chains fed by common input streams
                                        common_Xinput_index= find(inX(ix,:));
                                        common_Yinput_index= find(inY(ix,:));
                                        if !isempty(common_Xinput_index) && ...
                                                        !all(rcblk_X(1,common_Xinput_index(1)) ==
rcblk_X(1,common_Xinput_index))
                                                        fprintf(fidlog,"\nWARNING from check_register_capture: X

recirc. block\n");
                                                        fprintf(fidlog,"       differs between chains with common

input data stream.\n");
                                                        fprintf(fidlog,"       Check chains associated with data

stream %d.\n", ix-1);
                                                        Lwarn= !0;
                                                        Lcapture_input= !0;
                                        endif
                                        if !isempty(common_Yinput_index) && ...
                                                        !all(rcblk_Y(1,common_Yinput_index(1)) ==
rcblk_Y(1,common_Yinput_index))
                                                        fprintf(fidlog,"\nWARNING from check_register_capture: Y

recirc. block\n");
                                                        fprintf(fidlog,"       differs between chains with common

input data stream.\n");
                                                        fprintf(fidlog,"       Check chains associated with Y data

stream %d.\n", ix-1);
                                                        Lwarn= !0;
                                                        Lcapture_input= !0;
                                        endif
                                        if !isempty(common_Yinput_index) && ...
                                                        !all(pbin(1,common_Yinput_index(1)) ==
pbin(1,common_Yinput_index))
                                                        fprintf(fidlog,"\nWARNING from check_register_capture:

phase bin\n");
                                                        fprintf(fidlog,"       differs between chains with common

input data stream.\n");
                                                        fprintf(fidlog,"       Check chains associated with Y data

stream %d.\n", ix-1);
                                                        Lwarn= !0;
                                                        Lcapture_input= !0;
                                        endif
                                        if !isempty(common_Yinput_index) && ...
                                                        !all(cmmd(1,common_Yinput_index(1)) ==
cmmd(1,common_Yinput_index))
                                                        fprintf(fidlog,"\nWARNING from check_register_capture:

dump command\n");
                                                        fprintf(fidlog,"       differs between chains with common

input data stream.\n");
                                                        fprintf(fidlog,"       Check chains associated with Y data

stream %d.\n", ix-1);
                                                        Lwarn= !0;
                                                        Lcapture_input= !0;
                                        endif
                                endfor
                                if !Lcapture_input
                                        fprintf(fidlog,"\nCheck OK: Captured X,Y recirculation blocks, phase
bins and\n");
                                        fprintf(fidlog,"          dump commands are consistent between chains

with common\n");
                                        fprintf(fidlog,"          input data streams.\n");
                                endif
                        endif

                        # generate a table of bias and dump-bias difference
                        if Lbias_table
                                bias_diff_out= sprintf("stream %s dump %s bias:!OVR %s dump-bias:!OVR %s bias:OVR %s
dump-bias:OVR\n",...
                                        CS,CS,CS,CS,CS);
                                for iot=1:rows(iactive)
                                        bias_diff= applied_dump(iactive)-input_bias(iactive) - base_holdoff;
```

```
                                        bias_diff(find(!input_bias(iactive)))= -1;
                                        bias_diff_OVR= applied_dump(iactive)-input_bias_OVR(iactive) -
base_holdoff;
                                        bias_diff_OVR(find(!input_bias_OVR(iactive)))= -1;
                                        outstr= sprintf("%d %s %d %s %d %s %d %s %d %s %d\n",...
                                        iactive(iot)-1,CS,applied_dump(iactive(iot)),CS,
input_bias(iactive(iot)),...
                                        CS, bias_diff(iot),CS,  input_bias_OVR(iactive(iot)),CS,
bias_diff_OVR(iot));
                                        outstr= strrep(outstr,"-1","n-a");
                                        bias_diff_out= [bias_diff_out, outstr];
                        endfor
                        fidbias= fopen(biasfspec,"w");
                        fprintf(fidbias,"%s", bias_diff_out);
                        fclose(fidbias);
        endif
        Lmcsr= !1;
        if mcb.MCSR.AOV != test_mcb.MCSR.AOV
                        fprintf(fidlog,"\n%s from check_register_capture: MCSR.AOV differs\n\
                        from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.MCSR.OVR != test_mcb.MCSR.OVR
                        fprintf(fidlog,"\n%s from check_register_capture: MCSR.OVR differs from\n\
                        template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.MCSR.SyncER != test_mcb.MCSR.SyncER
                        fprintf(fidlog,"\n%s from check_register_capture: MCSR.SynceER differs from\n\
                        template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.MCSR.TvER != test_mcb.MCSR.TvER
                        fprintf(fidlog,"\n%s from check_register_capture: MCSR.TVER differs from\n\
                        template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if any(mcb.X_DUMP_EN != test_mcb.X_DUMP_EN)
                        fprintf(fidlog,"\n%s from check_register_capture: status X_DUMP_EN\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if any(mcb.Y_DUMP_EN != test_mcb.Y_DUMP_EN)
                        fprintf(fidlog,"\n%s from check_register_capture: status Y_DUMP_EN\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.X_DUMP_SYNC != test_mcb.X_DUMP_SYNC
                        fprintf(fidlog,"\n%s from check_register_capture: status X_DUMP_SYNC\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.Y_DUMP_SYNC != test_mcb.Y_DUMP_SYNC
                        fprintf(fidlog,"\n%s from check_register_capture: status Y_DUMP_SYNC\n\
                        differs       from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.X_SCHID_SYNC != test_mcb.X_SCHID_SYNC
                        fprintf(fidlog,"\n%s from check_register_capture: status X_SCHID_SYNC\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.Y_SCHID_SYNC != test_mcb.Y_SCHID_SYNC
                        fprintf(fidlog,"\n%s from check_register_capture: status Y_SCHID_SYNC\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.X_TIMESTAMP != test_mcb.X_TIMESTAMP
                        fprintf(fidlog,"\n%s from check_register_capture: status X_TIMESTAMP\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if mcb.Y_TIMESTAMP != test_mcb.Y_TIMESTAMP
                        fprintf(fidlog,"\n%s from check_register_capture: status Y_TIMESTAMP\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if any(any(mcb.X_DATA')' != test_mcb.X_DATA)
                        fprintf(fidlog,"\n%s from check_register_capture: status X_DATA\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
        if any(any(mcb.Y_DATA')' != test_mcb.Y_DATA)
                        fprintf(fidlog,"\n%s from check_register_capture: status Y_DATA\n\
                        differs from template.\n",flgstr);
                        Lmcsr= !0;
        endif
```

```
                if any(any(mcb.X_PHASE')' != test_mcb.X_PHASE)
                                fprintf(fidlog,"\n%s from check_register_capture: status X_PHASE\n\
                                differs from template.\n",flgstr);
                                Lmcsr= !0;
                endif
                if any(any(mcb.Y_PHASE')' != test_mcb.Y_PHASE)
                                fprintf(fidlog,"\n%s from check_register_capture: status Y_PHASE\n\
                                differs from template.\n",flgstr);
                                Lmcsr= !0;
                endif
                if any(mcb.X_DVALID != test_mcb.X_DVALID)
                                fprintf(fidlog,"\n%s from check_register_capture: status X_DVALID\n\
                                differs from template.\n",flgstr);
                                Lmcsr= !0;
                endif
                if any(mcb.Y_DVALID != test_mcb.Y_DVALID)
                                fprintf(fidlog,"\n%s from check_register_capture: status Y_DVALID\n\
                                differs from template.\n",flgstr);
                                Lmcsr= !0;
                endif
                if any(mcb.X_SE_CLK != test_mcb.X_SE_CLK)
                                fprintf(fidlog,"\n%s from check_register_capture: status X_SE_CLK\n\
                                differs from template.\n",flgstr);
                                Lmcsr= !0;
                endif
                if any(mcb.Y_SE_CLK != test_mcb.Y_SE_CLK)
                                fprintf(fidlog,"\n%s from check_register_capture: status Y_SE_CLK\n\
                                differs from template.\n",flgstr);
                                Lmcsr= !0;
                endif
                if !Lmcsr
                                fprintf(fidlog,"\nCheck OK: MCSR and status registers agree with the\n\
                                template of expected states.\n",flgstr);
                else
                                if LWARN
                                                Lwarn = !0;
                                else
                                                Ladvis = !0;
                                endif
                endif
%%              if !Lwarn & !Ladvis
%%                              printf("\nCheck OK: no inconsistencies reported in captured data\n");
%%                              printf("          or register contents; no unexpected register reports;\n");
%%                              printf("          details in log file.\n");
%%              else
%%                              if Lwarn
%%                                              printf("\nWARNING from check_register_capture: WARNINGS encountered;\n\
%%                                              details in log file.\n");
%%                              endif
%%                              if Ladvis
%%                                              printf("\nADVISORY from check_register_capture: ADVISORIES
encountered;\n\
%%                                              details in log file.\n");
%%                              endif
%%              endif
                fclose(fidlog);
# ------------------------------------------------------------------------
else
                printf("\nMSSG from check_register_capture: raw frame structure identified.\n");
                printf("This version returns.\n");
                return
endif

# endfunction
```

3.24 postsim_lib function listing

3.24.1 get_lag_frames

```
function[frame, nframes, description, bad_frames, dontcare_frames] = get_lag_frames(filename)
#
# usage: [frame, num_frames, desc, bad_frames, dontcare_frames]
#                                              = get_lag_frames(filespec)
#
# description: This function reads the output frame file into a structure array
#              of num_frames elements; organized as one frame per element.
#              Hexadecimal digits marked as 'don't care' in the file are
#              read and are translated in the output structure as either NaN
#              in the case of numbers or with special flags in the case of
#              boolean quantities. Frames with bad checksums are also read and
#              flagged as such in the output structure.
#
# frame(1:num_frames): a structure containing the 128 complex lags and
#                      related information
# num_frames: the number of 128 complex lag frames read; 0 if an error occurs
# desc: a string containing the test case description
#       as extracted from the file header
# bad_frames: indexes of frames with bad checksums (column vector)
# dontcare_frames: indexes of frames containing don't care hex digits
# filespec: complete file specifier for the lag file
#
# frame.status.B31     (boolean)
# frame.dontcare.B31   (boolean)
# frame.status.Yin     (boolean)
# frame.dontcare.Yin   (boolean)
# frame.status.Xin        (boolean)
# frame.dontcare.Xin      (boolean)
# frame.status.YSyncerr   (boolean)
# frame.dontcare.YSyncerr (boolean)
# frame.status.XSyncer    (boolean)
# frame.dontcare.XSyncer  (boolean)
# frame.status.ACC_OV     (boolean)
# frame.dontcare.ACC_OV   (boolean)
# frame.status.OVR        (boolean)
# frame.dontcare.OVR      (boolean)
# frame.NUM_CLAGS
# frame.CCC
# frame.cmmd   (string)
# frame.BBID.X
# frame.BBID.Y
# frame.SBID.X
# frame.SBID.Y
# frame.SID.Y
# frame.SID.X
# frame.Phase_bin
# frame.RECIRC_BLK.X
# frame.RECIRC_BLK.Y
# frame.TIMESTAMP_0
# frame.TIMESTAMP_1
# frame.DVCOUNT_centre
# frame.DVCOUNT_edge
# frame.DATA_BIAS
# frame.clag           (complex array)
# frame.bad_checksum   (boolean)    : true for a bad checksum on this frame
# frame.dontcare_recs  (col. vector): record (word) numbers (1:264) containing
#                                     don't care hex digits
#
#
## [Ver_date]= get_lag_frames();
## Ver_date: version number: date this version was created

## ********************* Revision history *********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
## 1.10; Oct. 26, 2004: add bad_checksum flag to frame output
##                      structure; flag "don't care" hex digits
##                      by NaN for numbers and a special flag for
##                      boolean; add the bad_frames and dontcare
##                      vectors in the output list; add
##                      dontcare_recs to the output structure;
##                      remove warnings.
## 1.20; Dec. 04, 2004: identify frame start and stop by
##                      by using the complete line text.
## 1.30; Aug. 18, 2005: reverse order of cmmd string
##
## ************************************************************

Version_date= "1.30: 05-08-18";
if nargin == 0
```

```
                frame= Version_date;
                return;
endif

FRAME_SIZE = 264;   # number of records per lag frame (excluding the start sync., stop sync. and checksum)

lagfile=fopen(filename,"r");
[allframes,count]=fscanf(lagfile,"%c",Inf);
fclose(lagfile);

bad_frames= zeros(0);  # inialize a list of frames with bad checksums to empty

if (count == 0)
                disp("No records were read from the specified file. File specifier entered is:"), disp(filename);
                disp("Returning from get_lag_frames.");
                nframes = 0;
                return;
endif

framestart=findstr(allframes, "aaaaaaaa     W0: START SYNC WORD -- OK");

if isempty(framestart)
                disp("No lag frames were identified (no start sync). Returning.");
                nframes=0;
endif

nframes=length(framestart);
framestop=findstr(allframes, "1c71c71c     W265: END SYNC WORD -- OK");

if (length(framestop) != nframes)
                printf("ERROR: Message from get_lag_frames:\n");
                printf("Check the file. The number of start and stop sync. records are not equal.\n Returning.\n");
endif

description=allframes(1:(framestart(1)-1));
end_length = count-framestop(nframes)-1;  # to catch the checksum record following the end sync. record

dontcare_frames=zeros(0); # inialize the don't care frame list to empty
for j=1:nframes
                one_frame=allframes(framestart(j):(framestop(j)+end_length));
                eol=findstr(one_frame,"\n");
                num_lines=length(eol);
                dontcare_list=zeros(0);  # inialize the don't care record list to empty

                if (num_lines < FRAME_SIZE)
                                printf("The number of lines identified in lag frame %d  is %d.\n",j,num_lines);
                                printf("This does not appear long enough to accomodate %d
lags.\nReturning.\n",FRAME_SIZE);
                                nframes=j-1;
                                return;
                endif

                for i=1:FRAME_SIZE
                                hframe(i,:)=one_frame((eol(i)+1):(eol(i)+8));
                                anydontcare= [findstr(hframe(i,:), "x"),findstr(hframe(i,:),"X")]; # find recs with
                                                                                                    don't care
                                if !isempty(anydontcare)
                                                dontcare_list= [dontcare_list,i];  # add this record number to the don't
                                                                                                    care list
                                endif
                endfor
                if any(dontcare_list) dontcare_frames= [dontcare_frames,j]; endif;  # add frame number to the don't
                                                                                                    care list

                checksum_report=one_frame((eol(FRAME_SIZE+2)+14):(eol(FRAME_SIZE+2)+41));

                bad_chksum_frame= !1;        # initialize flag as having a good checksum
                if !strcmp(checksum_report,"W266: Checksum calculated OK")
                                bad_chksum_frame= !0;            # flag this frame as having a bad checksum
                                bad_frames=[bad_frames,j];  # add this index to the bad checksums vector
                endif

                # flip l-r bit order so that bit significance increases with array index, hence enhance readability

                for kn=1:3  # consider the first three records of the lag frame (following the frame start record)
                                # hex string padded with an extra digit to assure that dec2bin provides 8 bits of
                                binary characters
                                # remove the extra digit later
                                wdec= hex2dec([hframe(kn,:),"8"]')';
                                wnan=isnan(wdec);                       # find the don't care digits; these are NaN
                                if any(wnan) wdec(find(wnan))=0; endif; # set the don't care decimal elements to zero
                                                                            for now
                                Wpad= reshape(dec2bin(wdec)',1,36);    # reshape vector Wpad to make a single string of
                                                                            binary digits
                                for im=find(wnan);                      # "true" elements of wnan flag don't care hex
                                                                            digits
                                                Wpad((4*im-3):((4*im)))="x";   # replace binary don't care digits with
```

```
"x"
                            endfor;
                            W(kn,:)=Wpad(1:32);                    # remove the characters corresponding to the pad
                endfor

