Lab Worksheet 05

Part 1:

Main Objective:

To learn about VHDL timing models.

Sub-Objectives:

1) To learn transaction scheduling of VHDL signals .

2) To learn how Quartus timing models are used in ModelSim simulations

Part 2:

Main Objective:

To analyse DUT timing within a testbench.

Sub-Objectives:

1) To learn about built-in VHDL functions.

2) To learn the basic use of VHDL Components & Configurations.

3) devise a scheme to automate measurements of propagation delay.

Part 3:

Main Objective:

To analyse both cost & timing for the Div7 circuit.

Sub-Objectives:

1) To create and verify a low-cost and high-performance implementation of Div7, optimized for a Cyclone 4, EP4CE-115-F29-C7

Lab Worksheet 05

Adding Timing Information to a Simulation Model: Part 1:

Introduction:

A design is first simulated to verify that it correctly processes values. This is called “functional” simulation. A second simulation is then performed to verify that signal timing is acceptable. This simulation is referred to as “Timing” simulation.

The VHDL code for a design may include information to model the timing behaviour. Timing information is ignored by a synthesis tool. Timing properties of a design depend on the technology used for implementation. If we know precise details about the implementation technology we could create our own timing models and include this in our design code. Alternatively, tools such as Quartus know the final implementation technology and contain measured data about the timing. For example, the LUTs within a Cyclone IV are known and thus once Quartus has fitted a circuit it can back-annotate the VHDL code with timing information.

- The following part is an extension of the work accomplished in LWS04.

VHDL Signals: LRM-2008 §11.6, page 174-176, LRM-2008 §10.5, page 149-159

A VHDL signal is a communication channel between simulation processes. All concurrent statements are simulation processes. The following code segments are equivalent.

A concurrent process statement:

```
assg: process( SigA )
      begin
         SigB <= SigA after 10 ps;
      end process assg;
```

A concurrent signal assignment statement:

```
SigB <= SigA after 10 ps;
```

concurrent_signal_assignment_statement ::=
  [label :] [postponed] concurrent_simple_signal_assignment
  | [label :] [postponed] concurrent_conditional_signal_assignment
  | [label :] [postponed] concurrent_selected_signal_assignment

concurrent_simple_signal_assignment ::=
  target <= [guarded] [delay_mechanism] waveform ;

concurrent_conditional_signal_assignment ::=
  target <= [guarded] [delay_mechanism] conditional_waveforms ;

concurrent_selected_signal_assignment ::=
  with expression select [?]
    target <= [guarded] [delay_mechanism] selected_waveforms ;

delay_mechanism ::=
transport | [reject time_expression] inertial

waveform ::=
  waveform_element { , waveform_element }
  | unaffected

conditional_waveforms ::=
  waveform **when** condition
  { **else** waveform **when** condition }
  [**else** waveform]

selected_waveforms ::=
  { waveform **when** choices , }
  waveform **when** choices

Lab Worksheet 05

All signals have a driver. LRM-2008 §14.7.2, page 214-215.
A driver is the node where values are placed on the signal. (outputs of a simulation process)
The driver is implied by the target signal on the left-hand side of a concurrent signal assignment operator.

The driver is a queue containing many transactions. A transaction is a (time,value) pair.

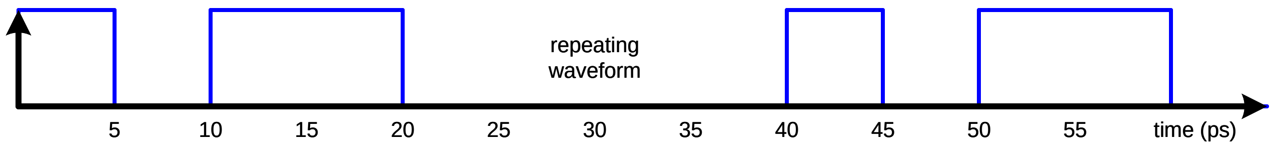
```
SigB <= SigA after 10 ps, '0' after 20 ps, '1' after 30 ps;
```

1) Add an new internal signal to the testbench called clock. (std_logic) Make the following (repeating) waveform using a process.

