REQUIREMENTS AND FUNCTIONAL SPECIFICATION

EVLA Correlator Chip Testing Software

Comparison of Functional Simulation Output with Post-Place-and-Route Simulation Output

RFS Document: **A25082N0010**

Revision: 1.2

Frances Lau, May 27, 2003

National Research Council Canada
Herzberg Institute of Astrophysics
Dominion Radio Astrophysical Observatory

P.O. Box 248, 717 White Lake Rd
Penticton, B.C., Canada
V2A 6K3

Table of Contents

List of Figures

1 Revision History

Revision	Date	Changes/Notes	Author
1.0	May 8, 2003	Initial Revision	Frances Lau
1.1	May 14, 2003	Revision	Frances Lau
1.2	May 27, 2003	Revision to match actual design	Frances Lau

2 Introduction

This document describes the requirements and design concepts for a piece of testing software that will compare the correlator chip functional simulation output with the post-place-and-route (post-PAR) simulation output. It will analyze

a) the data frames [1] transmitted to the Long Term Accumulator (LTA) Controller and

b) the contents of the Monitor and Control Bus (MCB) registers.

The software will be written in Ansi-C. It will be portable, capable of being run on UNIX machines.

3 Context

This piece of software will be a fundamental part of the testing and verification aspect of the correlator chip design process. The software will compare output produced by the functional simulator with output produced post-place-and-route. Testing using this software will increase confidence that the placed and routed chip is functionally performing as required.

4 Overview

The data that will be compared are the data frames transmitted to the Long Term Accumulator (LTA) Controller and the contents of the Monitor & Control Bus (MCB) registers.

Data is transmitted to the LTA Controller in data frames, where each frame is the control, status, and data for one Correlator Chip Cell (CCC). An independent frame is transmitted for each CCC. Each data frame contains 267 Words (Word0 to Word266), as defined in Figure 5-3 of the EVLA Correlator Chip Requirements and Functional Specification document [1].

The data frames transmitted to the LTA Controller will be compared by analyzing lag frame files. These lag frame files are ASCII representations of the correlator chip data frames produced by the simulator. They contain eight digit hexadecimal numbers that represent each word in the data frame, as well as some comments that decode the hexadecimal numbers. The comments will be ignored in the comparison.

The contents of MCB registers will be compared by analyzing ASCII files containing representations of the bits in each register. The comments that decode the register contents are ignored in the comparison.

A block diagram showing an overview of the flow of data is in Figure 4-1.

