Chapter 5 . WAVES

5.1 Introduction

WAVES (Waveform and Vector Exchange Specification) [9] is the industry standard representation for specifying digital stimulus and response for both the design and test communities. When testing a model we require input stimuli as well as the expected responses. WAVES provides an environment for specifying these test waveforms.

WAVES is an application of VHDL (IEEE Standard 1076-87) [10] and uses only sequential statements. As stated in section 2.4, the WAVES specifications defines two “levels” of datasets. These levels are called WAVES level 1 and 2. Level 1 is simple and very restrictive and can be used for the majority of waveforms. All language structures in WAVES Level 1 are included in WAVES level 2 which is less restrictive and is similar to a programming language. Both Level 1 and 2 are in turn subsets of VHDL.

The WAVES data sets interact with a test bench through the use of one or two signals. The first is a WAVES port list (***WPL***). This is an array containing N elements where N is the number of inputs and outputs of the MUT. The list of pins generated for the Sobel edge detector is shown in Figure 5.7. Each of the elements either contain a value to apply or sense from the MUT. The *WPL* also indicates the type of delay to be applied which is, either a ***timed*** or a ***handshake*** delay, both of which are explained in detail in the subsequent pages. The second signal is the WAVES match list. This contains the results of all the match events.

A WAVES waveform represents stimulus/response information for the MUT. It is produced by the ***Waveform Generator procedure*** (***WGP***) which is a subprogram or collection of subprograms which reads in the external files, interprets their contents and constructs the waveforms using the ***Apply, Match and Tag*** subprograms/functions described later. The only parameters allowed for the *WGP* are the *WAVES port list* and the *WAVES match list*. The waveform generator procedure uses the external files as a source of data. A given waveform, can be viewed as a sequence of time-ordered events across a set of signals.

A waveform in WAVES is usually organized in terms of ***slices***. A ***slice*** is a specification of part of the waveform for some period of time across all signals. A waveform consists of a number of slices and the starting time for each slice. For example the waveform shown in Figure 5.1, could be represented as 3 slices in Figure 5.2(a) or as 6 slices in Figure 5.2(b). A slice could be further divided into ***frames*** which consists of a list of all events in a slice for a single signal. A slice then can be specified as a list of frames, such that each signal in the test bench has an associated frame. In the slices shown in Figure 5.2(a), each frame has some number of events, whereas in the slices shown in Figure 5.2(b) each frame has only one event. In depth information on slices and frames can be found in [9].

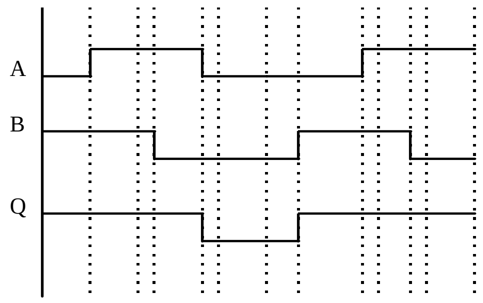

Figure 5.1 Sample Waveform

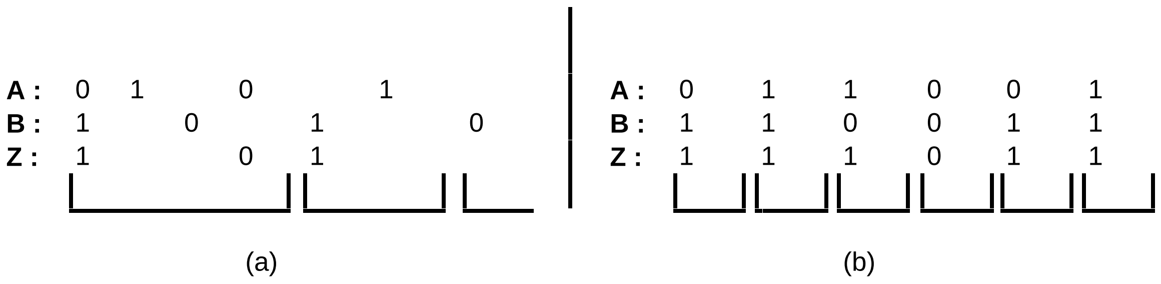

Figure 5.2 Two representations of the above waveform

A WAVES dataset is a collection of data required to completely specify one or more waveforms. It typically consists of the following files :

- ***Header file*** : The header file specifies the file names of the waves files and the reference waves unit in a legal order for WAVES analysis. The order of analysis for waves files and standard waves units is the same as that of equivalent VHDL design units. In addition, the header file may also refer to the WAVES libraries: WAVES_Standard, WAVES_system, WAVES_Interface and WAVES_Objects. The WAVES_system library is not visible to the user. The other three libraries are available to the user.