                W=fliplr(W);

                frame(j).status.B31=  (W(1,32)=="1");
                frame(j).dontcare.B31=(W(1,32)=="x");
                frame(j).status.Yin=  (W(1,31)=="1");
                frame(j).dontcare.Yin=(W(1,31)=="x");
                frame(j).status.Xin=  (W(1,30)=="1");
                frame(j).dontcare.Xin=(W(1,30)=="x");
                frame(j).status.YSyncerr=  (W(1,29)=="1");
                frame(j).dontcare.YSyncerr=(W(1,29)=="x");
                frame(j).status.XSyncerr=  (W(1,28)=="1");
                frame(j).dontcare.XSyncerr=(W(1,28)=="x");
                frame(j).status.ACC_OV=  (W(1,27)=="1");
                frame(j).dontcare.ACC_OV=(W(1,27)=="x");
                frame(j).status.OVR=  (W(1,26)=="1");
                frame(j).dontcare.OVR=(W(1,26)=="x");
                frame(j).NUM_CLAGS=bin2dec(fliplr(W(1,10:19)));
                frame(j).CCC=bin2dec(fliplr(W(1,4:9)));
                frame(j).cmmd=fliplr(W(1,1:3));
                frame(j).BBID.X=bin2dec(fliplr(W(2,14:16)));
                frame(j).BBID.Y=bin2dec(fliplr(W(2,30:32)));
                frame(j).SBID.X=bin2dec(fliplr(W(2,9:13)));
                frame(j).SBID.Y=bin2dec(fliplr(W(2,25:29)));
                frame(j).SID.Y=bin2dec(fliplr(W(2,17:24)));
                frame(j).SID.X=bin2dec(fliplr(W(2,1:8)));
                frame(j).Phase_bin=bin2dec(fliplr(W(3,17:32)));
                frame(j).RECIRC_BLK.X=bin2dec(fliplr(W(3,1:8)));
                frame(j).RECIRC_BLK.Y=bin2dec(fliplr(W(3,9:16)));
                frame(j).TIMESTAMP_0=hex2dec(hframe(4,:));
                frame(j).TIMESTAMP_1=hex2dec(hframe(5,:));
                frame(j).DVCOUNT_centre=hex2dec(hframe(6,:));
                frame(j).DVCOUNT_edge=hex2dec(hframe(7,:));
                frame(j).DATA_BIAS=hex2dec(hframe(8,:));
                frame(j).clag=hex2dec(hframe(9:2:(FRAME_SIZE-1),:))+hex2dec(hframe(10:2:FRAME_SIZE,:))*1j;
                frame(j).bad_checksum= bad_chksum_frame;
                frame(j).dontcare_recs= dontcare_list;
endfor
endfunction
```

3.24.2 make_dump_index

```
function[dump, idump, ibadstamp] = make_dump_index(frame)
#
# usage: [dump, ndump, ibadstamp] = make_dump_index(frame)
#
# description: Given a structure array containing correlator model
#              output frames, create a structure containing arrays of
#              indices pertaining to each dump, one array per dump.
#
# dump.TIMESTAMP_0: timestamp word 0
# dump.TIMESTAMP_1: timestamp word 1
# dump.index: an array containg indices of the elements of frame associated
#             with the above time stamps.
# ndump: number of dumps processed.
# ibadstamp: an array of frame indices with corrupt
#            TIMESTAMP (words 0 or 1).
#
# NOTE: This function detects a time stamp rollover and flags pre-rollover
#       times with negative values. Convert to the original time stamp by
#       adding 2^32 to all negative time stamps.
#       Limitation: The total frame sequence cannot span more than
#       one increment of time stamp: word 1
#
## [Ver_date]= make_dump_index();
## Ver_date: version number: date this version was created

## ********************* Revision history *********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
## 1.10; Oct. 27, 2004: detect corrupted TIMESTAMP and add the
##                      array ibadstamp to the output list.
##
## ************************************************************

Version_date= "1.10: 04-10-27";
if nargin == 0
                dump= Version_date;
```

```
                return;
endif

nframes = columns(frame);
timestamp0=zeros(nframes,1);
timestamp1=zeros(nframes,1);

# copy the time stamps from the input structure to vectors
for j=1:nframes; timestamp0(j)=frame(j).TIMESTAMP_0; timestamp1(j)=frame(j).TIMESTAMP_1; end

badstamp= isnan(timestamp0) | isnan(timestamp1);
ibadstamp= find(badstamp);
igoodstamp= find(!badstamp);
if !isempty(ibadstamp)
                printf("WARNING from make_dump_index: Corrupt TIMESTAMP detected.\n");
endif


# deal with the case where a timestamp rollover appears within this output frame.
# Limitation: The total frame sequence cannot span more than one increment word 1
if (max(timestamp1(igoodstamp)) == (2^32-1) & min(timestamp1(igoodstamp)) == (0));
                tsatmax = find(timestamp1(igoodstamp) == (2^32-1));
                timestamp1(tsatmax)=-1; # allow timestamp rollover
                timestamp0(tsatmax)=timestamp0(tsatmax)-2^32;
                printf("WARNING from make_dump_index: Times stamp rollover detected.\n Assume rollover in
ordering.\n")
endif

# create boolean arrays that flag elements of frame associated with the
# time stamps under consideration (being processed).
rdprev= badstamp;          # previously processed
rd= badstamp;              # currently being processed

idump=0;  # dump counter

do
                idnew=find(!rdprev);
                mints0=min(timestamp0(idnew));
                mints1=min(timestamp1(idnew));
                rd=((timestamp0 == mints0) & (timestamp1 == mints1));
                ird=(find(rd));
                if isempty(ird)
                                printf("Inconsistent timestamp data.\n Returning.\n"); return;
                endif
                ++idump;
                dump(idump).TIMESTAMP_0 = mints0;
                dump(idump).TIMESTAMP_1 = mints1;
                dump(idump).index = ird;
                rdprev = rd | rdprev;
until all(rdprev)   # continue until all timestamps are flagged as previously processed

endfunction
```

3.24.3 add_CCchain_dump

```
function[dump, with_chains]=add_CCchain_dump(dump, frame)
##
## usage: [dump, with_chains] = add_CCchain_dump(dump, frame)
##
## description: Check if structure "CCchain" exists within
##              dump; if not, create "CCchain" from frame
##              and append this structure to dump. Return
##              the extended dump structure.
##
##       notes: (1) No attempt is made to either detect or
##                  recover chains from dumps made with
##                  frames containing corrupt timestamps.
##              (2) Chains from dumps containing frames
##                  with corrupt status bits or cell
##                  numbers are excluded.
##              (3) Single cells are processed before
##                  chains of concatenated cells.
##              (4) The chain data bias is returned as zero
##                  if the OVR bits differ between cells of
##                  that chain. However, the DATA_BIAS
##                  of each cell will be independently
##                  applied to the lag data of that cell.
##              (5) Scaling and bias subtraction are not
##                  applied to a chain if DATA_BIAS is
##                  not identical for each cell of the
##                  chain AND (4) does not hold; data_bias
##                  is returned empty.
##              (6) Malformed chains will be ignored and a
```

```
##                 warning issued.
##              (7) Broken chains (containing non adjacent
##                  cells) are returned with zero length and
##                  contain no data; a warning is issued.
## CCchain(ic).length: length of chain ic in cells.
## CCchain(ic).CCC: array of cell numbers in chain ic.
## CCchain(ic).OVR: boolean array of chain length, true if
##                  overrun bit is set.
## CCchain(ic).ACC_OV: boolean array of chain length, true if
##                     accumulator overflow bit is set.
## CCchain(ic).XSyncerr: boolean array of chain length, true if
##                       a X-input synch. error was detected.
## CCchain(ic).YSyncerr: boolean array of chain length, true if
##                       a Y-input synch. error was detected.
## CCchain(ic).DATA_BIAS: DATA_BIAS for each cell of chain ic.
## CCchain(ic).DVCOUNT_centre: for each cell of chain ic.
## CCchain(ic).DVCOUNT_edge: for each cell of chain ic.
## CCchain(ic).Phase_bin: for each cell of chain ic.
## CCchain(ic).RECIRC_BLK: for each cell of chain ic.
## CCchain(ic).RECIRC_BLK_Y: for each cell of chain ic.
## CCchain(ic).BBID_X: for each cell of chain ic.
## CCchain(ic).BBID_Y: for each cell of chain ic.
## CCchain(ic).SBID_X: for each cell of chain ic.
## CCchain(ic).SBID_Y: for each cell of chain ic.
## CCchain(ic).SID_X: for each cell of chain ic.
## CCchain(ic).SID_Y: for each cell of chain ic.
## CCchain(ic).cmmd: string holding the dump command (cmmd)
##                   for each cell of chain ic.
## CCchain(ic).clag: complex lags concatenated for chain ic.
##                   These lag data have the data bias removed
##                   and scaled by dvcount.
## CCchain(ic).data_bias: The bias selected for application to clag.
##                        (See notes (4) and (5).)
## CCchain(ic).dvcount: The data valid count applied to clag.
##
## with_chains: a row vector list of all dumps containing
##               chains; returned empty if no chains were added.
##               Chains of zero length are counted.
##
## [Ver_date]= add_CCchain_dump();
## Ver_date: version number: date this version was created

## ********************** Revision history **********************
## (version; date created: description of changes)
## 1.00: first working version
##
## 1.10; July 20, 2004: added BBID_X, BBID_Y fields to CCchain
##                      structure; return Version_date.
##
## 1.11; Nov.  1, 2004: added check to exclude frames
##                      corrupt status bits and cell numbers; add
##                      the vector with_dump to the output list;
##                      seek valid NUM_CLAGS; bug fix corrects
##                      operation with mixed single cells and
##                      chains; bug fix corrects formation
##                      of the .cmmd field array.
##
## 1.20; Nov. 21, 2004: add check for OVR variation over cells
##                      of a chain and apply DATA_BIAS
##                      independently to each cell. Return
##                      a zero data_bias value. Version 1.10 or
##                      greater of scale_lag_data is required.
##
## 1.30; Jan. 14, 2005: added code to process autocorrelation
##                      data.
##
## 1.40; Mar. 02, 2005: fix error on dumps without chains;
##                      identification and recovery of valid
##                      chains on dumps with incomplete chains.
##
## 1.41; Jun. 26, 2005: typo fix: ichain_temp to ichains_temp
##
## ************************************************************

Version_date= "1.41: 05-06-26";
if nargin == 0
              dump= Version_date;
              return;
endif

if isfield(dump,"CCchain")
              printf("The field CCchain already exists in input to function: add_CCchain_dump.\n Returning.\n");
              return;
endif

NUMCELLS= 16;            # number of cells on the chip
ndump = columns(dump);   # number of dumps
```

```
idfull= !ones(ndump,1);  # boolean array to flag dumps where chains were added
mixed_OVR_chain= !1;     # flag is true if chains with mixed OVR status between cells was detected.
Lmatch_start_end= !1;    # flag is true if number of the chain starts and ends are unequal.
Lsequence= !1;           # True if chain ends do not follow their corresponding chain starts.
Lnotadjacent= !1;        # True for cell adjacency failure in a chain.

for idump=1:ndump
               dumpn = rows(dump(idump).index);
               # dumpn: number of frames associated with this dump (time stamp).
               # idump: the index of the current dump
               CCC_dump=zeros(dumpn,1);
               CCC_inp= !ones(dumpn,2);
               status_invalid= !ones(dumpn,1);
               for k=1:dumpn
                              CCC_dump(k,1) = frame(dump(idump).index(k)).CCC;
                              CCC_inp(k,1) = frame(dump(idump).index(k)).status.Xin;   # X-input concatenation flag
                              CCC_inp(k,2) = frame(dump(idump).index(k)).status.Yin;   # Y-input concatenation flag
                              status_invalid(k) = (frame(dump(idump).index(k)).dontcare.Xin) | ...
                                 (frame(dump(idump).index(k)).dontcare.Yin) |
(frame(dump(idump).index(k)).dontcare.XSyncerr) | ...
                                 (frame(dump(idump).index(k)).dontcare.YSyncerr) | ...
                                 (frame(dump(idump).index(k)).dontcare.ACC_OV) |
(frame(dump(idump).index(k)).dontcare.OVR);
               endfor
               if (any(isnan(CCC_dump)) | any(status_invalid))
                              continue;              # skip this dump for corrupt cell numbers and corrupt status
                                                            bits
               endif

               # scc contains the cell numbers sorted in ascending order (for this dump);
               # icc are the corresponding indices of the unsorted cell vector.

               [scc,icc]=sort(CCC_dump);

               isingle_cell = icc(find(!CCC_inp(icc,1) & !CCC_inp(icc,2)));  # find any isolated cells (not
                                                                                    concatenated)

               # process any isolated (non-concatenated) cells, in ascending cell number order
               ncells= 0;  # initialize the number of single cells. Zero used later for chains if there are no single
                              cells.
               if !isempty(isingle_cell)
                              ncells = length(isingle_cell);
                              for ic=1:ncells
                                             CCchain(ic).length = 1;
                                             CCchain(ic).CCC = CCC_dump(isingle_cell(ic));
                                             CCchain(ic).OVR = 1 &
frame(dump(idump).index(isingle_cell(ic))).status.OVR;
                                             CCchain(ic).ACC_OV = 1 &
frame(dump(idump).index(isingle_cell(ic))).status.ACC_OV;
                                             CCchain(ic).XSyncerr = 1 &
frame(dump(idump).index(isingle_cell(ic))).status.XSyncerr;
                                             CCchain(ic).YSyncerr = 1 &
frame(dump(idump).index(isingle_cell(ic))).status.YSyncerr;
                                             CCchain(ic).DATA_BIAS =
frame(dump(idump).index(isingle_cell(ic))).DATA_BIAS;
                                             CCchain(ic).DVCOUNT_centre =
frame(dump(idump).index(isingle_cell(ic))).DVCOUNT_centre;
                                             CCchain(ic).DVCOUNT_edge =
frame(dump(idump).index(isingle_cell(ic))).DVCOUNT_edge;
                                             CCchain(ic).Phase_bin =
frame(dump(idump).index(isingle_cell(ic))).Phase_bin;
                                             CCchain(ic).RECIRC_BLK_X =
frame(dump(idump).index(isingle_cell(ic))).RECIRC_BLK.X;
                                             CCchain(ic).RECIRC_BLK_Y =
frame(dump(idump).index(isingle_cell(ic))).RECIRC_BLK.Y;
                                             CCchain(ic).BBID_X = frame(dump(idump).index(isingle_cell(ic))).BBID.X;
                                             CCchain(ic).BBID_Y = frame(dump(idump).index(isingle_cell(ic))).BBID.Y;
                                             CCchain(ic).SBID_X = frame(dump(idump).index(isingle_cell(ic))).SBID.X;
                                             CCchain(ic).SBID_Y = frame(dump(idump).index(isingle_cell(ic))).SBID.Y;
                                             CCchain(ic).SID_X = frame(dump(idump).index(isingle_cell(ic))).SID.X;
                                             CCchain(ic).SID_Y = frame(dump(idump).index(isingle_cell(ic))).SID.Y;
                                             CCchain(ic).cmmd = frame(dump(idump).index(isingle_cell(ic))).cmmd;
                                             CCchain(ic).clag = frame(dump(idump).index(isingle_cell(ic))).clag;
                                             [sclags,dvcount,data_bias] = scale_lag_data(CCchain(ic));
                                             if (!isempty(sclags) && !isempty(dvcount) && !isempty(data_bias))
                                                            CCchain(ic).dvcount= dvcount;
                                                            CCchain(ic).data_bias= data_bias;
                                                            CCchain(ic).clag = sclags;
                                             endif
                              endfor
               endif

               jk=0;
               nframes= columns(frame);
               do                                          # search for a valid NUM_CLAGS
                              nlags = frame(++jk).NUM_CLAGS;
                              if (jk == nframes)
```

```
                                        printf("ERROR from add_CCchain_dump: NUM_CLAGS in all frames is
corrupt.\n");
                                        printf("Returning.\n");
                                        return;
                        endif
                until (!isnan(nlags))                   # NaN interpreted as a corrupt NUM_CLAGS field

                if (dumpn == NUMCELLS) & all(CCC_inp(:,1)) & all(CCC_inp(:,2)) # also accounts for dumps without
                                                                                chains
                        printf("\nMSSG from add_CCchain_dump: autocorrelation mode detected.\n");
                        CCC_inp(icc(1),1)= !1;          # make cell 0 the start
                        CCC_inp(icc(NUMCELLS),2)= !1; # make cell NUMCELLS the end
                endif

                ichain_start= find(!CCC_inp(icc,1) & CCC_inp(icc,2));  # find the start of (the sorted) cell chains
                ichain_end= find(CCC_inp(icc,1) & !CCC_inp(icc,2));    # find the ends of (the sorted) cell chains

                if (!isempty(ichain_start) & !isempty(ichain_end))
                        nchains= length(ichain_start);
                        nchains_end= length(ichain_end);
                        if nchains_end != nchains
                                        # Number of concatenated cell chain starts does not equal the number of
                                                ends.
                                        Lmatch_start_end= !0;
                                        # attempt to recover valid chains
                                        if nchains < nchains_end          # find the shortest of start or end
                                                                          vectors
                                                ichains_temp= ichain_start;
                                                for iss= 1:nchains
                                                        try_end=
ichain_end(min(find((ichain_start(iss) < ichain_end))));
                                                        if isempty(try_end)
                                                                ichains_temp(iss)= -1;
                                                        else
                                                                ichains_temp(iss)= try_end;
                                                        endif
                                                endfor
                                                ichain_end= ichains_temp(find(ichains_temp >= 0));
                                        else
                                                ichains_temp= ichain_end;
                                                for iss= 1:nchains_end
                                                        try_start=
ichain_start(max(find((ichain_end(iss) > ichain_start))));
                                                        if isempty(try_start)
                                                                ichains_temp(iss)= -1;
                                                        else
                                                                ichains_temp(iss)=
try_start;
                                                        endif
                                                endfor
                                                ichain_start= ichains_temp(find(ichains_temp >= 0));
                                                nchains= nchains_end;
                                        endif
                        endif