- Add the clock to the waveforms and update the waveform script.
- Run the simulation.

2) Remove the concurrent process and make the waveform using only concurrent signal assignments. (you may need two signal assignments; one to create a symmetric clock and a second to make the pulsating waveform.)

- Run the simulation.



3) Let's assume that a leaf-node xor gate has a contamination delay, tcd = 15 ps and a propagation delay, tpd = 20 ps. If not already done, modify the leaf-node xor entity so that it includes this timing model.

- Run the simulation.

Lab Worksheet 05

Timing Simulations:

Part 2:

Quartus knows the timing information for Intel FPGAs and can modify your VHDL to include accurate timing models. The new VHDL model is stored in a file with the suffix .**vho** and the values for the timing parameters are stored in a file with suffix **.sdo**.
ModelSim reads the timing parameter and back-substitutes the values into the model.

1) Make a folder structure for a new project. Create a Quartus project and a ModelSim project.
refer to the document "LWS-01.2-SW-Install-Setup-350-xxxx.pdf" for correct settings.

2) The following Entity is a recursive Magnitude Comparator.
Synthesise and Fit the Entity for a Cyclone IV EP4CE-115-F29-C7 FPGA.

- Check that the design uses 107 Logic Elements.
- Check that MagComp.vho and MagComp_vhd.sdo were created in simulation/modelsim.

```
Library ieee;
Use   ieee.std_logic_1164.all;

Entity MagComp is
Generic ( N : natural := 64 );
Port (
        X, Y : in std_logic_vector( N-1 downto 0 );
        EQ, GT : out std_logic );
End Entity MagComp;

Architecture rtl of MagComp is
        Signal   Xup, Yup, Xlo, Ylo : std_logic_vector( N/2-1 downto 0 );
        Signal   EQup, GTup, EQlo, GTlo : std_logic;
Begin

Recur:   If N > 1 Generate

         Xup <= X(N-1 downto N/2);
         Yup <= Y(N-1 downto N/2);
         Xlo <= X(N/2-1 downto 0 );
         Ylo <= Y(N/2-1 downto 0 );

Upper:   Entity work.MagComp(rtl) Generic map( N => N/2 )
              port map( X => Xup, Y => Yup, EQ => EQup, GT => GTup );

Lower:   Entity work.MagComp(rtl) Generic map( N => N/2 )
              port map( X => Xlo, Y => Ylo, EQ => EQlo, GT => GTlo );

         EQ <= EQup and EQlo;
         GT <= GTup or (EQup and GTlo);

         End Generate Recur;

Leaf:    If N = 1 Generate
             EQ <= X(0) xnor Y(0);
             GT <= X(0) and not(Y(0));
         End Generate Leaf;

End Architecture rtl;
```

3) Close Quartus and start the ModelSim project.
Write a testbench for the Magnitude Comparator that uses an array containing 5 test vectors.
You may wish to modify a copy of the testbench from LSW03 part 1.

- Label the instance of the Magnitude Comparator, DUT:
- Loop through five measurements with an measurement index variable.
- Use assert/report to verify correct results.
- Include PREPTIME = 40 ns and MEASTIME = 100 ns.

4) Verify that everything is functionally correct.

- Add the testbench to the ModelSim project & Compile.
- Start a simulation of the testbench
- Add signals to a wave window – save the script.
- Run the simulation for 800 ns to verify that everything is functionally correct.
- End the simulation.

5) Quartus's version of the design is an architecture called "structure".
Modify the testbench to use this version as the DUT.

- Add MagComp.vho to the project.
- Change the Testbench instance to refer to the "structure" architecture.
- Compile, - verify that both architectures of MagComp are now in the work library.