- ***Waves file*** : The waves files contain the WAVES specifications for the test bench.

- ***External files*** : These files contain information on a specific test case. These have to be in a format required by the waves files to produce the WAVES waveform.

WAVES provides VHDL procedures and types that are used to build events, frames and slices. These procedures and types are specified in the WAVES_Standard package, some of which are described below :

- ***Logic values*** : The type *Logic_Value* enumerates the different logic types used in the test bench.

- ***Value dictionary*** : The Value dictionary defines the meaning of the logical values by stating their state, strength and relevance.

- ***Pin Codes*** : A one-character code is associated with each frame type. These are listed in the constant string *Pin_Codes*.

- ***Test Pins*** : The type *Test_Pins* lists all the input and output pins of the device under test. Given this, a slice can be specified as a list of pin codes, one code per test pin; each code refers to a frame to apply to the corresponding pin.

- ***Frame set*** : A frame is specified as a list of frame events. The predefined type *Frame_Set* is used to define sets of frames and bind the pin codes to the individual frames.

- ***Waveform Generation Procedure*** : As already defined, WGP is a procedure used to generate the waveforms.

Generation of waveform events is done through the use of WAVES subprograms. Each of these procedures and functions issues the WAVES events necessary to specify the expected response on the waveform. These are briefly described below :

- ***Apply*** : The *Apply* procedure takes a number of arguments representing a new frame and a list of relevant pins and schedules the events in its slice on the output waveform. After scheduling the events, the *Apply* operation causes time to advance. Time is advanced in one of two ways. The *Apply* operation may specify a ***timed delay*** or a ***handshake delay***. The first simply causes a fixed amount of time to advance. A *handshake delay* causes the *Apply* procedure to wait for a specified event value or values to occur on some pin. The *handshake* delay does not wait on values that are scheduled by the *Apply* operation itself but rather waits for responses from the MUT.

- ***Tag*** : The tag operation annotates the waveform with arbitrary text, without advancing time.

- ***Match*** : The match operation samples the actual response of the MUT, compares it to the predicted response and yields true if the actual was as predicted and false otherwise.

5.2 Example WAVES Test benches

The Sobel Edge detector was tested both at the structural level (with *std_logic* data type) and at the register transfer level with the WAVES test bench.

The testing procedure adopted for testing the Sobel edge detector at the structural level is now explained in detail. The MUT at the register transfer level was tested in a similar fashion. The testing procedure for developing the test bench module for the MUT is detailed below:

Step 1 : There are two files which are required by WAVES for creating the test bench module. These are detailed below :

- ***MUT file*** : This is the file containing the Sobel edge detector model. One important aspect which had to be considered is that all the input and output ports have to be strictly specified as *std_logic* data type since this is the only data type allowed by the present version of the WAVES tool. The user defined data types of ***PIXEL*** and ***FILTER_OUT*** specified as std_logic data type in the ***IMAGE_PROCESSING package*** could not be used to define the ports of the MUT. The entity declaration portion of the original MUT file is shown in Figure 5.3. The modified file is shown in Figure 5.4. Differences are highlighted in bold face type.

```
 entity EDGE_DETECTOR is
   port ( CLOCK: in STD_LOGIC;
          EDGE_START : in STD_LOGIC;
          INPUT: in PIXEL;
          THRESHOLD: in FILTER_OUT;
          OUTPUT: inout PIXEL;
          DIR: inout DIRECTION);
end EDGE_DETECTOR;
```