                        # Assure that the sorted end indices follow the start indices
                        # This will test for the case where all ends follow starts.
                        if (!all(ichain_start < ichain_end)) | isempty(ichain_start) | isempty(ichain_end)
                                        # The chain ends do not follow their corresponding chain starts.
                                        Lsequence= !0;
                                        continue;
                        endif

                        for ic=(1):nchains
                                        ich= ic+ncells;  # chains are formed after any single cells, hence
(+ncells)
                                        one_chain_indices=icc(ichain_start(ic):ichain_end(ic));
                                        chain_length=length(one_chain_indices);
                                        CCchain(ich).length = chain_length;
                                        CCchain(ich).CCC = zeros(chain_length,1);
                                        CCchain(ich).OVR = 0 & zeros(chain_length,1);
                                        CCchain(ich).ACC_OV = 0 & zeros(chain_length,1);
                                        CCchain(ich).XSyncerr = 0 & zeros(chain_length,1);
                                        CCchain(ich).YSyncerr = 0 & zeros(chain_length,1);
                                        CCchain(ich).DATA_BIAS = zeros(chain_length,1);
                                        CCchain(ich).DVCOUNT_centre = zeros(chain_length,1);
                                        CCchain(ich).DVCOUNT_edge = zeros(chain_length,1);
                                        CCchain(ich).Phase_bin = zeros(chain_length,1);
                                        CCchain(ich).RECIRC_BLK_X = zeros(chain_length,1);
                                        CCchain(ich).RECIRC_BLK_Y = zeros(chain_length,1);
                                        CCchain(ich).SBID_X = zeros(chain_length,1);
                                        CCchain(ich).SBID_Y = zeros(chain_length,1);
                                        CCchain(ich).BBID_X = zeros(chain_length,1);
                                        CCchain(ich).BBID_Y = zeros(chain_length,1);
                                        CCchain(ich).SID_X = zeros(chain_length,1);
                                        CCchain(ich).SID_Y = zeros(chain_length,1);
```

```
                                        # define temporary arrays
                                        clags = zeros(nlags,chain_length);  # complex lags for each cell in the
                                                                                chain
                                        cmmd = "";

                                        for icd=1:chain_length
                                                CCchain(ich).CCC(icd) =
frame(dump(idump).index(one_chain_indices(icd))).CCC;
                                                CCchain(ich).OVR(icd) = (1 &
frame(dump(idump).index(one_chain_indices(icd)))...
                                                                                        .status.OVR);
                                                CCchain(ich).ACC_OV(icd) = (1 & frame(dump(idump)...

.index(one_chain_indices(icd))).status.ACC_OV);
                                                CCchain(ich).XSyncerr(icd) = (1 & frame(dump(idump)...

.index(one_chain_indices(icd))).status.XSyncerr);
                                                CCchain(ich).YSyncerr(icd) = (1 & frame(dump(idump)...

.index(one_chain_indices(icd))).status.YSyncerr);
                                                CCchain(ich).DATA_BIAS(icd) = (frame(dump(idump)...

.index(one_chain_indices(icd))).DATA_BIAS);
                                                CCchain(ich).DVCOUNT_centre(icd) = (frame(dump(idump)...

.index(one_chain_indices(icd))).DVCOUNT_centre);
                                                CCchain(ich).DVCOUNT_edge(icd) = (frame(dump(idump)...

.index(one_chain_indices(icd))).DVCOUNT_edge);
                                                CCchain(ich).Phase_bin(icd) = (frame(dump(idump)...

.index(one_chain_indices(icd))).Phase_bin);
                                                CCchain(ich).RECIRC_BLK_X(icd) = (frame(dump(idump)...

.index(one_chain_indices(icd))).RECIRC_BLK.X);
                                                CCchain(ich).RECIRC_BLK_Y(icd) = (frame(dump(idump)...

.index(one_chain_indices(icd))).RECIRC_BLK.Y);
                                                CCchain(ich).BBID_X(icd) =
(frame(dump(idump).index(one_chain_indices(icd)))...
                                                                                        .BBID.X);
                                                CCchain(ich).BBID_Y(icd) =
(frame(dump(idump).index(one_chain_indices(icd)))...
                                                                                        .BBID.Y);
                                                CCchain(ich).SBID_X(icd) =
(frame(dump(idump).index(one_chain_indices(icd)))...
                                                                                        .SBID.X);
                                                CCchain(ich).SBID_Y(icd) =
(frame(dump(idump).index(one_chain_indices(icd)))...
                                                                                        .SBID.Y);

cmmd=[cmmd;frame(dump(idump).index(one_chain_indices(icd))).cmmd];
                                                CCchain(ich).SID_X(icd) =
(frame(dump(idump).index(one_chain_indices(icd)))...
                                                                                        .SID.X);
                                                CCchain(ich).SID_Y(icd) =
(frame(dump(idump).index(one_chain_indices(icd)))...
                                                                                        .SID.Y);
                                                clags(:,icd) =
frame(dump(idump).index(one_chain_indices(icd))).clag;
                                        endfor
                                        # apply data bias to each cell independently when the the OVR bit is set
                                        # in only some of the cells of the chain. The version of scale_lag_data
                                        # used here must return a zero bias for similar OVR flag conditions.
                                        if any(!CCchain(ich).OVR) & any(CCchain(ich).OVR)
                                                mixed_OVR_chain= !0;
                                                for icd=1:chain_length
                                                        clags(:,icd) = clags(:,icd) -
(CCchain(ich).DATA_BIAS(icd)*(1+1i));
                                                endfor
                                        endif
                                # Perform cell adjacency check before assigning chain.
                                if !any(diff(CCchain(ich).CCC) > 1)
                                        # reverse order of cells in clags (increasing lag with decreasing cell
number) and reshape
                                        # to form a complete lag chain with lags in ascending order
                                        CCchain(ich).clag = reshape(fliplr(clags),(nlags*chain_length),1); #
concatenate lags

                                        # subtract the bias and scale by the data valid count of the centre lag
                                        # Note: data_bias is returned empty if the bias is not identical for all
cells in the chain
                                        [sclags,dvcount,data_bias] = scale_lag_data(CCchain(ich));
                                        if (!isempty(sclags) && !isempty(dvcount) && !isempty(data_bias))
                                                CCchain(ich).dvcount= dvcount;
                                                CCchain(ich).data_bias= data_bias;
                                                CCchain(ich).clag = sclags;
```

```
                                        endif
                                        CCchain(ich).cmmd= cmmd;
                                        Lnotadjacent= !1;
                                else
                                        # adjacency condition failure: empty data from chain and set length to
                                                zero
                                        chain_length= 0;
                                        CCchain(ich).length = chain_length;
                                        CCchain(ich).CCC = zeros(chain_length,1);
                                        CCchain(ich).OVR = 0 & zeros(chain_length,1);
                                        CCchain(ich).ACC_OV = 0 & zeros(chain_length,1);
                                        CCchain(ich).XSyncerr = 0 & zeros(chain_length,1);
                                        CCchain(ich).YSyncerr = 0 & zeros(chain_length,1);
                                        CCchain(ich).DATA_BIAS = zeros(chain_length,1);
                                        CCchain(ich).DVCOUNT_centre = zeros(chain_length,1);
                                        CCchain(ich).DVCOUNT_edge = zeros(chain_length,1);
                                        CCchain(ich).Phase_bin = zeros(chain_length,1);
                                        CCchain(ich).RECIRC_BLK_X = zeros(chain_length,1);
                                        CCchain(ich).RECIRC_BLK_Y = zeros(chain_length,1);
                                        CCchain(ich).SBID_X = zeros(chain_length,1);
                                        CCchain(ich).SBID_Y = zeros(chain_length,1);
                                        CCchain(ich).BBID_X = zeros(chain_length,1);
                                        CCchain(ich).BBID_Y = zeros(chain_length,1);
                                        CCchain(ich).SID_X = zeros(chain_length,1);
                                        CCchain(ich).SID_Y = zeros(chain_length,1);
                                        Lnotadjacent= !0;
                                endif


                                endfor
                        endif

                        if exist("CCchain","var")  # some dumps may not have chains
                                dump(idump).CCchain = CCchain;
                                idfull(idump)= !0;
                                clear("CCchain");
                        endif
        endfor

        if !all(idfull) printf("\nWARNING from add_CCchain_dump: some dumps have no valid chains.\n"); endif;
        with_chains= find(idfull);

        if Lmatch_start_end
                        printf("\nWARNING from add_CCchain_dump:\n");
                        printf("        Lost chains on some dumps; number of concatenated cell\n");
                        printf("        chain starts does not equal the number of ends. \n");
        endif

        if Lsequence
                        printf("\nWARNING from add_CCchain_dump:\n");
                        printf("        Lost chains on some dumps; chain ends do not follow their\n");
                        printf("        corresponding chain starts.\n");
        endif

        if Lnotadjacent
                        printf("\nWARNING from add_CCchain_dump:\n");
                        printf("        Cell adjacency check failed on some chains.\n");
                        printf("        Chain length set to zero! \n");
        endif

        if mixed_OVR_chain
                        printf("\nWARNING from add_CCchain_dump: OVR status bits\n  differ between cells of some chains.\n");
                        printf("  data_bias= 0 indicates that data bias has been applied idependently\n");
                        printf("  to each cell of the chain.\n\n");
        endif

        endfunction
```

3.24.4 get_mcb_data

```
function[mcb, description] = get_mcb_data(filename)
#
# usage: [mcb, desc] = get_mcb_data(filespec)
#
# description: read the mcb register file of filename
#              and return the contents in the structure mcb.
#
# desc: a string holding the test case description from
#       the above file.
# mcb: a structure of boolean arrays in which the columns
#      hold flags representing bit settings and rows
#      correspond to input lines or cells.
#      The arrays are ordered such that bits represented
```

```
#      decrease in significance (from MSB to LSB) as
#      the index of the column increase. Hence,
#      column index 1 refers to the MSB.
#      The row index increases with line or cell number.
#      Don't care flags are set for either x or X in the
#      mcb_reg file.
#
#      mcb.MCSR.CEN      1x1 (boolean array)
#      mcb.MCSR.TVEN     1x1 (boolean array)
#      mcb.MCSR.PhEN     1x1 (boolean array)
#      mcb.MCSR.OVR      1x1 (boolean array)
#      mcb.MCSR.AOV      1x1 (boolean array)
#      mcb.MCSR.TvER     1x1 (boolean array)
#      mcb.MCSR.SyncER   1x1 (boolean array)
#      mcb.MCSR.XYDS     1x1 (boolean array)
#      mcb.CCCSCR        16x4 (boolean array)
#      mcb.CCQR          4x4 (boolean array)
#      mcb.X_DATA        8x4 (boolean array)
#      mcb.X_PHASE       8x4 (boolean array)
#      mcb.X_DVALID      8x1 (boolean array)
#      mcb.Y_DATA        8x4 (boolean array)
#      mcb.Y_PHASE       8x4 (boolean array)
#      mcb.Y_DVALID=     8x1 (boolean array)
#      mcb.X_SE_CLK      8x1 (boolean array)
#      mcb.X_DUMP_EN     8x1 (boolean array)
#      mcb.X_TIMESTAMP   1x1 (boolean array)
#      mcb.X_SCHID_SYNC  1x1 (boolean array)
#      mcb.X_DUMP_SYNC   1x1 (boolean array)
#      mcb.Y_SE_CLK      1x1 (boolean array)
#      mcb.Y_DUMP_EN     1x1 (boolean array)
#      mcb.Y_TIMESTAMP   1x1 (boolean array)
#      mcb.Y_SCHID_SYNC  1x1 (boolean array)
#      mcb.Y_DUMP_SYNC   1x1 (boolean array)
#      mcb.DESSR        16x1 (boolean array)
#      mcb.Q0.X0= CCQR   1x1 (boolean array)
#      mcb.Q0.Y0= CCQR   1x1 (boolean array)
#      mcb.Q1.X0= CCQR   1x1 (boolean array)
#      mcb.Q1.Y0= CCQR   1x1 (boolean array)
#      mcb.Q2.X0= CCQR   1x1 (boolean array)
#      mcb.Q2.Y0= CCQR   1x1 (boolean array)
#      mcb.Q3.X0= CCQR   1x1 (boolean array)
#      mcb.Q3.Y0= CCQR   1x1 (boolean array)
#      mcb.Q0.X1= CCQR   1x1 (boolean array)
#      mcb.Q0.Y1= CCQR   1x1 (boolean array)
#      mcb.Q1.X1= CCQR   1x1 (boolean array)
#      mcb.Q1.Y1= CCQR   1x1 (boolean array)
#      mcb.Q2.X1= CCQR   1x1 (boolean array)
#      mcb.Q2.Y1= CCQR   1x1 (boolean array)
#      mcb.Q3.X1= CCQR   1x1 (boolean array)
#      mcb.Q3.Y1= CCQR   1x1 (boolean array)
#      mcb.XS0= CCCSCR  16x1 (boolean array)
#      mcb.XS1= CCCSCR  16x1 (boolean array)
#      mcb.YS0= CCCSCR  16x1 (boolean array)
#      mcb.YS1= CCCSCR  16x1 (boolean array)
#
#      mcb.dontcare.MCSR.CEN      1x1 (boolean array)
#      mcb.dontcare.MCSR.TVEN     1x1 (boolean array)
#      mcb.dontcare.MCSR.PhEN     1x1 (boolean array)
#      mcb.dontcare.MCSR.OVR      1x1 (boolean array)
#      mcb.dontcare.MCSR.AOV      1x1 (boolean array)
#      mcb.dontcare.MCSR.TvER     1x1 (boolean array)
#      mcb.dontcare.MCSR.SyncER   1x1 (boolean array)
#      mcb.dontcare.MCSR.XYDS     1x1 (boolean array)
#      mcb.dontcare.CCCSCR        16x4 (boolean array)
#      mcb.dontcare.CCQR          4x4 (boolean array)
#      mcb.dontcare.X_DATA        8x4 (boolean array)
#      mcb.dontcare.X_PHASE       8x4 (boolean array)
#      mcb.dontcare.X_DVALID      8x1 (boolean array)
#      mcb.dontcare.Y_DATA        8x4 (boolean array)
#      mcb.dontcare.Y_PHASE       8x4 (boolean array)
#      mcb.dontcare.Y_DVALID=     8x1 (boolean array)
#      mcb.dontcare.X_SE_CLK      8x1 (boolean array)
#      mcb.dontcare.X_DUMP_EN     8x1 (boolean array)
#      mcb.dontcare.X_TIMESTAMP   1x1 (boolean array)
#      mcb.dontcare.X_SCHID_SYNC  1x1 (boolean array)
#      mcb.dontcare.X_DUMP_SYNC   1x1 (boolean array)
#      mcb.dontcare.Y_SE_CLK      1x1 (boolean array)
#      mcb.dontcare.Y_DUMP_EN     1x1 (boolean array)
#      mcb.dontcare.Y_TIMESTAMP   1x1 (boolean array)
#      mcb.dontcare.Y_SCHID_SYNC  1x1 (boolean array)
#      mcb.dontcare.Y_DUMP_SYNC   1x1 (boolean array)
#      mcb.dontcare.DESSR        16x1 (boolean array)
#      mcb.dontcare.Q0.X0= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q0.Y0= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q1.X0= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q1.Y0= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q2.X0= CCQR   1x1 (boolean array)
```

```
#      mcb.dontcare.Q2.Y0= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q3.X0= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q3.Y0= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q0.X1= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q0.Y1= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q1.X1= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q1.Y1= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q2.X1= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q2.Y1= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q3.X1= CCQR   1x1 (boolean array)
#      mcb.dontcare.Q3.Y1= CCQR   1x1 (boolean array)
#      mcb.dontcare.XS0= CCCSCR  16x1 (boolean array)
#      mcb.dontcare.XS1= CCCSCR  16x1 (boolean array)
#      mcb.dontcare.YS0= CCCSCR  16x1 (boolean array)
#      mcb.dontcare.YS1= CCCSCR  16x1 (boolean array)
#
## [Ver_date]= get_mcb_data();
## Ver_date: version number: date this version was created

## ********************** Revision history **********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
## 1.10; Nov.  1, 2004: add a field for don't care flags.
##
## ************************************************************

