Lab Worksheet 04

Main Objective:

To learn how to perform file I/O in testbenches.

Sub-Objectives:

1) To learn the Read functions available in std.TEXTIO.

Reading Test Vectors from a File: **Part 1:**

Introduction:

When constructing the testbenches for the previous labs the input stimuli and the correct "expected responses" were stored in an array of records. The main process within the testbench reads the elements of the array in a loop, assigned the stimuli values to the inputs of the DUT and compared the actual outputs to the expected responses.

In practice it is better to store the records (called Test Vectors) in a text file and have the main process read from this file. Typically the text file could be created as the output of a computer program written in a language such as C++, Python, Matlab, etc.

Conventions for Tracking Incremental Changes to a Testbench:

For this part we will build upon the testbench that was created in LWS03. We wish to preserve the original work for future documentation. Let's adopt some simple conventions for naming files.

Use the same ModelSim project as used in LWS03-part1.

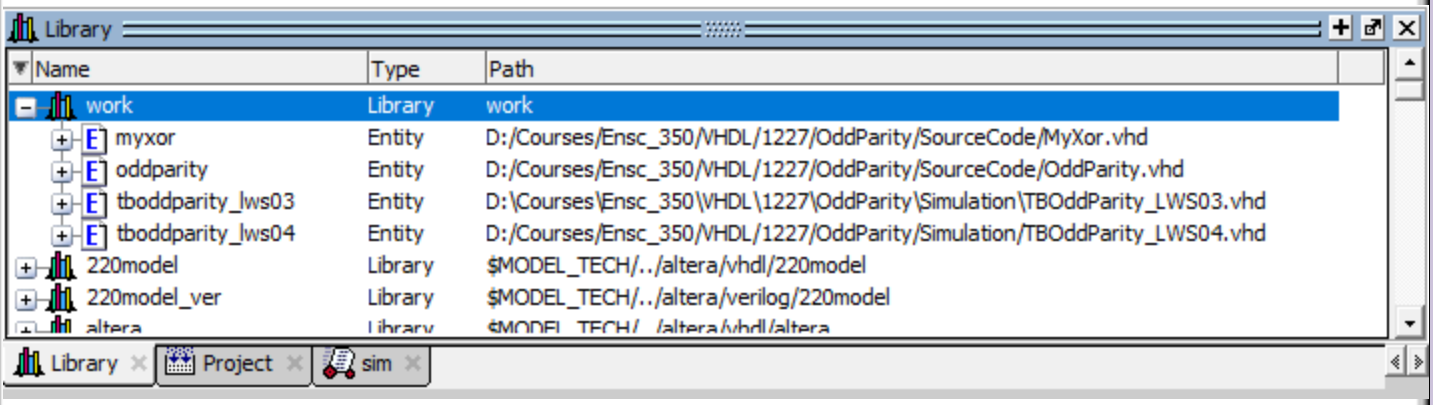
- Rename the original Testbench file used in LWS03-part1 to "TBOddParity_LWS03.vhd".
- Rename the Entity to TBOddPArity_LWS03
- Rename the original script used in LWS03-part1 to "OddParityExperiment_LWS03.do"

We have renamed the Entity so that it is easy to check that we are running simulations on the correct version. The script used to construct the wave window will also need to be modified. Rename the script with the suffix _LWS03 and edit the entries that refer to the testbench entity.

Preparing the files for LWS04:

- Make copies of the testbench files and the scripts but change the suffices to **_LWS04**.
  (*For this lab you will modify the _LWS04 files.*)
- You should always be able to reconstruct the simulation runs from LWS03 using the original files.
- Add the new testbench to the project.
- Update the Entity name to TBOddParity_LWS04.
- Update the script, "OddParityExperiment_LWS04.do" so that it compiles the new testbench, and simulates the correct Entity.
- Update the wave window script so that it adds the correct signals.

Check that the new script compiles the correct files and produces results from a simulation run.
Look in the work library to ensure that both testbench entities are names correctly. You can delete the old version. (the entity without the suffix.)



- You now have a valid script/testbench from LWS03 – needed for future documentation.

Now we shall continue with our studies using the scripts/testbench with the suffix, _LWS04.

Changing the Main Process to Read from a File:

There are four functions that you need. These functions are defined in the package **`TEXIO`**.
LRM-2008 §16.4, page 268-274

FILE_OPEN, FILE_CLOSE, READLINE, and READ – of course there are many more functions that you may wish to review but at present we need only these four.