Figure 5.3 Portion of the original MUT file specifying the entity declaration for the MUT.

```
entity EDGE_DETECTOR is
   port ( CLOCK: in STD_LOGIC;
          EDGE_START : in STD_LOGIC;
          INPUT: in STD_LOGIC_VECTOR(7 downto 0);          --STD_LOGIC specified instead of PIXEL
          THRESHOLD: in STD_LOGIC_VECTOR(11 downto 0);--STD_LOGIC specified
                                                          --instead of FILTER_OUT
          OUTPUT: inout STD_LOGIC_VECTOR(7 downto 0);  --STD_LOGIC specified instead of PIXEL
          DIR: inout STD_LOGIC_VECTOR(2 downto 0));     --STD_LOGIC specified instead of DIRECTION
end EDGE_DETECTOR;
```

Figure 5.4 Portion of the modified MUT file specifying the entity declaration for the MUT.

- ***Waves External file*** : Since most of the image files are in the form of a two-dimensional array, a program was developed to convert the image files of integer type to a format suitable for the WAVES tool. This program can be found in Appendix C. A Portion of the test file used to test the Sobel edge detector with the WAVES test bench is shown in Figure 5.5. The number of test vectors specified on each line must be exactly equal to the number of pins specified in Figure 5.7. The order of the test vectors in each line must be the same as the order of pins in the TYPE *test_pins* in Figure 5.7. The first two values of '**1**' are for the *CLOCK* and *EDGE_START* signals. Next we specify an 8-bit and a 12-bit std_logic vector for the *INPUT* and *THRESHOLD* ports and then we specify an 8-bit and a 3-bit std_logic vector represented by *dashes* for the expected magnitude and direction outputs respectively. The last '1' after the colon sign is for the length of the time slice, whose actual value has been specified in the WGP file shown in Figure 5.9. Generally, the user needs to specify actual values instead of *dashes*, so that WAVES can directly compare the outputs of the MUT with the specified expected outputs and generate a PASS/FAIL indication. However, in the case of the Sobel edge detector, since the valid set of outputs are not continuous and are also not generated simultaneously with the input data, the expected responses are represented by dashes. A separate process is developed in the test bench to compare the outputs of the MUT with the expected response which is explained in detail in the section describing WAVES test bench files.

```
1 1 01100100 000000110010  -------- --- : 1;
1 1 01100100 000000110010  -------- --- : 1;
1 1 01100100 000000110010  -------- --- : 1;
1 1 01100100 000000110010  -------- --- : 1;
1 1 01100100 000000110010  -------- --- : 1;
1 1 00000000 000000110010  -------- --- : 1;
1 1 00000000 000000110010  -------- --- : 1;
1 1 00000000 000000110010  -------- --- : 1;
1 1 00000000 000000110010  -------- --- : 1;
```

Figure 5.5 External file developed for testing the MUT

Step 2: After the MUT file and the external file have been successfully developed, the MUT is analyzed, and then compiled in WAVES using the following command :

MB (name of the MUT file).

Once the MUT is compiled, WAVES toolset automatically generates a number of files, which are explained below.

- Header file : This file is created by WAVES and is called ((*name of the MUT)_Header.txt*). It is a text file and is not required during the simulation. It defines the order of analysis of the VHDL source code files comprising the WAVES test suite. Figure 5.6 shows the example of the Header file created for the MUT (Sobel edge detector). The first part of the header file contains brief text strings defining the title of the data set, name of the individual who developed the data set and the date when the data set was developed. All the above details have to be filled in by the user. After the identification section, the WAVES files are specified in the order in which they are to be analyzed. The Header file also includes the default external file name which is defined as " *vectors.txt*" which we changed to *"test.txt"*, the actual name of the external file used by us for simulation.

```
******** Header File for Entity: edge_detector
-- Data Set Identification Information
TITLE        A General Description
DEVICE_ID    edge_detector
DATE         Tue May 20 14:51:32 1997
ORIGIN       Virginia Tech
AUTHOR       Sucharita Gopalakrishnan
DATE         Tue May 20 14:51:32 1997
OTHER        Built Using the WAVES-VHDL 1164 STD Libraries
-- Data Set Construction Information
WAVES_FILENAME   edge_s_std_uut_pins.vhd              WORK
library      IEEE;
use          IEEE.WAVES_1164_Declarations.all;
use          IEEE.WAVES_Interface.all;
use          WORK.UUT_Test_Pins.all;
WAVES_UNIT       WAVES_Objects
WAVES_FILENAME   edge_s_std_dff.vhd                   WORK
--
EXTERNAL_FILENAME   test.txt                          VECTORS
WAVEFORM_GENERATOR_PROCEDURE     WORK.waves_edge_detector.waveform
```