Version_date= "1.10: 04-11-01";
if nargin == 0
               mcb= Version_date;
               return;
endif

mcbfile=fopen(filename,"r");
[file_contents,count]=fscanf(mcbfile,"%c",Inf);
fclose(mcbfile);

if (count == 0)
               printf("Mssg. from get_mcb_data: No data was read from %s.\n Returning empty.\n", filename);
               mcb = zeros(0);
               description = zeros(0);
               return;
endif

first_MCSR_data=findstr(file_contents, "MCSR:");
if isempty(first_MCSR_data)
               printf("Mssg. from get_mcb_data: Could not identify the\n");
               printf("the first data line in %s.\n. Returning empty.\n", filename);
               mcb = zeros(0);
               description = zeros(0);
               return;
endif
MCSR_data_start= first_MCSR_data(1) - 10;

description= file_contents(1:(MCSR_data_start-1));

MCSR= file_contents(MCSR_data_start:MCSR_data_start+7) == "1";
MCSR_dontcare= (file_contents(MCSR_data_start:MCSR_data_start+7) == "x") | ...
               (file_contents(MCSR_data_start:MCSR_data_start+7) == "X");

# locate the CCCSCR data block
first_CCCSCR_data=findstr(file_contents, "CCCSCR_0_1:");
if isempty(first_CCCSCR_data)
               printf("Mssg. from get_mcb_data: Data format in file %s is not as expected.\n", filename);
               printf("Returning empty.\n");
               mcb = zeros(0);
               return
endif
CCCSCR_data_start= first_CCCSCR_data(1)-10;
eol= findstr(file_contents(CCCSCR_data_start-1:count),"\n"); # find the EOLs in the remaining data
CCCSCR= 0 & zeros(16,4);
CCCSCR_dontcare= 0 & zeros(16,4);
for ii = 1:8
               iccc=CCCSCR_data_start-1+eol(ii);  # index of CCCSCR data within file_contents
               CCCSCR(2*ii-1,:)= file_contents(iccc+4:iccc+7) == "1";
               CCCSCR(2*ii,:)= file_contents(iccc:iccc+3) == "1";
               CCCSCR_dontcare(2*ii-1,:)= (file_contents(iccc+4:iccc+7) == "x") | (file_contents(iccc+4:iccc+7) ==
"X");
               CCCSCR_dontcare(2*ii,:)= (file_contents(iccc:iccc+3) == "x") | (file_contents(iccc:iccc+3) == "X");
endfor

# locate the CCQR data block
first_CCQR_data=findstr(file_contents, "CCQR_0_1:");
if isempty(first_CCQR_data)
               printf("Mssg. from get_mcb_data: Data format in file %s is not as expected.\n", filename);
               printf("Returning empty.\n");
               mcb = zeros(0);
```

```
                return
endif

CCQR_data_start= first_CCQR_data(1)-10;
eol= findstr(file_contents(CCQR_data_start-1:count),"\n"); # find the EOLs in the remaining data
CCQR= 0 & zeros(2,4);
CCQR_dontcare= 0 & zeros(2,4);
for ii = 1:2
                iccc=CCQR_data_start-1+eol(ii);  # index of CCQR data within file_contents
                CCQR(2*ii-1,:)= file_contents(iccc+4:iccc+7) == "1";
                CCQR(2*ii,:)= file_contents(iccc:iccc+3) == "1";
                CCQR_dontcare(2*ii-1,:)= (file_contents(iccc+4:iccc+7) == "x") | (file_contents(iccc+4:iccc+7) ==
"X");
                CCQR_dontcare(2*ii,:)= (file_contents(iccc:iccc+3) == "x") | (file_contents(iccc:iccc+3) == "X");
endfor

# locate the XSTATUS data block
first_XSTATUS_data=findstr(file_contents, "XSTATUS:");
if isempty(first_XSTATUS_data)
                printf("Mssg. from get_mcb_data: Data format in file %s is not as expected.\n", filename);
                printf("Returning empty.\n");
                mcb = zeros(0);
                return
endif

XSTATUS_data_start= first_XSTATUS_data(1)-10;
eol= findstr(file_contents(XSTATUS_data_start-1:count),"\n"); # find the EOLs in the remaining data

XSDATA= 0 & zeros(8,4);
YSDATA= 0 & zeros(8,4);
XSDATA_dontcare= 0 & zeros(8,4);
YSDATA_dontcare= 0 & zeros(8,4);
XSDATA_Dontcare= 0 & zeros(8,4);
YSDATA_Dontcare= 0 & zeros(8,4);

XSDATA(1,:)= file_contents(XSTATUS_data_start-1+eol(1)+4:XSTATUS_data_start-1+eol(1)+7) == "1";
XSDATA(2,:)= file_contents(XSTATUS_data_start-1+eol(1):XSTATUS_data_start-1+eol(1)+3) == "1";
YSDATA(1,:)= file_contents(XSTATUS_data_start-1+eol(2)+4:XSTATUS_data_start-1+eol(2)+7) == "1";
YSDATA(2,:)= file_contents(XSTATUS_data_start-1+eol(2):XSTATUS_data_start-1+eol(2)+3) == "1";

XSDATA(3,:)= file_contents(XSTATUS_data_start-1+eol(3)+4:XSTATUS_data_start-1+eol(3)+7) == "1";
XSDATA(4,:)= file_contents(XSTATUS_data_start-1+eol(3):XSTATUS_data_start-1+eol(3)+3) == "1";
YSDATA(3,:)= file_contents(XSTATUS_data_start-1+eol(4)+4:XSTATUS_data_start-1+eol(4)+7) == "1";
YSDATA(4,:)= file_contents(XSTATUS_data_start-1+eol(4):XSTATUS_data_start-1+eol(4)+3) == "1";

XSDATA(5,:)= file_contents(XSTATUS_data_start-1+eol(5)+4:XSTATUS_data_start-1+eol(5)+7) == "1";
XSDATA(6,:)= file_contents(XSTATUS_data_start-1+eol(5):XSTATUS_data_start-1+eol(5)+3) == "1";
YSDATA(5,:)= file_contents(XSTATUS_data_start-1+eol(6)+4:XSTATUS_data_start-1+eol(6)+7) == "1";
YSDATA(6,:)= file_contents(XSTATUS_data_start-1+eol(6):XSTATUS_data_start-1+eol(6)+3) == "1";

XSDATA(7,:)= file_contents(XSTATUS_data_start-1+eol(7)+4:XSTATUS_data_start-1+eol(7)+7) == "1";
XSDATA(8,:)= file_contents(XSTATUS_data_start-1+eol(7):XSTATUS_data_start-1+eol(7)+3) == "1";
YSDATA(7,:)= file_contents(XSTATUS_data_start-1+eol(8)+4:XSTATUS_data_start-1+eol(8)+7) == "1";
YSDATA(8,:)= file_contents(XSTATUS_data_start-1+eol(8):XSTATUS_data_start-1+eol(8)+3) == "1";

XSDATA_dontcare(1,:)= file_contents(XSTATUS_data_start-1+eol(1)+4:XSTATUS_data_start-1+eol(1)+7) == "x";
XSDATA_dontcare(2,:)= file_contents(XSTATUS_data_start-1+eol(1):XSTATUS_data_start-1+eol(1)+3) == "x";
YSDATA_dontcare(1,:)= file_contents(XSTATUS_data_start-1+eol(2)+4:XSTATUS_data_start-1+eol(2)+7) == "x";
YSDATA_dontcare(2,:)= file_contents(XSTATUS_data_start-1+eol(2):XSTATUS_data_start-1+eol(2)+3) == "x";

XSDATA_dontcare(3,:)= file_contents(XSTATUS_data_start-1+eol(3)+4:XSTATUS_data_start-1+eol(3)+7) == "x";
XSDATA_dontcare(4,:)= file_contents(XSTATUS_data_start-1+eol(3):XSTATUS_data_start-1+eol(3)+3) == "x";
YSDATA_dontcare(3,:)= file_contents(XSTATUS_data_start-1+eol(4)+4:XSTATUS_data_start-1+eol(4)+7) == "x";
YSDATA_dontcare(4,:)= file_contents(XSTATUS_data_start-1+eol(4):XSTATUS_data_start-1+eol(4)+3) == "x";

XSDATA_dontcare(5,:)= file_contents(XSTATUS_data_start-1+eol(5)+4:XSTATUS_data_start-1+eol(5)+7) == "x";
XSDATA_dontcare(6,:)= file_contents(XSTATUS_data_start-1+eol(5):XSTATUS_data_start-1+eol(5)+3) == "x";
YSDATA_dontcare(5,:)= file_contents(XSTATUS_data_start-1+eol(6)+4:XSTATUS_data_start-1+eol(6)+7) == "x";
YSDATA_dontcare(6,:)= file_contents(XSTATUS_data_start-1+eol(6):XSTATUS_data_start-1+eol(6)+3) == "x";

XSDATA_dontcare(7,:)= file_contents(XSTATUS_data_start-1+eol(7)+4:XSTATUS_data_start-1+eol(7)+7) == "x";
XSDATA_dontcare(8,:)= file_contents(XSTATUS_data_start-1+eol(7):XSTATUS_data_start-1+eol(7)+3) == "x";
YSDATA_dontcare(7,:)= file_contents(XSTATUS_data_start-1+eol(8)+4:XSTATUS_data_start-1+eol(8)+7) == "x";
YSDATA_dontcare(8,:)= file_contents(XSTATUS_data_start-1+eol(8):XSTATUS_data_start-1+eol(8)+3) == "x";

XSDATA_Dontcare(1,:)= file_contents(XSTATUS_data_start-1+eol(1)+4:XSTATUS_data_start-1+eol(1)+7) == "X";
XSDATA_Dontcare(2,:)= file_contents(XSTATUS_data_start-1+eol(1):XSTATUS_data_start-1+eol(1)+3) == "X";
YSDATA_Dontcare(1,:)= file_contents(XSTATUS_data_start-1+eol(2)+4:XSTATUS_data_start-1+eol(2)+7) == "X";
YSDATA_Dontcare(2,:)= file_contents(XSTATUS_data_start-1+eol(2):XSTATUS_data_start-1+eol(2)+3) == "X";

XSDATA_Dontcare(3,:)= file_contents(XSTATUS_data_start-1+eol(3)+4:XSTATUS_data_start-1+eol(3)+7) == "X";
XSDATA_Dontcare(4,:)= file_contents(XSTATUS_data_start-1+eol(3):XSTATUS_data_start-1+eol(3)+3) == "X";
YSDATA_Dontcare(3,:)= file_contents(XSTATUS_data_start-1+eol(4)+4:XSTATUS_data_start-1+eol(4)+7) == "X";
YSDATA_Dontcare(4,:)= file_contents(XSTATUS_data_start-1+eol(4):XSTATUS_data_start-1+eol(4)+3) == "X";

XSDATA_Dontcare(5,:)= file_contents(XSTATUS_data_start-1+eol(5)+4:XSTATUS_data_start-1+eol(5)+7) == "X";
XSDATA_Dontcare(6,:)= file_contents(XSTATUS_data_start-1+eol(5):XSTATUS_data_start-1+eol(5)+3) == "X";
```

```
YSDATA_Dontcare(5,:)= file_contents(XSTATUS_data_start-1+eol(6)+4:XSTATUS_data_start-1+eol(6)+7) == "X";
YSDATA_Dontcare(6,:)= file_contents(XSTATUS_data_start-1+eol(6):XSTATUS_data_start-1+eol(6)+3) == "X";

XSDATA_Dontcare(7,:)= file_contents(XSTATUS_data_start-1+eol(7)+4:XSTATUS_data_start-1+eol(7)+7) == "X";
XSDATA_Dontcare(8,:)= file_contents(XSTATUS_data_start-1+eol(7):XSTATUS_data_start-1+eol(7)+3) == "X";
YSDATA_Dontcare(7,:)= file_contents(XSTATUS_data_start-1+eol(8)+4:XSTATUS_data_start-1+eol(8)+7) == "X";
YSDATA_Dontcare(8,:)= file_contents(XSTATUS_data_start-1+eol(8):XSTATUS_data_start-1+eol(8)+3) == "X";

XPHASE= 0 & zeros(8,4);
YPHASE= 0 & zeros(8,4);
XPHASE(1,:)= file_contents(XSTATUS_data_start-1+eol(9)+4:XSTATUS_data_start-1+eol(9)+7) == "1";
XPHASE(2,:)= file_contents(XSTATUS_data_start-1+eol(9):XSTATUS_data_start-1+eol(9)+3) == "1";
YPHASE(1,:)= file_contents(XSTATUS_data_start-1+eol(10)+4:XSTATUS_data_start-1+eol(10)+7) == "1";
YPHASE(2,:)= file_contents(XSTATUS_data_start-1+eol(10):XSTATUS_data_start-1+eol(10)+3) == "1";

XPHASE(3,:)= file_contents(XSTATUS_data_start-1+eol(11)+4:XSTATUS_data_start-1+eol(11)+7) == "1";
XPHASE(4,:)= file_contents(XSTATUS_data_start-1+eol(11):XSTATUS_data_start-1+eol(11)+3) == "1";
YPHASE(3,:)= file_contents(XSTATUS_data_start-1+eol(12)+4:XSTATUS_data_start-1+eol(12)+7) == "1";
YPHASE(4,:)= file_contents(XSTATUS_data_start-1+eol(12):XSTATUS_data_start-1+eol(12)+3) == "1";

XPHASE(5,:)= file_contents(XSTATUS_data_start-1+eol(13)+4:XSTATUS_data_start-1+eol(13)+7) == "1";
XPHASE(6,:)= file_contents(XSTATUS_data_start-1+eol(13):XSTATUS_data_start-1+eol(13)+3) == "1";
YPHASE(5,:)= file_contents(XSTATUS_data_start-1+eol(14)+4:XSTATUS_data_start-1+eol(14)+7) == "1";
YPHASE(6,:)= file_contents(XSTATUS_data_start-1+eol(14):XSTATUS_data_start-1+eol(14)+3) == "1";

XPHASE(7,:)= file_contents(XSTATUS_data_start-1+eol(15)+4:XSTATUS_data_start-1+eol(15)+7) == "1";
XPHASE(8,:)= file_contents(XSTATUS_data_start-1+eol(15):XSTATUS_data_start-1+eol(15)+3) == "1";
YPHASE(7,:)= file_contents(XSTATUS_data_start-1+eol(16)+4:XSTATUS_data_start-1+eol(16)+7) == "1";
YPHASE(8,:)= file_contents(XSTATUS_data_start-1+eol(16):XSTATUS_data_start-1+eol(16)+3) == "1";

XPHASE_dontcare= 0 & zeros(8,4);
YPHASE_dontcare= 0 & zeros(8,4);
XPHASE_Dontcare= 0 & zeros(8,4);
YPHASE_Dontcare= 0 & zeros(8,4);
XPHASE_dontcare(1,:)= file_contents(XSTATUS_data_start-1+eol(9)+4:XSTATUS_data_start-1+eol(9)+7) == "x";
XPHASE_dontcare(2,:)= file_contents(XSTATUS_data_start-1+eol(9):XSTATUS_data_start-1+eol(9)+3) == "x";
YPHASE_dontcare(1,:)= file_contents(XSTATUS_data_start-1+eol(10)+4:XSTATUS_data_start-1+eol(10)+7) == "x";
YPHASE_dontcare(2,:)= file_contents(XSTATUS_data_start-1+eol(10):XSTATUS_data_start-1+eol(10)+3) == "x";

XPHASE_dontcare(3,:)= file_contents(XSTATUS_data_start-1+eol(11)+4:XSTATUS_data_start-1+eol(11)+7) == "x";
XPHASE_dontcare(4,:)= file_contents(XSTATUS_data_start-1+eol(11):XSTATUS_data_start-1+eol(11)+3) == "x";
YPHASE_dontcare(3,:)= file_contents(XSTATUS_data_start-1+eol(12)+4:XSTATUS_data_start-1+eol(12)+7) == "x";
YPHASE_dontcare(4,:)= file_contents(XSTATUS_data_start-1+eol(12):XSTATUS_data_start-1+eol(12)+3) == "x";

XPHASE_dontcare(5,:)= file_contents(XSTATUS_data_start-1+eol(13)+4:XSTATUS_data_start-1+eol(13)+7) == "x";
XPHASE_dontcare(6,:)= file_contents(XSTATUS_data_start-1+eol(13):XSTATUS_data_start-1+eol(13)+3) == "x";
YPHASE_dontcare(5,:)= file_contents(XSTATUS_data_start-1+eol(14)+4:XSTATUS_data_start-1+eol(14)+7) == "x";
YPHASE_dontcare(6,:)= file_contents(XSTATUS_data_start-1+eol(14):XSTATUS_data_start-1+eol(14)+3) == "x";