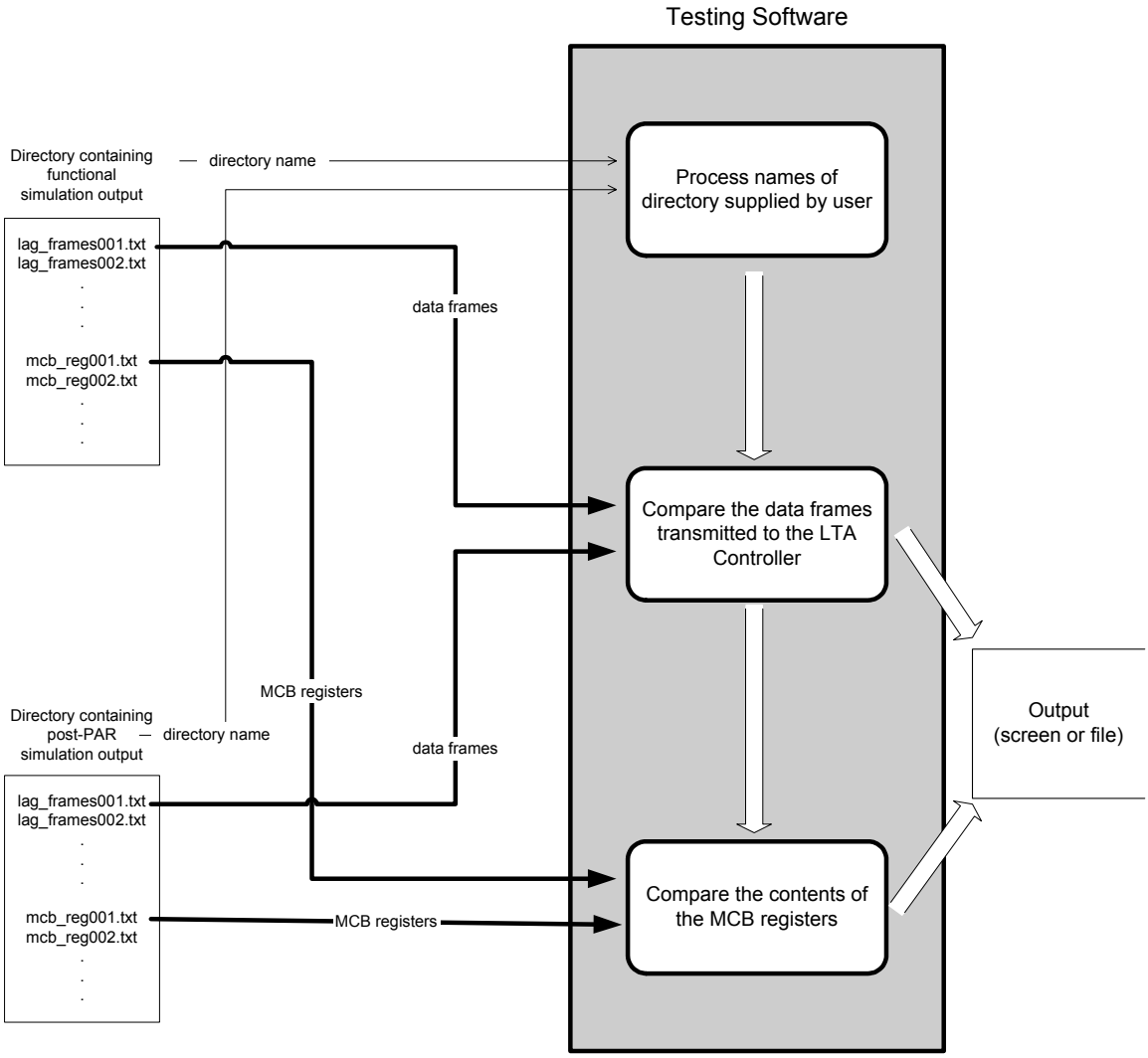

Figure 4-1: Block Diagram of Flow of Data in Testing Software

5 Requirements

The following is a list of the requirements for this piece of testing software.

5.1 Functional Requirements

A) LTA Controller Data Frames

1. The program will compare the data frames from the functional simulation with the data frames from the post-place-and-route simulation. For each functional simulation data frame, it will try to find the matching post-place-and-route simulation data frame. If an exact match is not found, the frame that matches most closely will be flagged and the discrepancy will be indicated.

2. The words that are most critical to a match are:
   - Word 1: the Correlator Chip Cell Number (CCC#)
   - Word 4 and Word 5: the time stamps

   The program will compare Word0 to Word5 first because these contain the CCC# and timestamps. The rest of the words are compared *only* when a match in the CCC# and timestamps is found.

3. This process will repeat for each file in the directory.

4. The data frames in the two files may be in a different order. The program should be able to find the matching data frame even if it is not in order.

5. When one matching data frame is found, the program will continue to check if there are any additional matching data frames.

B) MCB Registers

1. The MCB register files will be compared. Only the contents of the registers, not the comments, will be analyzed.

5.2 Performance Requirements

1. The processing time and complexity of the program must be reasonable.

5.3 Environmental Requirements

1. The program will be run in a UNIX environment.

5.4 Interface Requirements

1. The program will have a UNIX command line user interface.

2. To compile the program, the user can either type:

   ```
   gcc -Wall -ansi -pedantic -o comparison comparison.c
   ```

   or type `make`, and the makefile will automatically execute the above command. Note that the `-Wall -ansi -pedantic` options are not mandatory, but they are useful for debugging.

3. To execute the program, the user will enter the command `comparison`, followed by the directory names at the command prompt. The syntax is below:

   ```
   comparison [funcsim] [pparsim] > [outputfile.txt]
   ```

   `funcsim` is the name of the directory for the functional simulation files
   `pparsime` is the name of the directory for the post-place-and-route simulation files
   `outputfile.txt` is the name of the file where the results will be output. This parameter is optional. If it is omitted, the results will be output on the screen.

   If the user does not specify a directory name for the functional simulation and post-place-and-route simulation files, the default directory names `funcsim` and `pparsim` will be used.

4. The names of the files in the two directories must be identical. If they are not or if a file is missing, an error message will be displayed and the user must correct this before proceeding. Lag frame files must have the filename format `lag_frames*.txt`. MCB register files must have the filename format `mcb_reg*.txt`.

5. The user will have the option of having the results output on the screen or in a file. If the user wants to have the output in a file, he/she must supply a filename.