Figure 5.6 Waves header file created for Sobel edge detector

- ***UUT_Test_pins file*** : Figure 5.7 shows the *UUT_Test_pins* file for the Sobel edge detector((*name of the MUT)_Test_pins.vhd*). It defines the input and output pins of the MUT. The pins are specified according to the input and output ports defined in the VHDL model for the MUT in the MUT file. The test vectors provided for each pin has to be in the same order as defined in the *UUT_Test_pins file*.

```
******** This File Was Automatically Generated  ********
-- ******** By The WAVES96-VHDL Tool Set      ********
-- ******** Generated for Entity: edge_detector
-- ******** This File Was Generated on: Tue May 20 14:51:25 1997
PACKAGE uut_test_pins IS
TYPE test_pins IS (clock, edge_start, input_7, input_6, input_5, input_4,
         input_3, input_2, input_1, input_0, output_7, output_6, output_5,
         output_4, output_3, output_2, output_1, output_0, dir_2, dir_1,
         dir_0);
END uut_test_pins;
```

Figure 5.7 UUT_Test_pins file for Sobel edge detector

- ***Waves_Objects_file*** : This file is provided by the WAVES tool.

- ***WGP package file*** : Portions of the source code of the WGP package generated for the Sobel Edge detector, are shown in Figures 5.8-5.11. The following have to be specified before analyzing the file.

1. First we need to declare the name of the external file to be used for testing the MUT. For our example we have used the sample file shown in Figure 5.5. WAVES declares a default name, “vectors.txt” which has to be changed to the actual name of the external file to be used. As seen from Figure 5.8, the name of the external file has been specified as “test.txt” which was the file used for simulating the test bench for the Sobel Edge detector.

```
package WGP_edge_detector is
          procedure WAVEFORM(signal WPL : inout WAVES_PORT_LIST);
end WGP_edge_detector;
--------------------------------------------------------
package body WGP_edge_detector is
           procedure WAVEFORM(signal WPL : inout WAVES_PORT_LIST) is
    -- Standard Waveform Generator
   -- Declare external file
   file VECTOR_FILE : text is in "test.txt";       ---   ( EXTERNAL FILE NAME SPECIFIED )
```

Figure 5.8 Portion of the source code of the WGP package where the external file is specified.

2. The WGP file generated by WAVES does not contain any timing values. These have to be filled in by the user according to the test and verification needs of our specific design. The timing values are shown in bold in the portion of the source code in Figure 5.9. According to the values specified, the clock and output waveforms which are generated by the procedure defined in the WGP package are shown in Figure 5.10.

```
variable WTL : WAVE_TIMING_LIST( 1 to 2 ) := (
 -- Frame Set 1
    ( PERIOD =>100 ns,
      FSA    => BUILD_FRAME_DATA(
                 ((+clock, PULSE_HIGH(25 ns,75 ns)),           ----
                   (+edge_start, NON_RETURN(0 ns )),           ----
                   (output, NON_RETURN(0 ns )),                ---- TIMING VALUES SPECIFIED
                   (dir, NON_RETURN(0 ns )),                   ----
                   (INPUTS, NON_RETURN(0 ns))                  ----
                 )) ),
    -- Frame Set 2
    ( PERIOD =>100 ns,
      FSA    => BUILD_FRAME_DATA(
                 ((+clock, PULSE_HIGH(25 ns,75 ns)),           ----
                   (+edge_start, NON_RETURN(0 ns )),           ----
                   (output, WINDOW(50 ns,100 ns )),            ---- TIMING VALUES SPECIFIED
                    (dir, WINDOW(50 ns,100 ns )),              ----
                   (INPUTS, NON_RETURN(0 ns))                  ----
                 ) )  ) );
```

Figure 5.9 Portion of the source code of the WGP package specifying timing values.