XPHASE_dontcare(7,:)= file_contents(XSTATUS_data_start-1+eol(15)+4:XSTATUS_data_start-1+eol(15)+7) == "x";
XPHASE_dontcare(8,:)= file_contents(XSTATUS_data_start-1+eol(15):XSTATUS_data_start-1+eol(15)+3) == "x";
YPHASE_dontcare(7,:)= file_contents(XSTATUS_data_start-1+eol(16)+4:XSTATUS_data_start-1+eol(16)+7) == "x";
YPHASE_dontcare(8,:)= file_contents(XSTATUS_data_start-1+eol(16):XSTATUS_data_start-1+eol(16)+3) == "x";

XPHASE_Dontcare(1,:)= file_contents(XSTATUS_data_start-1+eol(9)+4:XSTATUS_data_start-1+eol(9)+7) == "X";
XPHASE_Dontcare(2,:)= file_contents(XSTATUS_data_start-1+eol(9):XSTATUS_data_start-1+eol(9)+3) == "X";
YPHASE_Dontcare(1,:)= file_contents(XSTATUS_data_start-1+eol(10)+4:XSTATUS_data_start-1+eol(10)+7) == "X";
YPHASE_Dontcare(2,:)= file_contents(XSTATUS_data_start-1+eol(10):XSTATUS_data_start-1+eol(10)+3) == "X";

XPHASE_Dontcare(3,:)= file_contents(XSTATUS_data_start-1+eol(11)+4:XSTATUS_data_start-1+eol(11)+7) == "X";
XPHASE_Dontcare(4,:)= file_contents(XSTATUS_data_start-1+eol(11):XSTATUS_data_start-1+eol(11)+3) == "X";
YPHASE_Dontcare(3,:)= file_contents(XSTATUS_data_start-1+eol(12)+4:XSTATUS_data_start-1+eol(12)+7) == "X";
YPHASE_Dontcare(4,:)= file_contents(XSTATUS_data_start-1+eol(12):XSTATUS_data_start-1+eol(12)+3) == "X";

XPHASE_Dontcare(5,:)= file_contents(XSTATUS_data_start-1+eol(13)+4:XSTATUS_data_start-1+eol(13)+7) == "X";
XPHASE_Dontcare(6,:)= file_contents(XSTATUS_data_start-1+eol(13):XSTATUS_data_start-1+eol(13)+3) == "X";
YPHASE_Dontcare(5,:)= file_contents(XSTATUS_data_start-1+eol(14)+4:XSTATUS_data_start-1+eol(14)+7) == "X";
YPHASE_Dontcare(6,:)= file_contents(XSTATUS_data_start-1+eol(14):XSTATUS_data_start-1+eol(14)+3) == "X";

XPHASE_Dontcare(7,:)= file_contents(XSTATUS_data_start-1+eol(15)+4:XSTATUS_data_start-1+eol(15)+7) == "X";
XPHASE_Dontcare(8,:)= file_contents(XSTATUS_data_start-1+eol(15):XSTATUS_data_start-1+eol(15)+3) == "X";
YPHASE_Dontcare(7,:)= file_contents(XSTATUS_data_start-1+eol(16)+4:XSTATUS_data_start-1+eol(16)+7) == "X";
YPHASE_Dontcare(8,:)= file_contents(XSTATUS_data_start-1+eol(16):XSTATUS_data_start-1+eol(16)+3) == "X";

XDVALID= 0 & zeros(8,1);
YDVALID= 0 & zeros(8,1);
XDVALID(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(17):\
                             XSTATUS_data_start-1+eol(17)+7) == "1",8,1));
YDVALID(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(18):\
                             XSTATUS_data_start-1+eol(18)+7) == "1",8,1));

XDVALID_dontcare= 0 & zeros(8,1);
YDVALID_dontcare= 0 & zeros(8,1);
XDVALID_Dontcare= 0 & zeros(8,1);
YDVALID_Dontcare= 0 & zeros(8,1);
XDVALID_dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(17):\
                             XSTATUS_data_start-1+eol(17)+7) == "x",8,1));
```

```
YDVALID_dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(18):\
                              XSTATUS_data_start-1+eol(18)+7) == "x",8,1));

XDVALID_Dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(17):\
                              XSTATUS_data_start-1+eol(17)+7) == "X",8,1));
YDVALID_Dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(18):\
                              XSTATUS_data_start-1+eol(18)+7) == "X",8,1));

# Quantities that correspond to input lines or cells should be ordered so that
# the line number or cell number increases with row index (hence, the flipud operation).
X_SE_CLK= 0 & zeros(8,1);
Y_SE_CLK= 0 & zeros(8,1);
X_SE_CLK(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(19):\
                              XSTATUS_data_start-1+eol(19)+7) == "1",8,1));
Y_SE_CLK(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(20):\
                              XSTATUS_data_start-1+eol(20)+7) == "1",8,1));

X_SE_CLK_dontcare= 0 & zeros(8,1);
Y_SE_CLK_dontcare= 0 & zeros(8,1);
X_SE_CLK_Dontcare= 0 & zeros(8,1);
Y_SE_CLK_Dontcare= 0 & zeros(8,1);
X_SE_CLK_dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(19):\
                              XSTATUS_data_start-1+eol(19)+7) == "x",8,1));
Y_SE_CLK_dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(20):\
                              XSTATUS_data_start-1+eol(20)+7) == "x",8,1));

X_SE_CLK_Dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(19):\
                              XSTATUS_data_start-1+eol(19)+7) == "X",8,1));
Y_SE_CLK_Dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(20):\
                              XSTATUS_data_start-1+eol(20)+7) == "X",8,1));

X_DUMP_EN= 0 & zeros(8,1);
Y_DUMP_EN= 0 & zeros(8,1);
X_DUMP_EN(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(21):\
                              XSTATUS_data_start-1+eol(21)+7) == "1",8,1));
Y_DUMP_EN(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(22):\
                              XSTATUS_data_start-1+eol(22)+7) == "1",8,1));

X_DUMP_EN_dontcare= 0 & zeros(8,1);
Y_DUMP_EN_dontcare= 0 & zeros(8,1);
X_DUMP_EN_Dontcare= 0 & zeros(8,1);
Y_DUMP_EN_Dontcare= 0 & zeros(8,1);
X_DUMP_EN_dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(21):\
                              XSTATUS_data_start-1+eol(21)+7) == "x",8,1));
Y_DUMP_EN_dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(22):\
                              XSTATUS_data_start-1+eol(22)+7) == "x",8,1));

X_DUMP_EN_Dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(21):\
                              XSTATUS_data_start-1+eol(21)+7) == "X",8,1));
Y_DUMP_EN_Dontcare(:,1)= flipud(reshape(file_contents(XSTATUS_data_start-1+eol(22):\
                              XSTATUS_data_start-1+eol(22)+7) == "X",8,1));

X_TIMESTAMP= file_contents(XSTATUS_data_start-1+eol(23)+7) == "1";
X_SCHID_SYNC= file_contents(XSTATUS_data_start-1+eol(23)+6) == "1";
X_DUMP_SYNC= file_contents(XSTATUS_data_start-1+eol(23)+5) == "1";

Y_TIMESTAMP= file_contents(XSTATUS_data_start-1+eol(24)+7) == "1";
Y_SCHID_SYNC= file_contents(XSTATUS_data_start-1+eol(24)+6) == "1";
Y_DUMP_SYNC= file_contents(XSTATUS_data_start-1+eol(24)+5) == "1";

X_TIMESTAMP_dontcare= (file_contents(XSTATUS_data_start-1+eol(23)+7) == "x") | ...
                                                    (file_contents(XSTATUS_data_start-1+eol(23)+7) == "X");
X_SCHID_SYNC_dontcare= (file_contents(XSTATUS_data_start-1+eol(23)+6) == "x") | ...
                                                    (file_contents(XSTATUS_data_start-1+eol(23)+6) == "X");
X_DUMP_SYNC_dontcare= (file_contents(XSTATUS_data_start-1+eol(23)+5) == "x") | ...
                                                    (file_contents(XSTATUS_data_start-1+eol(23)+5) == "X");

Y_TIMESTAMP_dontcare= (file_contents(XSTATUS_data_start-1+eol(24)+7) == "x") | ...
                                                    (file_contents(XSTATUS_data_start-1+eol(24)+7) == "X");
Y_SCHID_SYNC_dontcare= (file_contents(XSTATUS_data_start-1+eol(24)+6) == "x") | ...
                                                    (file_contents(XSTATUS_data_start-1+eol(24)+6) == "X");
Y_DUMP_SYNC_dontcare= (file_contents(XSTATUS_data_start-1+eol(24)+5) == "x") | ...
                                                    (file_contents(XSTATUS_data_start-1+eol(24)+5) == "X");

# locate the DESSR data block
first_DESSR_data=findstr(file_contents, "DESSR_0_7:");
if isempty(first_DESSR_data)
              printf("Mssg. from get_mcb_data: Data format in file %s is not as expected.\n", filename);
              printf("Returning empty.\n");
              mcb = zeros(0);
              return
endif

DESSR_data_start= first_DESSR_data(1)-10;
eol= findstr(file_contents(DESSR_data_start-1:count),"\n"); # find the EOLs in the remaining data
DESSR= 0 & zeros(1,16);
DESSR(9:16)= file_contents(DESSR_data_start-1+eol(1):DESSR_data_start-1+eol(1)+7) == "1";
```

```
DESSR(1:8)= file_contents(DESSR_data_start-1+eol(2):DESSR_data_start-1+eol(2)+7) == "1";
DESSR= flipud(reshape(DESSR,16,1));  # flip to maintain line number increase with row index

DESSR_dontcare= 0 & zeros(1,16);
DESSR_dontcare(9:16)= (file_contents(DESSR_data_start-1+eol(1):DESSR_data_start-1+eol(1)+7) == "x") | ...
                                                (file_contents(DESSR_data_start-1+eol(1):DESSR_data_start-1+eol(1)+7) ==
"X");
DESSR_dontcare(1:8)= (file_contents(DESSR_data_start-1+eol(2):DESSR_data_start-1+eol(2)+7) == "x") | ...
                                                (file_contents(DESSR_data_start-1+eol(2):DESSR_data_start-1+eol(2)+7) ==
"X");
DESSR_dontcare= flipud(reshape(DESSR_dontcare,16,1));  # flip to maintain line number increase with row index


# Load registers into the structure.
# The bits of which these flags represent decrease in significance (from MSB to LSB) as
# the index of the row or column increase. Hence, index 1 refers to the MSB.

# For some reason, the boolean type attribute was dropped during the matrix assignments.
# Perorm a boolean operation to convert back before adding to the structure.


mcb.MCSR.CEN= MCSR(8) == 1;
mcb.MCSR.TVEN= MCSR(7) == 1;
mcb.MCSR.PhEN= MCSR(6) == 1;
mcb.MCSR.OVR= MCSR(5) == 1;
mcb.MCSR.AOV= MCSR(4) == 1;
mcb.MCSR.TvER= MCSR(3) == 1;
mcb.MCSR.SyncER= MCSR(2) == 1;
mcb.MCSR.XYDS= MCSR(1) == 1;
mcb.CCCSCR= CCCSCR == 1;
mcb.CCQR= CCQR == 1;
mcb.X_DATA= XSDATA == 1;
mcb.X_PHASE= XPHASE == 1;
mcb.X_DVALID= XDVALID == 1;
mcb.Y_DATA= YSDATA == 1;
mcb.Y_PHASE= YPHASE == 1;
mcb.Y_DVALID= YDVALID == 1;
mcb.X_SE_CLK= X_SE_CLK == 1;
mcb.X_DUMP_EN= X_DUMP_EN == 1;
mcb.X_TIMESTAMP= X_TIMESTAMP == 1;
mcb.X_SCHID_SYNC= X_SCHID_SYNC == 1;
mcb.X_DUMP_SYNC= X_DUMP_SYNC == 1;
mcb.Y_SE_CLK= Y_SE_CLK == 1;
mcb.Y_DUMP_EN= Y_DUMP_EN == 1;
mcb.Y_TIMESTAMP= Y_TIMESTAMP == 1;
mcb.Y_SCHID_SYNC= Y_SCHID_SYNC == 1;
mcb.Y_DUMP_SYNC= Y_DUMP_SYNC == 1;
mcb.DESSR= DESSR == 1;
mcb.Q0.X0= CCQR(1,4) == 1;
mcb.Q0.Y0= CCQR(1,2) == 1;
mcb.Q1.X0= CCQR(2,4) == 1;
mcb.Q1.Y0= CCQR(2,2) == 1;
mcb.Q2.X0= CCQR(3,4) == 1;
mcb.Q2.Y0= CCQR(3,2) == 1;
mcb.Q3.X0= CCQR(4,4) == 1;
mcb.Q3.Y0= CCQR(4,2) == 1;
mcb.Q0.X1= CCQR(1,3) == 1;
mcb.Q0.Y1= CCQR(1,1) == 1;
mcb.Q1.X1= CCQR(2,3) == 1;
mcb.Q1.Y1= CCQR(2,1) == 1;
mcb.Q2.X1= CCQR(3,3) == 1;
mcb.Q2.Y1= CCQR(3,1) == 1;
mcb.Q3.X1= CCQR(4,3) == 1;
mcb.Q3.Y1= CCQR(4,1) == 1;
mcb.XS0= CCCSCR(:,4) == 1;
mcb.XS1= CCCSCR(:,3) == 1;
mcb.YS0= CCCSCR(:,2) == 1;
mcb.YS1= CCCSCR(:,1) == 1;

mcb.dontcare.MCSR.CEN= MCSR_dontcare(8) == 1;
mcb.dontcare.MCSR.TVEN= MCSR_dontcare(7) == 1;
mcb.dontcare.MCSR.PhEN= MCSR_dontcare(6) == 1;
mcb.dontcare.MCSR.OVR= MCSR_dontcare(5) == 1;
mcb.dontcare.MCSR.AOV= MCSR_dontcare(4) == 1;
mcb.dontcare.MCSR.TvER= MCSR_dontcare(3) == 1;
mcb.dontcare.MCSR.SyncER= MCSR_dontcare(2) == 1;
mcb.dontcare.MCSR.XYDS= MCSR_dontcare(1) == 1;
mcb.dontcare.CCCSCR= CCCSCR_dontcare == 1;
mcb.dontcare.CCQR= CCQR_dontcare == 1;
mcb.dontcare.X_DATA= (XSDATA_dontcare == 1) | (XSDATA_Dontcare == 1);
mcb.dontcare.X_PHASE= (XPHASE_dontcare == 1) | (XPHASE_Dontcare == 1);
mcb.dontcare.X_DVALID= (XDVALID_dontcare == 1) | (XDVALID_Dontcare == 1);
mcb.dontcare.Y_DATA= (YSDATA_dontcare == 1) | (YSDATA_Dontcare == 1);
mcb.dontcare.Y_PHASE= (YPHASE_dontcare == 1) | (YPHASE_Dontcare == 1);
mcb.dontcare.Y_DVALID= (YDVALID_dontcare == 1) | (YDVALID_Dontcare == 1);
mcb.dontcare.X_SE_CLK= (X_SE_CLK_dontcare == 1) | (X_SE_CLK_Dontcare == 1);
mcb.dontcare.X_DUMP_EN= (X_DUMP_EN_dontcare == 1) | (X_DUMP_EN_Dontcare == 1);
```

```
mcb.dontcare.X_TIMESTAMP= X_TIMESTAMP_dontcare == 1;
mcb.dontcare.X_SCHID_SYNC= X_SCHID_SYNC_dontcare == 1;
mcb.dontcare.X_DUMP_SYNC= X_DUMP_SYNC_dontcare == 1;
mcb.dontcare.Y_SE_CLK= (Y_SE_CLK_dontcare == 1) | (Y_SE_CLK_Dontcare == 1);
mcb.dontcare.Y_DUMP_EN= (Y_DUMP_EN_dontcare == 1) | (Y_DUMP_EN_Dontcare == 1);
mcb.dontcare.Y_TIMESTAMP= Y_TIMESTAMP_dontcare == 1;
mcb.dontcare.Y_SCHID_SYNC= Y_SCHID_SYNC_dontcare == 1;
mcb.dontcare.Y_DUMP_SYNC= Y_DUMP_SYNC_dontcare == 1;
mcb.dontcare.DESSR= DESSR_dontcare == 1;
mcb.dontcare.Q0.X0= CCQR_dontcare(1,4) == 1;
mcb.dontcare.Q0.Y0= CCQR_dontcare(1,2) == 1;
mcb.dontcare.Q1.X0= CCQR_dontcare(2,4) == 1;
mcb.dontcare.Q1.Y0= CCQR_dontcare(2,2) == 1;
mcb.dontcare.Q2.X0= CCQR_dontcare(3,4) == 1;
mcb.dontcare.Q2.Y0= CCQR_dontcare(3,2) == 1;
mcb.dontcare.Q3.X0= CCQR_dontcare(4,4) == 1;
mcb.dontcare.Q3.Y0= CCQR_dontcare(4,2) == 1;
mcb.dontcare.Q0.X1= CCQR_dontcare(1,3) == 1;
mcb.dontcare.Q0.Y1= CCQR_dontcare(1,1) == 1;
mcb.dontcare.Q1.X1= CCQR_dontcare(2,3) == 1;
mcb.dontcare.Q1.Y1= CCQR_dontcare(2,1) == 1;
mcb.dontcare.Q2.X1= CCQR_dontcare(3,3) == 1;
mcb.dontcare.Q2.Y1= CCQR_dontcare(3,1) == 1;
mcb.dontcare.Q3.X1= CCQR_dontcare(4,3) == 1;
mcb.dontcare.Q3.Y1= CCQR_dontcare(4,1) == 1;
mcb.dontcare.XS0= CCCSCR_dontcare(:,4) == 1;
mcb.dontcare.XS1= CCCSCR_dontcare(:,3) == 1;
mcb.dontcare.YS0= CCCSCR_dontcare(:,2) == 1;
mcb.dontcare.YS1= CCCSCR_dontcare(:,1) == 1;
endfunction
```

3.24.5 make_mcb_chains