6. If an exact match is found for a data frame, the program will output a message indicating that a match was found for this CCC# and timestamp. If an exact match is not found, the program will display an error message indicating the word where a discrepancy was found. If no data frame is found with a matching data frame and timestamp, a message will be displayed indicating this.

7. A summary of the errors and warnings found will be displayed. This includes statistics summarizing the number of frames in each file.

6 Functional Specifications

The program will compare the functional simulation output and the post-place-and-route output from two sources:

a) the data frames transmitted to the LTA Controller, and
b) the contents of the MCB registers.

A similar method will be used for both comparisons. The flowchart in Figure 6-1 illustrates the overall program structure. A description of the functions that perform general tasks is below.

General Functions

main

- Calls getDirNames to process the names of the directories where the files are found.
- Calls compareLag to compare all the lag frame files in the directory.
- Calls compareMCB to compare all the MCB register files in the directory.

getDirNames

- Processes the directory and file names supplied by the user at the command line.
- If the user omits a parameter, the program will revert to a default setting. The directory names `funcsim` and `pparsim` will be used and the output will be printed to the screen.

findFile

- After the contents of the directory is output into a file, findFile reads the file in order to obtain the next filename.

outputFinalSumm

- Displays the total number of errors and total number of warnings for this test case

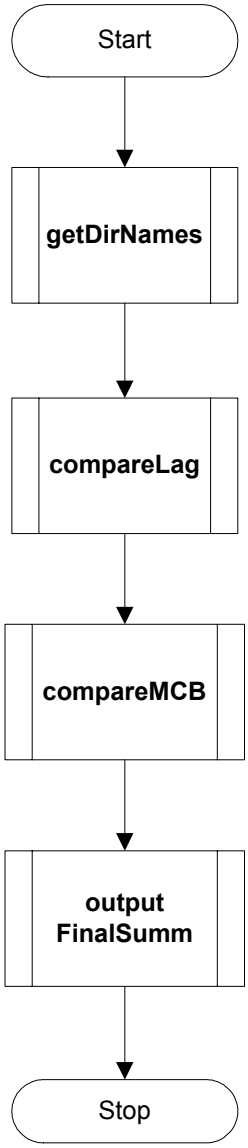

Figure 6-1: Flowchart of overall program structure

A) Functions for the Comparison of LTA Controller Interface Data Frames

Figure 6-2 illustrates compareLag, Figure 6-3 illustrates compareLagFile, and Figure 6-4 illustrates findMatch, the three main functions. A description of all the functions is below.

compareLag

- Cycles through all the lag frame *files* in the *directory*.
- The names of the lag frame files in the directory are stored in a temporary file so that they can be read.
- Calls compareLagFile to process each file.

compareLagFile

- Cycles through all the *frames* in the *functional simulation file*.
- Calls the findMatch function to find matches for each frame.
- Displays a summary of the results for this file.

findMatch

- Finds matches for each frame it is passed by cycling through all the *frames* in the *post-place-and-route simulation file*.
- Compares Word 0 to Word 5. If a frame with a matching CCC# and timestamp is found, the function compareRemainderOfFrame is called (Note: Word 0 to Word 5 are compared because these words contain the CCC# and the timestamps). The program will continue searching the frames even when a match is found in order to check if there is more than one matching data frame.

compareRemainderOfFrame

- Executed when we have found a frame that has matching CCC#, Timestamp-0, and Timestamp-1.
- Compares Word 6 to Word 266 of these 2 frames only

toNextFrame

- Goes to the next frame in the file.
- The next frame is indicated by:
    a) A blank line or a comment before Word0, and
    b) The hexadecimal number beginning with 'a', since Word0 is always aaaaaaaa.

readHex

- Reads the hexadecimal number, ignoring the comments at the end of each line that decodes the hexadecimal number.

extractCCC

- Decodes Word1 to extract the CCC#.