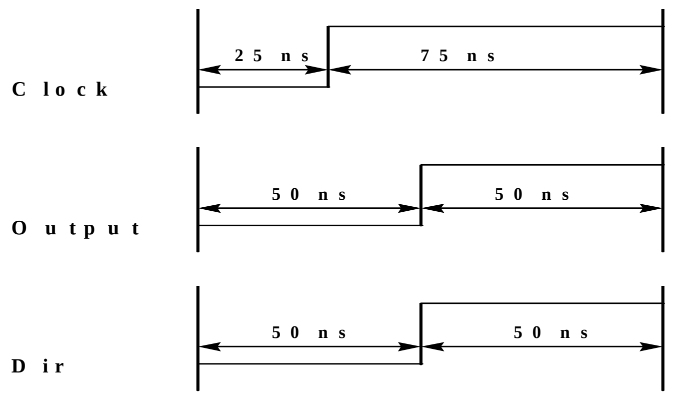

Figure 5.10 Waveforms generated for the Clock and Outputs of the MUT according to the timing specifications in figure 5.9 in the WGP package.

3. The procedure ***WAVEFORM*** for generating the clock waveform in the WGP package is such that, the clock pulses are generated only until the entire external file containing input test vectors are read completely. In the case of the Sobel edge detector system, the outputs are not generated simultaneously as the input vectors are read from the external file and hence, by stopping the clock pulses as soon as the input file is read, the outputs will not be generated completely. In order to solve the problem, we have padded the external file with dummy input vectors equivalent to the amount required for all the outputs for the image to be written

successfully into the output text file. The number of additional test vectors added = (2*NUMCOLS + 3). This value is the same as that calculated in Section 3.2.2.4.The portion of the source code containing the procedure is shown in Figure 5.11.

```
 begin -- waveform generator procedure
loop
          READ_FILE_SLICE( VECTOR_FILE, VECTOR);
          exit when VECTOR.END_OF_FILE;
          APPLY( WPL, VECTOR.CODES.all, WTL( VECTOR.FS_INTEGER ).FSA );
          DELAY( WTL( VECTOR.FS_INTEGER ).PERIOD );
end loop;
END WGP_edge_detector;
```

Figure 5.11 Package for the Waveform Generation Procedure

Waves Testbench file : This file is generated by the WAVES tool. Some source code has been added to the test bench to satisfy the design requirements. Figures 5.12 to 5.17 shows portions of the original source code of the waves test bench along with the necessary modifications. Modifications are noted in bold face type.

```
ENTITY TEST_bench IS
END TEST_bench;
```

Figure 5.12 Portion of the original source code for the entity declaration of the test bench model.

```
 ENTITY TEST_bench IS
generic(OUT_FILE         :STRING(1 to 11);          -- OUTPUT file for storing magnitude Outputs
          DIR_FILE         :STRING(1 to 11);          -- OUTPUT file which stores Direction Outputs
          MAG_GOLD_FILE: STRING(1 to 11);          -- OUTPUT file which stores magnitude Outputs
                                                        -- of the GOLD model
         DIR_GOLD_FILE :STRING(1 to 11); -- OUTPUT file which stores Direction Outputs of the GOLD model
          NUM_ROWS      :NATURAL;          -- number of rows in the INPUT IMAGE
          NUM_COLS       :NATURAL;         -- number of columns in the INPUT IMAGE
          WAIT_CYCLES   :NATURAL);        -- time required before Outputs are written into frame buffers.
END TEST_bench;
```

Figure 5.13 Portion of the modified source code of Figure 5.12 where generic constants have been added.

The test bench model generated by the WAVES tool contains an empty entity declaration as shown in Figure 5.12. Generic constants as required for verification of the MUT had to be specified inside the MUT as shown in Figure 5.13. The generic constants specified are similar to the ones specified in the test bench model shown in Section 3.2.2.4 .