In VHDL a FILE is an aggregate object similar to an array.
Here is some code from the TEXTIO package, and the LRM-2008 §16.4, page 272.

```
type LINE is access STRING;  --  A LINE is a pointer
                             --  to a STRING value.
  --  The predefined operations for this type are as follows:
  --  function"="  (anonymous, anonymous: LINE) return BOOLEAN;
  --  function"/=" (anonymous, anonymous: LINE) return BOOLEAN;
  --  procedure DEALLOCATE (P: inout LINE);

type TEXT is file of STRING;  --  A file of variable-length
                              --  ASCII records.
  --  The predefined operations for this type are as follows:
  --  procedure FILE_OPEN  (file F: TEXT; External_Name; in STRING;
  --                        Open_Kind: in FILE_OPEN_KIND := READ_MODE);
  --  procedure FILE_OPEN  (Status: out FILE_OPEN_STATUS; file F: TEXT;
  --                        External_Name: in STRING;
  --                        Open_Kind: in FILE_OPEN_KIND := READ_MODE);
  --  procedure FILE_CLOSE (file F: TEXT);
  --  procedure READ       (file F: TEXT; VALUE: out STRING);
  --  procedure WRITE      (file F: TEXT; VALUE: in STRING);
  --  procedure FLUSH      (file F: TEXT);
  --  function  ENDFILE    (file F: TEXT) return BOOLEAN;
```

FILE_OPEN – associates the OS filename with the VHDL FILE, & sets the access mode.
We shall adopt a convention that all test vector files will have the file suffix, **.tvs**.

```
Constant        TestVectorFile : string := "MyTestVectors.tvs";
FILE            VectorFile : TEXT;

FILE_OPEN( VectorFile, TestVectorFile, read_mode );
report "Using TestVectors from file " & TestVectorFile;
```

FILE_CLOSE – After all measurements are completed the file must be closed.

```
    Report "Simulation Completed";
    File_close( VectorFile );
    Wait;
End Process Main;
```

1) Modify the main process in the testbench so that a file containing test vectors is opened before the measurement loop and closed after the loop has completed.

2) Modify the loop in the main process so that it is controlled from the file and not the measurement index. Initialize and increment the MeasurementIndex as before, but remove the MeasurementIndex from the loop condition.

```
    while not ENDFILE( VectorFile ) loop

    End loop;
```

Each READ, SREAD, OREAD, and HREAD procedure declared in package TEXTIO extracts data from the beginning of the string value designated by parameter L and modifies the value so that it designates the remaining portion of the line on exit. Each procedure may modify the value of the object designated by the parameter L at the start of the call or may deallocate the object.

The READ procedures defined for a given type other than CHARACTER and STRING begin by skipping leading whitespace characters. A whitespace character is defined as a space, a nonbreaking space, or a horizontal tabulation character (SP, NBSP, or HT). For all READ procedures, characters are then removed from L and composed into a string representation (see 5.7) of the value of the specified type. The READ procedure for type BIT_VECTOR also removes underline characters from L, provided the underline character does not precede the string representation of the value and does not immediately follow another underline character. The removed underline characters are not added to the string representation. For all READ procedures, character removal and string composition stops when the end of the line is encountered. Character removal and string composition also stops when a character is encountered that cannot be part of the value according to the rules for string representations, or, in the case of the READ procedure for BIT_VECTOR, is not an underline character that can be removed according to the preceding rule; this character is not removed from L and is not added to the string representation of the value. The READ procedures for types STRING and BIT_VECTOR also terminate acceptance when VALUE'LENGTH characters have been accepted (not counting underline characters in the case of the READ procedure for BIT_VECTOR). Again using the rules of 5.7, the accepted characters are then interpreted as a string representation of the specified type. The READ does not succeed if the sequence of characters composed into the string representation is not a valid string representation of a value of the specified type or, in the case of types STRING and BIT_VECTOR, if the sequence does not contain VALUE'LENGTH characters.

READLINE – Reads one complete line from the file as a string. For convenience, the test vector file should be constructed so that each record is on a single line.

SREAD, OREAD, HREAD – are variations of the read function. LRM-2008 §16.4, page 273,274. (*StringRead, OctalRead & HexRead*)
READ - The read function parses a string to find the leading sub-string separated by <white space> The sub-string must be formatted correctly to be a string representation of the data type of the specified variable.

Each pass through the loop

- assigns values to the input signals and expected responses,
- waits, and then
- tests the DUT outputs against the expected response.

3) Modify the loop body so each pass fills a string buffer with one line (each record) from the file.

```
Variable        LineBuffer : line;

READLINE( VectorFile, LineBuffer );
```

4) Declare process variables corresponding to the DUT signals and True responses. READ values from the line buffer into the process variables and then assign the variable values to the corresponding signals.

```
Variable Xvar : std_logic_vector( N-1 downto 0 );
Variable TrueIsOddvar : std_logic;

hread( LineBuffer, Xvar );
read( LineBuffer, TrueIsOddvar );

-- Assign input stimulii variables to the signals.
TBX <= Xvar;
TBtrueIsOdd <= TrueIsOddvar;
```

If the HREAD function is not found, check that the testbench file is being compiled using VHDL-2008, *right-click on the source file and select properties*)

You can also specify the VHDL version when you invoke the compiler within a script.

vcom -work work -2008 -explicit -stats=none D:/OddParity/Simulation/TBOddParity_LWS04.vhd

5) Manually enter values into the test Vector file and run the script. You should see the stimuli values and the responses in the wave window. Intentionally choose a bad value for the expected response to convince yourself that the assert works correctly.

Lab Worksheet 04

LWS04 LWS04

Task ID	Show to TA during Lab Period for feedback and Lab Tick	Demonstration Required Signup for a demo time on Canvas	Documentation Required Canvas Submission	Documentation Required for feedback and Lab Tick
LWS-04-P1 (01-05)	☒	☒	☒	☒