You may notice a problem when both MagComp.vhd and MagComp.vho are both added to the project. Both files contain a declaration of the same entity. If both files are compiled then the simulator may indicate that an entity has changed. This depends on the compile order. Furthermore, the declaration in the .vho does not contain the generic parameter. If the DUT instance explicitly refers to a generic parameter there may be an error. This error is also dependent on the compile order.

- Ensure that the .vho compiles before the vhd. and
- Remove any mapping to the generic parameter in the testbench thus implying the default value.

6) Create two scripts, one for functional verification and the other for timing verification.
(ModelSim Command Reference: VCOM-page 304, VSIM-page 455, Transcript-page 273)
Each script should

- ✓ End a simulation (just in case the simulator is already running)
- ✓ Set a filename for the transcript file.
- ✓ Compile the correct sourcecode in the correct order. (example)
  - vcom -work work -2008 -explicit -stats=none ../SourceCode/MagComp.vh**?**
- ✓ Start a simulation
- ✓ Turn off the transcription
- ✓ setup the wave window – by calling a script.
- ✓ Turn on the transcription
- ✓ restart
- ✓ run for 800 ns
- ✓ turn off the transcription
- ✓ name the transcript file as “”. (to prevent further messages from being placed in the file.)

If you try running the Functional Script you will need to change the DUT instance in the testbench back to the rtl architecture. ignore the functional script for now.

7) Try to start a timing simulation. The simulator will load successfully but the model is incorrect because it is missing the timing information.

- a run will produce an error,
  - ➢ # GetModuleFileName: The specified module could not be found.
- This is because we haven’t included the .sdo file.

The .sdo file is not VHDL. It contains information need by tools within the simulator for back annotation. The file contains timing information using an industry standard format called Standard Delay Format. (SDF) - *IEEE-Standard for SDF-1497-2001*

8) There are two ways to tell the simulation tool about the .sdo file

- manually – when you start the simulation
  - ✓ select the SDF tab
  - ✓ browse to the .sdo file,
  - ✓ enter the region as DUT
- within a script, when starting the simulation
  - ✓ vsim -t 100ps -gui -sdftyp ../DUT=ModelSim/MagComp_vhd.sdo work.DesignUnit

9) Run a timing simulation.
This time everything should load correctly and you will see the message
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.

10) You should now notice that the output signals from the DUT are delayed by 10 ns to 40 ns. These are the actual delays cause by the LUTs within the Cyclone IV FPGA.

- Use the yellow measurement cursors to manually estimate the propagation delays.

The testbench explicitly references the architecture of the DUT. As such, the testbench must be edited, depending on whether the functional script or the timing script is being executed.

Now its time to learn about components and configurations.
You can read about components & configurations.

- Configuration declarations - LRM-2008 §3.4, pages 13-18
- Component declarations - LRM-2008 §6.8, page 93
- Configuration specification - LRM-2008 §7.3, page 98-103

11) Clean up: Removing items to avoid conflicts.

- Delete all design units within the work library.
- Remove both MagComp.vhd & MagComp.vho from the project.
- Compile

You should see an error because the pre-compiled entity does not exist in the library.

12) Changing the DUT instance to a component.

- Declare a component called TestUnit, with the same port specification as MagComp.
- Change the DUT instance to instantiate the component TestUnit. The port signal are mapped the same so no changes are necessary.
- Compile the testbench. Notice that the compilation is successful even though there is no corresponding sourcecode for the entity and the work library does not contain the entity.

The VHDL language defines five kinds of design units, (that can be compiled separately)

- Entity declaration,
- Architecture body,
- Package declaration,
- Package body &
- Configuration declaration.

13) Create two configurations.

Create a file called ConfigMagComp.vhd in the SourceCode folder.
Declare two configurations;

- one that binds Entity MagComp(rtl) to the component TestUnit, and
- another that binds Entity MagComp(structure) to the component TestUnit.
- Add the .vhd & .vho source files back into the project and compile.
- Compile the file, ConfigMagComp.vhd. Notice that the library now contains both configurations.

```
Configuration ConfigName of TBEntity is
  for behavioural
        for DUT : ComponentName use entity
                    work.EntityName(ArchitectureName);
        end for;
  end for;
End Configuration ConfigName;
```

Lab Worksheet 05

From now on, when you start the simulator, you simulate a configuration, not the testbench.