Figures 5.14 shows a portion of the original source code of the architecture body of the test bench. The modified source code for this section is shown in Figure 5.15. In Figure 5.15, two arrays of type *FRAME_IMAGE* AND *FRAME_DIRECTION* similar to the ones declared in the test bench model in Section 3.2.2.4 are defined to store the magnitude and direction outputs of the MUT. Two signals ***RD_START*** and ***INTER*** have been added to satisfy the design needs of the MUT. The ***RD_START*** signal is same as declared in the test bench in Section 3.2.24. This signal is mapped to *EDGE_START* in the port map instead of ***WAV_STIM_edge_start***, because we want to trigger the MUT only after the rising_edge of the clock instead of the falling_edge of the clock. The signal ***WAV_STIM_input*** is mapped to an intermediate signal ***INTER*** which in turn is mapped to the port ***INPUT*** in the port map statements. This is done because again we want to pass the input image data to the MUT only after the rising edge of the clock. These additional signals have been used only to satisfy the requirements of the system.

```
ARCHITECTURE EDGE_DETECTOR_TEST OF TEST_bench IS
--*********************************************************
-- stimulus signals for the waveforms mapped into UUT INPUTS
 --*********************************************************
  SIGNAL WAV_STIM_CLOCK                 :std_logic;
  SIGNAL WAV_STIM_EDGE_START            :std_logic;
  SIGNAL WAV_STIM_INPUT                 :std_logic_vector(  7 downto  0 );
  SIGNAL WAV_STIM_THRESHOLD             :std_logic_vector( 11 downto  0 );
 --******************************************
 -- UUT Port Map - Name Symantics Denote Usage
 --******************************************
 u1: EDGE_DETECTOR
 PORT MAP(
  CLOCK            => WAV_STIM_CLOCK,
  EDGE_START       => WAV_STIM_EDGE_START,
  INPUT            => WAV_STIMINPUT,
  THRESHOLD        => WAV_STIM_THRESHOLD,
  OUTPUT           => BI_DIREC_OUTPUT,
  DIR              => BI_DIREC_DIR);
```

Figure 5.14 Portion of the original source code of the architecture body of the test bench.

```
 ARCHITECTURE EDGE_DETECTOR_TEST OF TEST_bench IS
--FRAME TYPES DEFINED

type FRAME_IMAGE is array(1 to NUM_ROWS,1 to NUM_COLS) of INTEGER;
type FRAME_DIRECTION is array(1 to NUM_ROWS,1 to NUM_COLS) of BIT_VECTOR(2 downto 0);

--*********************************************************
 -- stimulus signals for the waveforms mapped into UUT INPUTS
 --*********************************************************
  SIGNAL WAV_STIM_CLOCK              :std_logic;
  SIGNAL WAV_STIM_EDGE_START         :std_logic;
  SIGNAL WAV_STIM_INPUT              :std_logic_vector(  7 downto  0 );
```

```
   SIGNAL WAV_STIM_THRESHOLD         :std_logic_vector( 11 downto  0 );
   SIGNAL RD_START   :STD_LOGIC:='0';                                    --SIGNAL ADDED
   SIGNAL INTER        :STD_LOGIC_VECTOR(7 downto 0):="00000000";      --SIGNAL ADDED
--*********************************************
  -- UUT Port Map - Name Symantics Denote Usage
  --*********************************************
  u1: EDGE_DETECTOR
  PORT MAP(      CLOCK              => WAV_STIM_CLOCK,
                 EDGE_START     => RD_START,                        --PORT MAPPING CHANGED
                  INPUT             => INTER,                           --PORT MAPPING CHANGED
                 THRESHOLD      => WAV_STIM_THRESHOLD,
                 OUTPUT           => BI_DIREC_OUTPUT,
                 DIR                  => BI_DIREC_DIR);
```

Figure 5.15 Portion of the modified source code of the architecture body of the test bench.