```
function [Cchain, config, unchain]= make_mcb_chains(mcb)
#
# usage:[Cchain, config, unchain]= make_mcb_chains(mcb)
#
# description: create correlator cell chains based on the
#              MCB register settings. This function will
#              abort in the presence of corrupted
#              configuration data (don't care bits
#              encountered in configuration registers).
#
#
# mcb: structure containing the CCCSCR and CCQR register
#      contents
#
# Cchain.length: the length in cells of this chain;
#
# Cchain.CCC: the array of cells in this chain
#
# Cchain.Xinput: the selected X input to this chain.
#                Chains with undefined inputs are excluded.
#
# Cchain.Yinput: the selected Y input to this chain,
#                -1 indicates "no-data". Chains with
#                undefined inputs are excluded.
#
# config: a string describing the input data configuration
#         for the chip; "-" indicates a "no-data" input,
#         cells with undefined inputs are excluded.
#
# unchain.undefined_X: boolean array, true for cells with
#                      undefined X inputs or undefined by
#                      concatenation with undefined X.
#
# unchain.undefined_Y: boolean array, true for cells with
#                      undefined Y inputs or undefined by
#                      concatenation with undefined Y.
#
# unchain.Xin: boolean array, true for cells where the
#              X input is from adjacent cells.
#
# unchain.Yin: boolean array:, true for cells where the
#              Y input is from adjacent cells.
#
#
## [Ver_date]= make_mcb_chains();
## Ver_date: version number: date this version was created

## ********************** Revision history **********************
## (version; date created: description of changes)
## 1.00: first working version
```

```
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
## 1.10; Nov.  2, 2004: check for corrupt configuration data;
##                      update data routing to Q0 in accordance
##                      with revised design.
##
## ************************************************************

Version_date= "1.10: 04-11-02";
if nargin == 0
               Cchain= Version_date;
               return;
endif

if !isfield(mcb,"XS1") | !isfield(mcb,"XS0") | !isfield(mcb,"YS1") | !isfield(mcb,"YS0")
               printf("ERROR from make_mcb_chains: The input data structure\n");
               printf("appears incorrect. Returning.\n");
               return;
endif

if (any(any(mcb.dontcare.CCQR)) | any(any(mcb.dontcare.CCCSCR)))
               printf("ERROR from make_mcb_chains: Corrupted configuration\n");
               printf("register contents (don't care bits). Returning.\n");
endif

# Find all undefined (no-lag) cells. Mark all cells concatenated in either X or Y
# with undefined cells as undefined. The adjacent X connections on the first cell and
# adjacent Y connections on the last cell are undefined.
first_cell_mask= zeros(16,1);
last_cell_mask= zeros(16,1);
first_cell_mask(1,1)= 1;
last_cell_mask(16,1)=1;
undefined_X = (mcb.XS1 & mcb.XS0) | (!mcb.XS1 & !mcb.XS0 & first_cell_mask);
undefined_Y = (mcb.YS1 & mcb.YS0) | (!mcb.YS1 & !mcb.YS0 & last_cell_mask);
for ik= 1:15  # any cells concatenated with undefined cells are undefined.
               if undefined_X(ik) & (!mcb.XS1(ik+1) & !mcb.XS0(ik+1))
                              undefined_X(ik+1)= undefined_X(ik) & 1;
               endif
               if undefined_Y(17-ik) & (!mcb.YS1(16-ik) & !mcb.YS0(16-ik))
                              undefined_Y(16-ik)= undefined_Y(17-ik) & 1;
               endif
endfor

undefined= undefined_X | undefined_Y;
unchain.undefined_X= undefined_X;
unchain.undefined_Y= undefined_Y;
unchain.Xin= !mcb.XS1 & !mcb.XS0;
unchain.Yin= !mcb.YS1 & !mcb.YS0;

# Locate chain start and ends, including single cells, excluding cells with undefined inputs.
# The cell is undefined if either the x or y switches are in the undefined state.
# The entire chain is undefined if either start, end or both are undefined.

starts = find(!(!mcb.XS1 & !mcb.XS0) & !undefined);
ends = find(!(!mcb.YS1 & !mcb.YS0) & !undefined);

if rows(starts) != rows(ends) | !all(starts <= ends)
               printf("There is a problem with the cell input configuration.\n");
               return;
endif

nchains= rows(starts);
config= "";  # initialize configuration string
for ii=1:nchains
               Cchain(ii).length= (ends(ii)-starts(ii) + 1);
               Cchain(ii).CCC= reshape((starts(ii):ends(ii))-1,Cchain(ii).length,1);

               # the following data routing is based on Table 5-2 in RFS document A25082N0000 Rev.1.3
               if (undefined(starts(ii)) | undefined(ends(ii)))
                              continue;  # skip the following part of this iteration for undefined (no-lag) chains
               endif
               startcell= starts(ii);
               if ((startcell-1) <= 3) && (!mcb.XS1(startcell) & mcb.XS0(startcell)) && !mcb.Q0.X0
                              Cchain(ii).Xinput= 0;
               elseif ((startcell-1) <= 3) && (!mcb.XS1(startcell) & mcb.XS0(startcell)) && mcb.Q0.X0
                              Cchain(ii).Xinput= 6;  # became input 6 in revised design (Nov. 1, 2004)
               elseif ((startcell-1) <= 3) && (mcb.XS1(startcell) & !mcb.XS0(startcell)) && !mcb.Q0.X1
                              Cchain(ii).Xinput= 1;
               elseif ((startcell-1) <= 3) && (mcb.XS1(startcell) & !mcb.XS0(startcell)) && mcb.Q0.X1
                              Cchain(ii).Xinput= 7;  # became input 7 in revised design (Nov. 1, 2004)

               elseif ((startcell-1) <= 7) && (4 <= (startcell-1)) &&\
                                                              (!mcb.XS1(startcell) & mcb.XS0(startcell)) && !mcb.Q1.X0
                              Cchain(ii).Xinput= 2;
               elseif ((startcell-1) <= 7) && (4 <= (startcell-1)) &&\
                                                              (!mcb.XS1(startcell) & mcb.XS0(startcell)) && mcb.Q1.X0
                              Cchain(ii).Xinput= 0;
               elseif ((startcell-1) <= 7) && (4 <= (startcell-1)) &&\
```

```
                                                (mcb.XS1(startcell) & !mcb.XS0(startcell)) && !mcb.Q1.X1
                Cchain(ii).Xinput= 3;
elseif ((startcell-1) <= 7) && (4 <= (startcell-1)) &&\
                                                (mcb.XS1(startcell) & !mcb.XS0(startcell)) && mcb.Q1.X1
                Cchain(ii).Xinput= 1;

elseif ((startcell-1) <= 11) && (8 <= (startcell-1)) &&\
                                                (!mcb.XS1(startcell) & mcb.XS0(startcell)) && !mcb.Q2.X0
                Cchain(ii).Xinput= 4;
elseif ((startcell-1) <= 11) && (8 <= (startcell-1)) &&\
                                                (!mcb.XS1(startcell) & mcb.XS0(startcell)) && mcb.Q2.X0
                Cchain(ii).Xinput= 0;
elseif ((startcell-1) <= 11) && (8 <= (startcell-1)) &&\
                                                (mcb.XS1(startcell) & !mcb.XS0(startcell)) && !mcb.Q2.X1
                Cchain(ii).Xinput= 5;
elseif ((startcell-1) <= 11) && (8 <= (startcell-1)) &&\
                                                (mcb.XS1(startcell) & !mcb.XS0(startcell)) && mcb.Q2.X1
                Cchain(ii).Xinput= 1;

elseif ((startcell-1) <= 15) && (12 <= (startcell-1)) &&\
                                                (!mcb.XS1(startcell) & mcb.XS0(startcell)) && !mcb.Q3.X0
                Cchain(ii).Xinput= 6;
elseif ((startcell-1) <= 15) && (12 <= (startcell-1)) &&\
                                                (!mcb.XS1(startcell) & mcb.XS0(startcell)) && mcb.Q3.X0
                Cchain(ii).Xinput= 0;
elseif ((startcell-1) <= 15) && (12 <= (startcell-1)) &&\
                                                (mcb.XS1(startcell) & !mcb.XS0(startcell)) && !mcb.Q3.X1
                Cchain(ii).Xinput= 7;
else Cchain(ii).Xinput= 1;  # (mcb.XS1(startcell) & !mcb.XS0(startcell)) && mcb.Q3.X1
                Cchain(ii).Xinput= 1;

endif

endcell= ends(ii);
if ((endcell-1) <= 3) && (!mcb.YS1(endcell) & mcb.YS0(endcell)) && !mcb.Q0.Y1
                Cchain(ii).Yinput= 1;
elseif ((endcell-1) <= 3) && (!mcb.YS1(endcell) & mcb.YS0(endcell)) && mcb.Q0.Y1
                Cchain(ii).Yinput= -1;  # no-data input
elseif ((endcell-1) <= 3) && (mcb.YS1(endcell) & !mcb.YS0(endcell)) && !mcb.Q0.Y0
                Cchain(ii).Yinput= 0;
elseif ((endcell-1) <= 3) && (mcb.YS1(endcell) & !mcb.YS0(endcell)) && mcb.Q0.Y0
                Cchain(ii).Yinput= -1;  # no-data input

elseif ((endcell-1) <= 7) && (4 <= (endcell-1)) &&\
                                                (!mcb.YS1(endcell) & mcb.YS0(endcell)) && !mcb.Q1.Y1
                Cchain(ii).Yinput= 3;
elseif ((endcell-1) <= 7) && (4 <= (endcell-1)) &&\
                                                (!mcb.YS1(endcell) & mcb.YS0(endcell)) && mcb.Q1.Y1
                Cchain(ii).Yinput= 1;
elseif ((endcell-1) <= 7) && (4 <= (endcell-1)) &&\
                                                (mcb.YS1(endcell) & !mcb.YS0(endcell)) && !mcb.Q1.Y0
                Cchain(ii).Yinput= 2;
elseif ((endcell-1) <= 7) && (4 <= (endcell-1)) &&\
                                                (mcb.YS1(endcell) & !mcb.YS0(endcell)) && mcb.Q1.Y0
                Cchain(ii).Yinput= 0;

elseif ((endcell-1) <= 11) && (8 <= (endcell-1)) &&\
                                                (!mcb.YS1(endcell) & mcb.YS0(endcell)) && !mcb.Q2.Y1
                Cchain(ii).Yinput= 5;
elseif ((endcell-1) <= 11) && (8 <= (endcell-1)) &&\
                                                (!mcb.YS1(endcell) & mcb.YS0(endcell)) && mcb.Q2.Y1
                Cchain(ii).Yinput= 1;
elseif ((endcell-1) <= 11) && (8 <= (endcell-1)) &&\
                                                (mcb.YS1(endcell) & !mcb.YS0(endcell)) && !mcb.Q2.Y0
                Cchain(ii).Yinput= 4;
elseif ((endcell-1) <= 11) && (8 <= (endcell-1)) &&\
                                                (mcb.YS1(endcell) & !mcb.YS0(endcell)) && mcb.Q2.Y0
                Cchain(ii).Yinput= 0;

elseif ((endcell-1) <= 15) && (12 <= (endcell-1)) &&\
                                                (!mcb.YS1(endcell) & mcb.YS0(endcell)) && !mcb.Q3.Y1
                Cchain(ii).Yinput= 7;
elseif ((endcell-1) <= 15) && (12 <= (endcell-1)) &&\
                                                (!mcb.YS1(endcell) & mcb.YS0(endcell)) && mcb.Q3.Y1
                Cchain(ii).Yinput= 1;
elseif ((endcell-1) <= 15) && (12 <= (endcell-1)) &&\
                                                (mcb.YS1(endcell) & !mcb.YS0(endcell)) && !mcb.Q3.Y0
                Cchain(ii).Yinput= 6;
else Cchain(ii).Yinput= 1;  # (mcb.YS1(endcell) & !mcb.YS0(endcell)) && mcb.Q3.Y0
                Cchain(ii).Yinput= 0;

endif
if Cchain(ii).Xinput == -1 & Cchain(ii).Yinput >= 0
                ccon=sprintf("-(%d:%d)%d ",(startcell-1), (endcell-1), Cchain(ii).Yinput);
elseif Cchain(ii).Yinput == -1 & Cchain(ii).Xinput >= 0
                ccon=sprintf("%d(%d:%d)- ",Cchain(ii).Xinput, (startcell-1), (endcell-1));
elseif Cchain(ii).Xinput == -1 & Cchain(ii).Yinput == -1
```

```
                        ccon=sprintf("-(%d:%d)- ", (startcell-1), (endcell-1));
                else
                        ccon=sprintf("%d(%d:%d)%d ", Cchain(ii).Xinput,\
                         (startcell-1), (endcell-1), Cchain(ii).Yinput);
                endif
                config= strcat(config, ccon);
endfor
endfunction
```

3.24.6 get_bmodel_chains

```
function [bmodel] = get_bmodel_chains(dataspec, CCchain, maxdumps)
#
# usage: [bmodel] = get_bmodel_chains(dataspec, CCchain, maxdumps)
#
# description: read complex arrays of behavioural model data,
#              one dump per file into a chain based structure.
#
# data_spec: data descriptor string, e.g.  "test00"
#                           (data files of form: CC0x0_test00_2048lagsD1.dat)
# CCchain: 1 x nchain structure array, nchain: number of chains
# CCchain.Xinput: X input data stream number of chain ic
# CCchain.Yinput: Y input data stream number of chain ic
# CCchain.length: the length in cells of chain (ic)
# maxdumps: the maximum number of dumps to search for
# blags: complex lags in nblag x ndump array.
#
# bmodel(ic).filespec: behavioural model file specification
# bmodel(ic).length: the length in cells of chain (ic)
# bmodel(ic).Xin: X input data stream number of chain ic
# bmodel(ic).Yin: Y input data stream number of chain ic
# bmodel(ic).ndumps: the number of dumps found for chain ic
# bmodel(ic).blags: nlag x ndumps array of complex correlation
#
## [Ver_date]= get_bmodel_chains();
## Ver_date: version number: date this version was created

## ********************** Revision history **********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
##
## ************************************************************

Version_date= "1.01: 04-10-18";
if nargin == 0
                bmodel= Version_date;
                return;
endif

nchain = columns(CCchain);
for ic= 1:nchain
                nblags= CCchain(ic).length*128;
                if ((CCchain(ic).Xinput < 0) | (CCchain(ic).Yinput < 0)); continue; endif; # skip: no data
                bmodel(ic).Xin= CCchain(ic).Xinput;
                bmodel(ic).Yin= CCchain(ic).Yinput;
                bmodel(ic).length= CCchain(ic).length;
                blags= zeros(nblags, maxdumps);

                dump_count= 0; # initialize dump counter
                for id=1:maxdumps
                        filespec= sprintf("CC%dx%d_%s_%dlags",\
                                        CCchain(ic).Xinput, CCchain(ic).Yinput,dataspec, nblags);
                        bmfilename= sprintf("%sD%d.dat", filespec, (id-1));
#                       printf("Attempting read %s\n",bmfilename);
                        [info,err,mssg]=stat(bmfilename);
                        if err != 0
                                        printf("Cannot open file %s.\n", bmfilename);
                                        continue;
                        endif
                        [blags(:,id), inblags]= get_bmodel_lags(bmfilename);
                        dump_count= id;