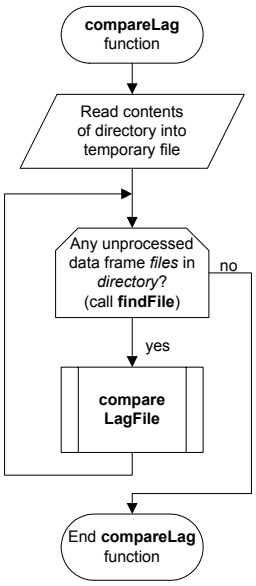

Figure 6-2: Flowchart of compareLag function

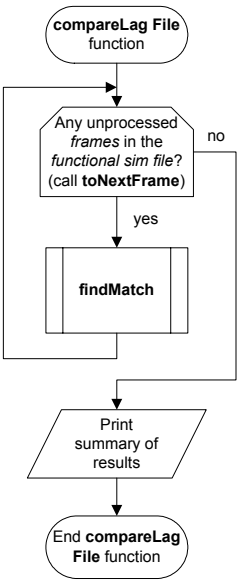

Figure 6-3: Flowchart of compareLagFile function

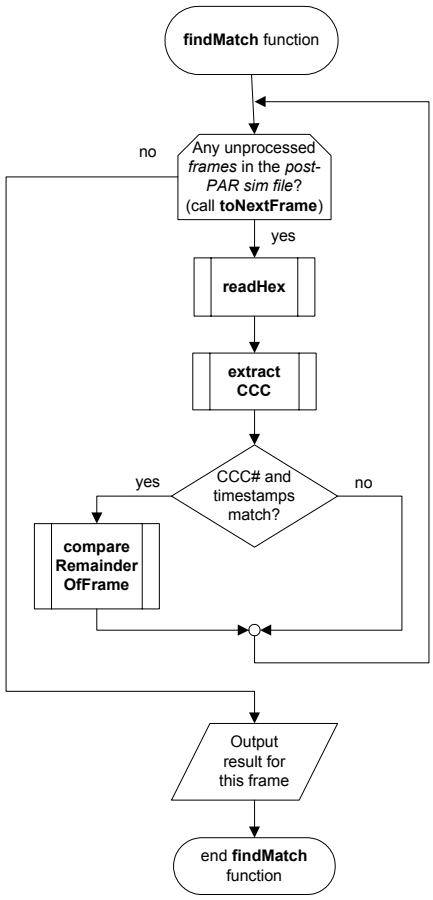

Figure 6-4: Flowchart of the findMatch function

B) Functions for the Comparison of MCB Register Files

The MCB register files will be compared using a similar method. Figure 6-5 illustrates the function compareMCB and Figure 6-6 illustrates the function compareMCBRegFile, the two main functions. A description of all the functions is below.

compareMCB

- Cycles through all the MCB Register *files* in the *directory*.
- Calls the compareMCBRegFile function to process each file.

compareMCBRegFile

- Cycles through each *register* of the MCB register *file.*
- Compares the contents of each register.
- Calls toNextMCBReg to go to the next line of the MCB register file, ignoring the comments.

toNextMCBReg

- Goes to the next register in the file.
- The next register is indicated by:
    a) The new line character '\n' as the last character in the previous line, and
    b) '1' or a '0' as the first character of the current line.

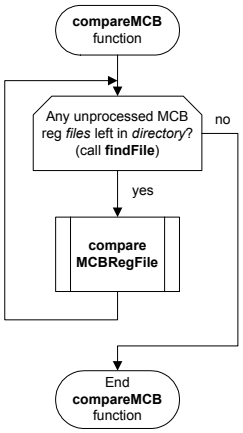

Figure 6-5: Flowchart of compareMCB function

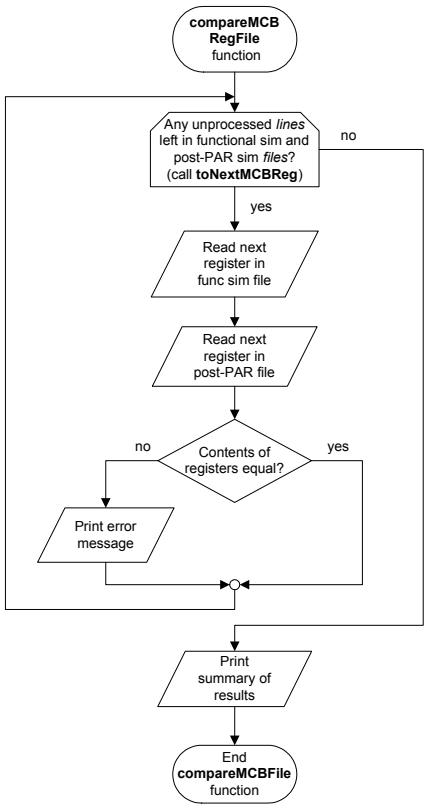

Figure 6-6: Flowchart of compareMCBRegFile function

7 References

[1] Carlson, Brent, Requirements and Functional Specification: EVLA Correlator Chip, July 26, 2002.