Figure 5.16 shows the portion of the original source code specifying the process for comparison of the output response of the MUT with the expected response in the test bench.

```
 Monitor_OUTPUT:
   PROCESS(BI_DIREC_OUTPUT, WAV_expect_OUTPUT)
   BEGIN
       assert(Compatible (actual => BI_DIREC_OUTPUT,
                   expected => WAV_expect_OUTPUT))
       report "Error on OUTPUT output" severity WARNING;

   IF ( Compatible ( BI_DIREC_OUTPUT,    WAV_expect_OUTPUT) ) THEN
     FAIL_SIGNAL <='L'; ELSE FAIL_SIGNAL <='1';
   END IF;
   END PROCESS;

Monitor_DIR:
   PROCESS(BI_DIREC_DIR, WAV_expect_DIR)
   BEGIN
       assert(Compatible (actual => BI_DIREC_DIR,
                   expected => WAV_expect_DIR))
       report "Error on DIR output" severity WARNING;

   IF ( Compatible ( BI_DIREC_DIR,    WAV_expect_DIR) ) THEN
     FAIL_SIGNAL <='L'; ELSE FAIL_SIGNAL <='1';
   END IF;
   END PROCESS;
```

Figure 5.16 Portion of the original source code for comparison of outputs.

Since we have “dashes” for the expected response, we cannot use this process created by WAVES. Hence a process was developed to compare the output response of the MUT with that of the GOLD model and provide a *pass/fail* indication.. The source code for the comparison process developed is shown in Figure 5.17. Further code has also been added for writing the outputs of the MUT into internal frame buffers.

Let us now consider Figure 5.17. When an internal variable *busy2* is set high, the data from the output text files of the GOLD model ***GOLD_magnitude*** and ***GOLD_direction*** are read into two frame buffers namely ***GOLD_MAG_IMAGE*** and ***GOLD_DIR_IMAGE*** which store the magnitude and direction outputs of integer data type respectively.

We then compare the magnitude and direction outputs of the MUT with that of the GOLD model one pixel at a time and if there is an error, we set an internal variable ***FLAG1*** or ***FLAG2*** to '1' depending on whether the error is in magnitude or in direction. If *FLAG1* or *FLAG2* is set high, we generate a report indicating that the outputs of the MUT don't match with the outputs of the GOLD model, or else a PASS indication is generated. The entire source code of the test bench file can be found in Appendix C

```
--Compare the magnitude and direction outputs with the outputs of the "GOLD MODEL--
when 2=>
   FLAG1:='0';
  FLAG2:='0';
--Compare the magnitude and Direction Outputs with the Outputs of the "GOLD MODEL--
       for i in 1 to NUM_ROWS loop
             readline(GOLD_MAGNITUDE,VLINE1);
            readline(GOLD_DIRECTION,VLINE2);
            for j in 1 to NUM_COLS loop
                   read(VLINE1,A);
            read(VLINE2,B);
                  GOLD_MAG_IMAGE(i,j):=A;
                  GOLD_DIR_IMAGE(i,j):=B;
      if FLAG1='0' then
                  if OUTPUT_MAG_IMAGE(i,j) /= GOLD_MAG_IMAGE(i,j) then
                   --set flag high--
                  FLAG1:='1';
                  end if;
            end if;
            if FLAG2 ='0' then
                   if OUTPUT_DIR_IMAGE(i,j) /= GOLD_DIR_IMAGE(i,j) then
                  --set flag high--
                  FLAG2:='1';
                  end if;
            end if;
            end loop;
      end loop;
      BUSY:=3;
if FLAG1='1' then
-- assert message after comparing all the data
    assert (false) report "MAGNITUDE VALUES DO NOT MATCH -- FAIL";
     else
    assert (false) report "MAGNITUDE VALUES MATCH -- PASS";
end if;
```

```
if FLAG2='1' then
 -- assert message after comparing all the data
        assert (false) report "DIRECTION VALUES DO NOT MATCH -- FAIL";
          else
        assert (false) report "DIRECTION VALUES MATCH -- PASS";
 end if;
```

Figure 5.17 Section of the code where the outputs of the MUT are compared with the expected response

Before any of the above mentioned files are analyzed, we need to first analyze the WAVES library files into the WAVES_STD library and the WAVES/VHDL interface file for IEEE 1164 must be compiled into WAVES 1164. These files can be obtained from the WAVES Home Page at *http://vhdl.org/vi/waves*. These files need to be compiled only once. The WAVES compiler used for generating test benches and the associated files can be obtained at http://server.vhdl.org/vi/waves/.

The Sobel edge detector developed at the register transfer level has been tested in a similar fashion as the MUT at the structural level.