                        if isempty(blags);
                                        printf("Data from file %s could not be obtained.\n");
                                        return;
                        endif
                        if nblags != inblags
                                        printf("The expected number of lags not read from file %s.\n",
bmfilename);
                                        return;
                        endif
```

```
                endfor
                bmodel(ic).blags= blags(:,1:dump_count);
                bmodel(ic).ndumps= dump_count;
                bmodel(ic).filespec= filespec;
endfor

endfunction
```

3.24.7 get_bmodel_lags

```
function [blags, nblags] = get_bmodel_lags(filename)
#
# usage: [blags, nblags] = get_bmodel_lags(filename)
#
# description: read a file into a complex array if data occurs in pairs, otherwise return empty.
#
#    filename: the name of the file
#    blags: a complex array of nblags elements
#
## [Ver_date]= get_bmodel_lags();
## Ver_date: version number: date this version was created

## ********************** Revision history **********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
##
## ************************************************************

Version_date= "1.01: 04-10-18";
if nargin == 0
                blags= Version_date;
                return;
endif

bmfile= fopen(filename, "r");
[blags,nblags]=fscanf(bmfile,"%e",Inf);
fclose(bmfile);
if (1 & mod(nblags,2))
                printf("The file %s does not\n",filename);
                printf("appear to contain data in real-imaginary pairs.\n Return empty.\n");
                nblags=0;
                blags=zeros(0);
                return;
endif
blags=blags(1:2:(nblags-1))+(1i)*blags(2:2:nblags);
nblags= rows(blags);

endfunction
```

3.24.8 lag2fd

```
function [fd]= lag2fd(lags, bw, Lwin)
#
# usage: [fd]= lag2fd(lags, bw, Lwin)
#        [fd]= lag2fd(lags, bw) no window applied
#
# description: converts lag data to frequency domain
#
# lags: complex lag data, one dump per column
# bw: bandwidth in Mhz
# Lwin: window specification, 0 or absent --> no window
#       1 (or greater) --> Hanning
# fd: complex frequency domain result, negative freq. first
#
## [Ver_date]= lag2fd();
## Ver_date: version number: date this version was created

## ********************** Revision history **********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
##
## ************************************************************

Version_date= "1.01: 04-10-18";
if nargin == 0
                fd= Version_date;
                return;
```

```
endif

if nargin < 3

               Lwin= fix(0);
else
               Lwin= fix(Lwin);
endif

np= rows(lags);
nd= columns(lags);
fbins = 2*bw;

if Lwin >= 1
               rwin= repmat(hanning(np),1,nd);
               fd= fbins*shift(ifft(shift(rwin .* lags, np/2)),np/2-1);
else
               fd= fbins*shift(ifft(shift(lags, np/2)),np/2-1);
endif

endfunction
```

3.24.9 chain2single

```
function [isingle, ccn]= chain2single(ichain, chain_length)
#
# usage: [isingle, ccn]= chain2single(ichain)
#
# ichain: index (first = 1) of lag in a concatenated chain
# isingle: index of lag within the single cell output frame
# ccn: index of cell within the chain (first = 1)
# chain_length: length of chain in number of cells
#
# NOTE: no range checks performed.
#
## [Ver_date]= chain2single();
## Ver_date: version number: date this version was created

## ********************* Revision history *********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
##
## ************************************************************

Version_date= "1.01: 04-10-18";
if nargin == 0
               isingle= Version_date;
               return;
endif
ccn = chain_length-fix((ichain-1)/128);
isingle = ichain - 128*(chain_length-ccn);

endfunction
```

3.24.10 single2chain

```
function [ichain]= single2chain(isingle,ccn, chain_length)
#
# usage: [ichain] = single2chain(isingle, ccn, chain_length)
#
# ichain: index (first = 1) of lag in a concatenated chain
# isingle: index of lag within the single cell output frame
# ccn: index of cell within the chain (first = 1)
# chain_length: length of chain in number of cells
#
# NOTE: no range checks performed.
#
## [Ver_date]= single2chain();
## Ver_date: version number: date this version was created

## ********************* Revision history *********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
##
## ************************************************************

Version_date= "1.01: 04-10-18";
```

```
if nargin == 0
               ichain= Version_date;
               return;
endif

ichain = isingle + 128*(chain_length-ccn);

endfunction
```

3.24.11 scale_lag_data

```
function [clags, dvcount, data_bias] = scale_lag_data(CCchain)
# usage: [clags, dvcount, data_bias] = scale_lag_data(CCchain)
#
# NOTE: zero data_bias returned if some of the cells in the
#       chain have the overrun (OVR) bit set.
#
## [Ver_date]= scale_lag_data();
## Ver_date: version number: date this version was created

## ********************* Revision history *********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
## 1.10: Nov. 21, 2004: return zero bias if OVR set in some cells
##
## ************************************************************

Version_date= "1.10: 04-11-21";
if nargin == 0
               clags= Version_date;
               return;
endif

if !(isstruct(CCchain) && isfield(CCchain, "length") && isfield(CCchain, "DATA_BIAS")...
 && isfield(CCchain, "DVCOUNT_centre") && isfield(CCchain, "DVCOUNT_edge") && isfield(CCchain, "clag"))
               printf("Input to scale_lag_data is not a structure or does not contain the required fields.\n");
               return;
endif

if (isfield(CCchain,"dvcount") && !isempty(CCchain.dvcount))
               printf("This lag data has already had bias removed and scaled by DVCOUNT.\n");
               printf("Returning empty lag vector, dvcount and data_bias.\n");
               clags=zeros(0);
               dvcount=zeros(0);
               data_bias=zeros(0);
               return;
endif

[dvcount_index, use_edge] = which_DVCOUNT(CCchain.length);

if use_edge
               dvcount = CCchain.DVCOUNT_edge(dvcount_index);
else
               dvcount = CCchain.DVCOUNT_centre(dvcount_index);
endif

if all(CCchain.DATA_BIAS == CCchain.DATA_BIAS(dvcount_index))

               data_bias = CCchain.DATA_BIAS(dvcount_index);

elseif any(!CCchain.OVR) & any(CCchain.OVR)
               data_bias= 0;
else
               printf("WARNING (from scale_lag_data): Not all data bias values in this chain are identical.\n");
               printf("                               Returning an empty lag matrix and data_bias.\n");
               data_bias = zeros(0);
               clags=zeros(0);
               return;

endif

clags = (CCchain.clag-data_bias*(1+1i))/dvcount;


endfunction
```

3.24.12 which_DVCOUNT

```
function [idv, edge]=which_DVCOUNT(ncells)
#
# usage: [idv, edge] = which_DVCOUNT(ncells)
#
# description: Given a concatenated chain of ncells in length,
#              and cell index starting from 1 to ncells,
#              provide idv, the cell index of DVCOUNT and
#              a boolean flag, edge which is true (edge = 1) for
#              the case where the DVCOUNT_edge is to be applied.
#              Use DVCOUNT_centre when edge is false.
#
## [Ver_date]= which_DVCOUNT();
## Ver_date: version number: date this version was created

## ********************* Revision history *********************
## (version; date created: description of changes)
## 1.00: first working version
## 1.01; Oct. 18, 2004: add code that returns the Version_date.
##
## ************************************************************

Version_date= "1.01: 04-10-18";
if nargin == 0
              idv= Version_date;
              return;
endif

edge = (1 & mod((ncells-1),2));
if edge
              idv = 2 + fix((ncells-1)/2);
else
              idv = 1 + fix((ncells-1)/2);
endif

idv = ncells - idv + 1;   #  consider lag numbering (increasing lag with decreasing cell number)
endfunction
```

3.24.13 list_versions

```
POSTSIM_LIB_VERSION= "1.40";
POSTSIM_LIB_VDATE= "05-08-18";

printf("postsim_lib          version: %s    date: August 18, 2005\n\n", POSTSIM_LIB_VERSION);
printf("function name \t\tversion: date \tcalls\n");
printf("%s",...
["add_CCchain_dump",          "\t",                        add_CCchain_dump,            "\t",
"scale_lag_data.m\n",...
 "chain2single",              "\t\t",                      chain2single,                "\n",...
 "get_bmodel_chains",         "\t",                        get_bmodel_chains,           "\t",
"get_bmodel_lags.m\n",...
 "get_bmodel_lags",           "\t\t",       get_bmodel_lags,             "\n",...
 "get_lag_frames",            "\t\t",       get_lag_frames,                           "\n",...
 "get_mcb_data",              "\t\t",       get_mcb_data,                "\n",...
 "lag2fd",                    "\t\t\t",     lag2fd,                      "\n",...
 "make_dump_index",           "\t\t",       make_dump_index,             "\n",...
 "make_mcb_chains",           "\t\t",       make_mcb_chains,             "\n",...
 "scale_lag_data",            "\t\t",       scale_lag_data,              "\t", "which_DVCOUNT.m\n",...
 "single2chain",              "\t\t",                    single2chain,                "\n",...
 "which_DVCOUNT",             "\t\t",       which_DVCOUNT,               "\n"]);
```

3.25 test vector editing scripts

Test vector sets DS10, DS11, DS14 and DS15 were created from DS4 by the scripts of this section. The code listings are found in their respective subsections below.

3.25.1 DS10

```
# generate data set DS10 for bit by bit comparison with 20K samples.
LOADPATH= ":/stage/rigel1/smegal/EVLA_CorrChip/testing/test_vectors";
[xamp, xpha, xdva, xnsf, xndva]= get_CCtest_data("../DS4/X_CCinput0.vhex", 50000);

xdva(1:4000)= !1;
xdva(14000:50000)= !1;
printf("Data valid count in modified X data set: %d\n",rows(find(!xdva)));
write_CCtest_data("X_CCinput0_DS10.vhex", xamp, xpha, xdva);

[yamp, ypha, ydva, ynsf, yndva]= get_CCtest_data("../DS4/Y_CCinput0.vhex", 50000);

ydva(1:4000)= !1;
ydva(14000:50000)= !1;
printf("Data valid count in modified X data set: %d\n",rows(find(!ydva)));
write_CCtest_data("Y_CCinput0_DS10.vhex", yamp, ypha, ydva);
```

3.25.2 DS11

```
# generate data set DS11 for bit by bit comparison with 100K samples (23 bit accumulator)
LOADPATH= ":/stage/rigel1/smegal/EVLA_CorrChip/testing/test_vectors";
[xamp, xpha, xdva, xnsf, xndva]= get_CCtest_data("../DS4/X_CCinput0.vhex", 150000);

xdva(1:4000)= !1;
xdva(54000:150000)= !1;
printf("Data valid count in modified X data set: %d\n",rows(find(!xdva)));
write_CCtest_data("X_CCinput0_DS11.vhex", xamp, xpha, xdva);

[yamp, ypha, ydva, ynsf, yndva]= get_CCtest_data("../DS4/Y_CCinput0.vhex", 150000);

ydva(1:4000)= !1;
ydva(54000:150000)= !1;
printf("Data valid count in modified X data set: %d\n",rows(find(!ydva)));
write_CCtest_data("Y_CCinput0_DS11.vhex", yamp, ypha, ydva);
```

3.25.3 DS14

```
# Generate data set DS14 for bit by bit comparison with 10K samples.
# Final accumulation starts approx. 1000 samples after data start.
#
# Allow for 1000+1024 samples before valid data.
# Allow at least 1024 sample to follow after data valid line is reset.
#
# April 4, 2005

[xamp, xpha, xdva, xnsf, xndva]= get_CCtest_data("../DS4/X_CCinput0.vhex", 50000);

xdva(1:3000)= !1;
xdva(8800:50000)= !1;
printf("Data valid count in modified X data set: %d\n",rows(find(xdva)));
write_CCtest_data("X_CCinput0_DS14.vhex", xamp, xpha, xdva);

[yamp, ypha, ydva, ynsf, yndva]= get_CCtest_data("../DS4/Y_CCinput0.vhex", 50000);

ydva(1:3000)= !1;
ydva(8800:50000)= !1;
printf("Data valid count in modified X data set: %d\n",rows(find(ydva)));
write_CCtest_data("Y_CCinput0_DS14.vhex", yamp, ypha, ydva);
```

3.25.4 DS15

```
# Generate data set DS15 for ad hoc diagnostic experiments with test case 20.
#
# Oct. 25, 2005

xampo= zeros(48000,1);
xphao= zeros(48000,1);
xdvao= !zeros(48000,1);
yampo= zeros(48000,1);
yphao= zeros(48000,1);
ydvao= !zeros(48000,1);

[xamp, xpha, xdva, xnsf, xndva]= get_CCtest_data("../DS4/X_CCinput4.vhex", 50000);
[xamps, xphas, xdvas, xnsfs, xndvas]= get_CCtest_data("X_CCinput0save20.vhex", 30000);

## subsitute X inp. 4 data and phase; shift by one sample
## keep X inp. saved data valid
xampo(2:48000)= xamp(518:48516);
xphao(2:48000)= xpha(518:48516);
xdvao(1:29483)= xdvas(518:30000);  # set dvalid according to data saver files (up to 30000)

## continue past data saver vector with dvalid reset every 512 samples
for nn=29180:512:48000;xdvao(nn:nn+5)= !1; endfor;
for nn=29480:27:48000;xdvao(nn)= !1; endfor;

printf("Data valid count in modified X data set: %d\n",rows(find(xdvao)));
write_CCtest_data("X_CCinput4_DS15.vhex", xampo, xphao, xdvao);

[yamp, ypha, ydva, ynsf, yndva]= get_CCtest_data("../DS4/Y_CCinput0.vhex", 50000);
[yamps, yphas, ydvas, ynsfs, yndvas]= get_CCtest_data("Y_CCinput0save20.vhex", 30000);
yampo(2:48000)= yamp(518:48516);
yphao(2:48000)= ypha(518:48516);
ydvao(1:29483)= ydvas(518:30000);

## continue past data saver vector with dvalid reset every 512 samples
for nn=29180:512:48000;ydvao(nn:nn+5)= !1; endfor;
for nn=29480:27:48000;ydvao(nn)= !1; endfor;

printf("Data valid count in modified Y data set: %d\n",rows(find(ydvao)));
write_CCtest_data("Y_CCinput0_DS15.vhex", yampo, yphao, ydvao);
```

3.25.5 DS16

```
# Generate data set DS16 for bit by bit comparison with 10K samples.
# Use DS14 as a template; change the input stream to 1.
# December 8, 2005

[xamp, xpha, xdva, xnsf, xndva]= get_CCtest_data("../DS4/X_CCinput1.vhex", 50000);

xdva(1:3000)= !1;
xdva(8800:50000)= !1;
printf("Data valid count in modified X data set: %d\n",rows(find(xdva)));
write_CCtest_data("X_CCinput1_DS16.vhex", xamp, xpha, xdva);

[yamp, ypha, ydva, ynsf, yndva]= get_CCtest_data("../DS4/Y_CCinput1.vhex", 50000);

ydva(1:3000)= !1;
ydva(8800:50000)= !1;
printf("Data valid count in modified X data set: %d\n",rows(find(ydva)));
write_CCtest_data("Y_CCinput1_DS16.vhex", yamp, ypha, ydva);
```

3.25.6 get_CCtest_data

```
function [amp, pha, dva, nsf, ndva]= get_CCtest_data(filespec, ns)
#
# usage: [amp, pha, dva, nsf, ndva]= get_CCtest_data(filespec, ns)
#

## version: 1.00

df1=fopen(filespec,"r");

if nargin < 2
               r1=fscanf(df1,'%s');
               fclose(df1);
               nsf= columns(r1)/3;        # total number of samples available in file
else
               ns= fix(ns);
               r1=fscanf(df1,'%s', ns);
               nsf= ns;
               fclose(df1);
endif

ramp=r1(1:3:(3*nsf));
rpha=r1(2:3:(3*nsf));
rdva=r1(3:3:(3*nsf));
r1= 0;

amp= zeros(nsf,1);
pha= zeros(nsf,1)';
dva= zeros(nsf,1) == 1;

amp= hex2dec(ramp');
itc= find(amp >= 8);
amp(itc)= amp(itc)-16;   # convert to signed number

pha= hex2dec(rpha');
dva= rdva' == "1";
ndva= rows(find(dva));
endfunction
```

3.25.7 write_CCtest_data

```
function write_CCtest_data(filespec, amp, pha, dva)
#
# usage: write_CCtest_data(filespec, amp, pha, dva)
#

nd= length(amp);
if (length(pha) != nd) && length(dva != nd)
               sprintf("Input data vectors are not the same length. Returning.\n");
               return
endif

# output in hex
df1=fopen(filespec,"w");
# convert the integers to positive
itc= find(amp < 0);
amp(itc)= amp(itc)+16;

for i=1:nd
               fprintf(df1,"%x%x%x\n",amp(i),pha(i),dva(i));
endfor
fclose(df1);
```

3.26 behavioural model

The source code for the behavioural model used in the post simulation analysis of this document is provided in the subsections below. This code was provided by B. Carlson of the National Research Council of Canada.

3.26.1 corrsimCCtest.c