14) Edit both scripts
- Include re-compilation of the configuration, and
- When starting in the simulator, reference the appropriate configuration (*with the VSIM command*.) instead of the testbench.

You should now be able to run each script independently and reproduce the results from functional simulation and also timing simulation.

Measuring Propagation Delay:

Review LRM-2008 §16.2, page 239-254 about object attributes. We are mostly interested in the signal attributes, in §16.2.4 . (notice that you have already encountered **'event**)
Attributes:
- S'Quiet(T), & S'Stable(T) – return signals.
- S'Active, & S'Event – return Boolean values.
- S'Last_Active, & S'Last_Event – return times.

To store the value of the current simulation time, assign the value NOW to a variable of type time.

When the stimuli are unchanging, the outputs of a combinational circuit should eventually become stable. A human being would look at the signal waveform and use their intelligence to determine when the signal has become stable.
15) Choose a suitable amount of time that you consider sufficient to ensure that the signal is stable. Declare a constant of type time, STABLETIME, for your choice.

16) Modify the testbench to determine propagation delays.
- Declare 3 variables of type time, StartTime, EndTime & PropDelayEQ.
- Use StartTime to mark the beginning of each measurement,
- Change the measurement wait statement to wakeup when the EQ signal has finished changing. It is up to you do determine which attributes to use.
- Use EndTime to mark this time.
- Use PropDelayEQ to calculate the propagation delay of the EQ signal.

17) In the script, change the run command to run -all. Verify that the script performs the measurements and terminates reasonably.

18) Display the measurement time markers.
- Add the three time variables to the wave window just below the measurement index.
- Format these three waveforms and save the window script.

19) In the testbench, include a `report` statement to print strings to the transcript.
For each measurement, the string should provide information about the measurement index and the propagation delay. -- to_string(PropDelay) will format time values nicely.

The final issue that you may find useful is measuring the propagation delay for a group of signals. For this example, there are two signals, EQ and GT.

- We should only mark the end of the measurement when both signals are stable.
- The propagation delay for individual signals can be calculated for the endtime using the attributes ‘Last_Active and ‘Last_Event.

20) Finding the EndTime.

- Create a signal array called DUTout and assign EQ and GT to the elements.
- Using the attributes of DUTout, find the endtime.
- Using two variables, PropDelayEQ and PropDelayGT, calculate the individual delay times.
- Update the wave window and save the script.
- Update the transcript report.

21) Using the measurement cursors, verify that the delay values written to the transcript are correct.

Optimized Div7: **Part 3:**

The present task is to iterate the design of the Div7 circuit from a previous LWS. The new design is to be optimized for both cost and timing, specifically targeted for implementation on a Cyclone 4, EP4CE-115-F29-C7 gate array.

1) Make a folder structure for a new project. Create a Quartus project and a ModelSim project. refer to the document “LWS-01.2-SW-Install-Setup(QnM)-350-xxxx.pdf” for correct settings.

2) Review the structure of the primitive logic elements withing the target gate array.

3) Review the lecture notes, specifically noting the methods for high-performance arithmetic circuits. Consider whether any of the techniques may be applicable for this particular design.

4) Write a testbench, test vectors & related scripts to verify the timing of an implementation of Div7.

5) Optimize the VHDL code for Div7, you should achieve a cost of less than XX logic elements and a worst-case timing of less than XX ns.

Lab Worksheet 05

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-05-P1 (01-02)	☒	☒	☒	☒
LWS-05-P1 (03)	☑	☒	☒	☒
LWS-05-P2 (14,21)	☑	☒	☒	☒
LWS-05-P3 (5)	☑	☒	☒	☒