```
/* Correlator simulator program to provide a reference correlation for testing
 * testvectors generated by noisegenCCtest that go to into the correlator chip
 * testbench.  The results of this test and the correlator chip test will not
 * be bit-by-bit comparable, but they will yield the same statistical result
 * with the same spectral signature.
 */
/* $Log: $ */

#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/stat.h>
#include <unistd.h>
#include "simparms_CCtest.h"


#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif

#ifndef M_PI
#define M_PI 3.14159265358979323846264
#endif


/* X and Y cross-corr delay lines...only the LSN of each one contains actual
 * data read from the .vhex file. */
typedef struct
{
  int    data;
  int    phase;
  int    dvalid;
} CROSSLAG_T;

typedef struct
{
  double in_phase;
  double quad;
} COMPLEX_T;


static CROSSLAG_T* X_lag_line=NULL;
static CROSSLAG_T* Y_lag_line=NULL;


/* insertion index for the X and Y delay lines */
static int crosslag_insert_index = 0;

static COMPLEX_T* crossaccum=NULL;

static int crosssample_count = 0;
static int crossinsert_count=0;


/* 3-level fringe stopping functions */
static int sin16[] = { 0, 1, 1, 1, 1, 1, 1, 0, 0, -1, -1, -1, -1, -1, -1,0 };
static int cos16[] = { 1, 1, 1, 0, 0, -1, -1, -1, -1, -1, -1, 0, 0, 1, 1,1 };

/* 5-level fringe stopping functions. */
static int sin16_5[] ={ 0, 1, 1, 2, 2, 2, 1, 1, 0, -1, -1, -2, -2, -2, -1,-1 };
static int cos16_5[] ={ 2, 2, 1, 1, 0, -1, -1, -2, -2, -2, -1, -1, 0, 1, 1,2 };

static int num_lags;
static int crosscorr_dellen;

static COMPLEX_T* localaccum=NULL;

static int testnum;
static int numdumps;
static int sample7bit=0;
```

```
/* FUNCTION PROTOTYPES */

static
void
crosscorrelate
( int xdata,
  int xphase,
  int xdvalid,
  int ydata,
  int yphase,
  int ydvalid );

static
void
output_crosscorr_data
(int testnumX,
 int testnumY,
 int dumpnum);



void
main
( int argc,
  char* argv[] )
{
  long samplenum;
  int corrsamples; /* number of samples to correlate per dump */
  int dumpnum;
  int i;
  FILE* fpX;
  FILE* fpY;
  char fnameX[256];
  char fnameY[256];
  int testnumX;
  int testnumY;
  int itemp;
  int xdata, xphase, xdvalid;
  int ydata, yphase, ydvalid;
  int xdata_m;
  int ydata_m;

  if( argc != 7 && argc !=8 )
    {
      fprintf(stderr,
   "Use: corrsimCCtest <#lags> <X-in#> <Y-in#> <#samples;0=all> <test#> <#dumps;0=all> (-7[7-bit])\n");
      exit(-1);
    }
  itemp = sscanf(argv[1],"%d",&num_lags);
  if( itemp != 1 )
    {
      fprintf(stderr,"***Error: could not determine #lags: '%s'\n",
              argv[1]);
      exit(-1);
    }
  if( num_lags % 2 )
    {
      fprintf(stderr,"***Error: #lags %d must be even!\n",num_lags);
      exit(-1);
    }
  crosscorr_dellen = ((num_lags/2)+1);

  /* dynamically allocate memory for lag correlator structures */
  X_lag_line = (CROSSLAG_T*)malloc(sizeof(CROSSLAG_T)*crosscorr_dellen);
  Y_lag_line = (CROSSLAG_T*)malloc(sizeof(CROSSLAG_T)*crosscorr_dellen);

  crossaccum = (COMPLEX_T*)malloc(sizeof(COMPLEX_T)*num_lags);
  localaccum = (COMPLEX_T*)malloc(sizeof(COMPLEX_T)*num_lags);

  if( X_lag_line==NULL || Y_lag_line==NULL || crossaccum==NULL || localaccum==NULL )
    {
      fprintf(stderr,"***Error: internal malloc() error!\n");
      exit(-1);
    }

  itemp = sscanf(argv[2],"%d",&testnumX);
  if( itemp != 1 )
    {
      fprintf(stderr,"***Error: could not determine X-input#: '%s'\n",
              argv[2]);
      exit(-1);
    }
  itemp = sscanf(argv[3],"%d",&testnumY);
  if( itemp != 1 )
    {
      fprintf(stderr,"***Error: could not determine Y-input#: '%s'\n",
              argv[3]);
```

```
    exit(-1);
  }
itemp = sscanf(argv[4],"%d",&corrsamples);
if( itemp != 1 )
  {
    fprintf(stderr,"***Error: could not determine #samples: '%s'\n",
            argv[4]);
    exit(-1);
  }
itemp = sscanf(argv[5],"%d",&testnum);
if( itemp != 1 )
  {
    fprintf(stderr,"***Error: could not determine test#: '%s'\n",
            argv[5]);
    exit(-1);
  }
itemp = sscanf(argv[6],"%d",&numdumps);
if( itemp != 1 )
  {
    fprintf(stderr,"***Error: could not determine #dumps: '%s'\n",
            argv[6]);
    exit(-1);
  }
if( argc==8 )
  {
    /* look for the -7 switch */
    if( !(strcmp(argv[7],"-7")) )
            {
              sample7bit=1;
              fprintf(stderr,"\nNote: 7-bit correlation\n");
            }
    else
            {
              fprintf(stderr,"***Error: unknown switch: '%s'\n",argv[7]);
              exit(-1);
            }
  }

/* open our input test files */
if( sample7bit )
  {
    sprintf(fnameX,"X_CCinput%d_7.vhex",testnumX);
    sprintf(fnameY,"Y_CCinput%d_7.vhex",testnumY);
  }
else
  {
    sprintf(fnameX,"X_CCinput%d.vhex",testnumX);
    sprintf(fnameY,"Y_CCinput%d.vhex",testnumY);
  }
fpX = fopen(fnameX,"r");
fpY = fopen(fnameY,"r");
if( fpX==NULL || fpY==NULL )
  {
    fprintf(stderr,
                "***Error: can't open '%s' or '%s' for reading\n",
                fnameX,fnameY);
    exit(-1);
  }

samplenum = 0;
dumpnum = 0;


/* ******************** Main sample loop ************************** */
while( fscanf(fpX,"%1x%1x%1x\n",&xdata,&xphase,&xdvalid) != EOF  &&
             fscanf(fpY,"%1x%1x%1x\n",&ydata,&yphase,&ydvalid) != EOF )
  {
    /* printf("samplenum: %d xdata: %d ydata: %d\n",samplenum,xdata,ydata); */
    if( sample7bit )
            {
              if( fscanf(fpX,"%1x%1x%1x\n",&xdata_m,&xphase,&xdvalid) != EOF &&
                  fscanf(fpY,"%1x%1x%1x\n",&ydata_m,&yphase,&ydvalid) != EOF )
                {
                  /* xdata contains the LSN, xdata_m contains the MSN...merge the two */
                  xdata = (xdata&0x0f) | ((xdata_m&0x07)<<4);
                  /* correct for sign extension loss */
                  if( xdata > 63 )
                          xdata = xdata-128;

                  /* same for ydata */
                  ydata = (ydata&0x0f) | ((ydata_m&0x07)<<4);
                  /* correct for sign extension loss */
                  if( ydata > 63 )
                          ydata = ydata-128;

                  /* don't worry about encoding data valid here for the actual test
                   * with ModelSim...it will be done in the test bench */
```

```
                }
              else
                {
                  break; /* all done */
                }
            }
      else
            {
              /* correct for sign-extension loss */
              if( xdata > 7 )
                xdata = xdata-16;
              if( ydata > 7 )
                ydata = ydata-16;
            }

      crosscorrelate(xdata,xphase,xdvalid,ydata,yphase,ydvalid);

      if( corrsamples > 0 )
            {
              /* performing multiple dumps... */
              if( !(samplenum % corrsamples) && samplenum > 10 )
                {
                  /* gotta dump the data */
                  output_crosscorr_data(testnumX,testnumY,dumpnum);
                  crosssample_count=0;
                  dumpnum++;
                  if( (numdumps > 0) && (dumpnum >= numdumps) )
                          exit(-1);
                }
            }
      samplenum++;
    } /* end while loop */
  output_crosscorr_data(testnumX,testnumY,dumpnum);
}


/* function to do a simple 'bi-directional' cross-correlation.  Note that the
 * center lag is at num_lags/2 */

static
void
crosscorrelate
( int xdata,
  int xphase,
  int xdvalid,
  int ydata,
  int yphase,
  int ydvalid )
{
  int lag;
  int i;
  int indexX;
  int indexY;
  double sin_part,cos_part;
  int phase;

  if( crossinsert_count == 0 )
    {
      for( i=0; i<crosscorr_dellen; i++ )
        {
          X_lag_line[i].data = 0;
          Y_lag_line[i].data = 0;
          X_lag_line[i].phase = 0;
          Y_lag_line[i].phase = 0;
                X_lag_line[i].dvalid = 0;
                Y_lag_line[i].dvalid = 0;
        }
      for( i=0; i<num_lags; i++ )
        {
          crossaccum[i].in_phase = crossaccum[i].quad = 0.0;
        }
    }

  /* insert data, phase, data valids into the delay line */
  X_lag_line[crosslag_insert_index].data = xdata;
  X_lag_line[crosslag_insert_index].phase = xphase;
  X_lag_line[crosslag_insert_index].dvalid = xdvalid;

  Y_lag_line[crosslag_insert_index].data = ydata;
  Y_lag_line[crosslag_insert_index].phase = yphase;
  Y_lag_line[crosslag_insert_index].dvalid = ydvalid;


  /* do not increment the insertion index yet! so that it points at the newest
   * sample. */

  crossinsert_count++;
```

```
  for( lag=0; lag<num_lags; lag++ )
    {

      if( !(lag % 2) )
        {
          /* even lags */
          indexX = (crosslag_insert_index + 1 + lag/2) % crosscorr_dellen;
          indexY = (crosslag_insert_index - lag/2);
          if( indexY < 0 )
            indexY += crosscorr_dellen;
        }
      else
        {
          /* odd lags */
          indexX = (crosslag_insert_index + 1 + (lag+1)/2) % crosscorr_dellen;
          indexY = (crosslag_insert_index - ( lag-1)/2);
          if( indexY < 0 )
            indexY += crosscorr_dellen;
        }

      phase = (X_lag_line[indexX].phase-Y_lag_line[indexY].phase);
      if( phase < 0 )
        phase += 16;
      else
        phase = phase % 16;

#if LEVEL5_FRINGE_STOPPING
      sin_part = (double)sin16_5[phase];
      cos_part = (double)cos16_5[phase];
#else
      sin_part = (double)sin16[phase];
      cos_part = (double)cos16[phase];
#endif


      crossaccum[lag].in_phase +=
        (double)(X_lag_line[indexX].data * X_lag_line[indexX].dvalid *
                 Y_lag_line[indexY].data * Y_lag_line[indexY].dvalid) * cos_part;

      crossaccum[lag].quad +=
        (double)(X_lag_line[indexX].data * X_lag_line[indexX].dvalid *
                 Y_lag_line[indexY].data * Y_lag_line[indexY].dvalid) * sin_part;


      /* maintain a data valid count at the center lag */
      if( lag == num_lags/2 )
                {
                  if( X_lag_line[indexX].dvalid & Y_lag_line[indexY].dvalid )
                    crosssample_count++;
                }
    }

  /* finally, increment the insertion index */
  crosslag_insert_index = (crosslag_insert_index+1) % crosscorr_dellen;
}


/* this functions spits out cross-correlated data to a file */
static
void
output_crosscorr_data
(int testnumX,
 int testnumY,
 int dumpnum )
{
  FILE* fp;
  int lag;
  double max;
  char buf[256];
  char buf1[256];
  int i;

  if( crosssample_count <= 0 )
     return; /* nothing to do */

  /* normalize and copy data locally */
  for( lag=0; lag<num_lags; lag++ )
    {
      localaccum[lag].in_phase = crossaccum[lag].in_phase /
        (double)crosssample_count;
      localaccum[lag].quad = crossaccum[lag].quad /
        (double)crosssample_count;

      /* clear the active accumulators as well...since we could be doing
      * multiple dumps */
      crossaccum[lag].in_phase = 0.0;
```

```
      crossaccum[lag].quad = 0.0;
    }

  printf("Data valid count=%d\n",crosssample_count);

  /* The only normalization that is done is to divide by the data valid
   * count.  The Van Vleck normalization is not done, and is pointless
   * anyway for this test.
   */
  sprintf(buf,"CC%dx%d_test%02d_%dlagsD%d.dat",
                testnumX,testnumY,testnum,num_lags,dumpnum);

  fp = fopen(buf,"w");
  if( fp == NULL )
    {
      fprintf(stderr,"***Error: can't open file '%s'\n",
                     buf);
      exit(-1);
    }

  for( lag=0; lag<num_lags; lag++ )
    {
      fprintf(fp,"%.15e %.15e\n",
              localaccum[lag].in_phase,
              localaccum[lag].quad);
    }
  fclose(fp);
}
```

3.26.2 sigparms_CCtest.h

```
#ifndef SIMPARMS_CCTEST_H
#define SIMPARMS_CCTEST_H

/*
Description:  This file contains common defines for the noisegenCCtest.c and the
corrsimCCtest.c simulators that are used for generating testvectors for
correlator chip simulation and for correlating those testvectors for correlator
chip behaviour comparisons.
*/

/* $Log: $ */

#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <sys/resource.h>

#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif

/* ************************************************************************ */
/* ************************** TEST VARIABLES ********************************** */
/* ************************************************************************ */


/* ************************** Quantizer defines ********************************** */
#define QUANT4THR       0.374        /* 4-bit quantizer threshold differential
                                                              * in terms of sigma of the input signal
                                                              * (0.374) */
/* ************************************************************************ */


/* ********************** Noise Generator defines ********************** */
#define ADD_SYS_NOISE              TRUE /* TRUE if independent system noise is
                                        * added to each station before
                                        * quantization, filtering, and
                                        * correlation */
#define SYS_NOISE_SIGMA            5.0  /* system noise amplitude in sigmas
                                         * relative to source noise (4.7)*/
#define SRC_NOISE_SIGMA            5.0   /* source noise amplitude */
/* ************************************************************************ */


/* ********************** Frequency-shift specific defines *************** */
#define FREQUENCY_SWITCH           TRUE /* Set TRUE for frequency shift
                                        * generation and fringe-stopping in
                                        * the correlator.  Set FALSE for no
                                                                * frequency switching. */

/* station frequency shifts in cycles/sample */
```

```
#if TRUE /* The normal fringe rate */
#define XFREQ_CPS                 0.00012821
#define YFREQ_CPS                 0.00060823
#endif

#if FALSE /* wind fringes 10X faster */
#define XFREQ_CPS                 0.0012821
#define YFREQ_CPS                 0.0060823
#endif


#define LEVEL3_FRINGE_STOPPING     TRUE   /* set TRUE for 3-level fringe
                                           * stopping.  If FALSE, then floatin
                                           * point fringe stopping. */
#define LEVEL5_FRINGE_STOPPING     FALSE  /* use 5-level fringe
stopping...this
                                                                            * overrides the 3-level switch */
#define HILBERT_TAP_LEN            165  /* num tap coeffs in the SSB mix H(n) */

/* ************************************************************************ */


/* ************************* Interference Lines specs ******************** */
#define INTERFERENCE_LINE_SWITCH   TRUE   /* turn on generation of correlated
                                           * narrowband "interference" lines
                                           * */
#define MAX_NUM_INTERF_LINES      0      /* number of interference lines... the
                                                                            * line amplitudes and frequencies are
                                                                            * set in the noisegenCCtest.c file.
                                                                            * If this is zero, then the number
                                                                            * of lines is set by the testnumber */
/* ************************************************************************ */

#endif /* #ifndef SIMPARMS_CCTEST_H */
```

Bibliography

TVP-02: Smegal, Rick, EVLA Correlator Chip Functional Test Cases, Test and Verification Plan, A25082N0002, Rev. 1.0, National Research Council of Canada, January 30, 2006

TVP-01: Carlson, Brent, EVLA Correlator Chip, Test and Verification Plan, A25082N0001, Rev. DRAFT, National Research Council of Canada, October 26, 2004

RFS-00: Carlson, Brent, EVLA Correlator Chip, Requirements and Functional Specification, A25082N0000, Rev. 2.3, National Research Council of Canada, October 29, 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