Lab Worksheet 03

Main Objective:

1) To enhance your knowledge of Circuit Verification.

Organising Project Files:

Your project should have a main top-level directory that contains

1) The Quartus project file, Quartus setting file.
2) All database and other directories/files created by Quartus.
3) A Directory called Source.
4) A Directory called Simulation.

The Directory called Source will contain your synthesisable VHDL design files. These files will be referenced by both Quartus and ModelSim.

The Directory called Simulation will contain

1) The ModelSim Project file.
2) The ModelSim work library.
3) Your VHDL (non-synthesisable) Testbenches. Use the convention that a testbench that tests an entity called “EntityName” will be called “TBEntityName”.

PART 1:

Objective:

1) To design, verify & test a simple combinational circuit.

Specification:

Behaviour : The circuit tests an 18-bit unsigned input to determine whether the unsigned value is divisible by 7. The output is true (‘1’) if the input value is divisible by 7.

Resource Constraints: The circuit must consume no more than 31 LUTs on a cyclone IVE.

Timing Constraints: Using the Full Adder Entity that I provide the simulation response time must be no longer than XX ns.

Lab Worksheet 03

1) **Preparing the Project Files:**

Set up the directory structure with the top-level called Div7.
I have provided a skeleton for an ENTITY Div7 and also a skelton Testbench.

Testbench convention:
Instantiate the DUT: create internal signals that have identical names to the port signals of the DUT entity.
Include three PROCESSes, labelled "**Stimulus**:" , "**TBDUT**:" & "**Detector**:".

This is a fictitious step; it is indended for your information.
Create a new Quartus project called Div7. Remember to import the pins.
Add the TestBench and add Div7.vhd to the Quartus project.
Set the TestBench as the top-level Entity and synthesise the project.
Make notes of your observations.

2) **Preparing the ModelSim Project:**

Close Quartus. Start ModelSim and create a new project in the Simulation Directory.
Name this project Div7.
Add the Testbench and other necessary VHDL source files.
Compile the skeleton code to ensure that there are no errors.

3) **Preparing the Testbench:**

The stim and detect processes are blank,
Complete the stimulation process to do the following.
a) Contains an infinite loop. (All PROCESSes are infinite loops)
b) sets x <= unknown, clears the start and then waits for a rising edge on Clock.
c) Once synchronised to the clock, assert an new value for x and assert start.
This indicates to the detector that the stimulus inputs have be initialized and that a measurement has begun.
d) wait for the Done = '1' before looping.
We assume that the detector de-asserts the done signal in response to the start signal being cleared.

Test values:
create an array of 10 constants.
For each constant, K, test K+0, K+1, K+2, K+3, K+4, K+5 and K+6.
One of these should be divisible by 7.

2) The detector process:
Waits for a rising edge on Start and then clears the Done.
Using clock events, Check to see if IsDivisible has been stable for "StableTime"
If So, Use ASSERT to REPORT errors, and update Done = '1';

PART 2:

Objective:

1) To design & verify an RV64I Logic Unit.

Specification:

behaviour : 18-bit unsigned input, output true i value is divisible by 7
resource constraint: Estimate the required LUTs. Synthesis and check.
Timing Constraints: None specified at present. To be defined later.

1) **Testing - Preparing the Quartus Project:**

Directories as described at the beginning.
New Quartus Project
Import Pin.
Create Top-Level Entity, - Will Quartus allow us to have multiple TestBeds in the same project? If yes then we'll do this, If NO then ---?

2) **Verification - Preparing the ModelSim Project:**

New ModelSim Project.

3) **Designing the Logic Unit:**

Remove the TBY process from the architecture and now code the correct architecture using rtl signal assignments.

You may use primitive VHDL operators. This is the first step of a much larger project. We are attempting to incrementally build a RISC-V processor. This is not a fake exercise like the previous labs so do not use lucky gates or the FA. These were provided in previous labs simply to ensure fake timing when simulated. We will eventually build this device and thus the timing will be a real consequence of the actual FPGA.

1) **Verification - Preparing the Testbench:**

Same protocol as used in part 1.
coverage: for each bit, 0..63 – perform 16 tests – 4 rows of the truth table on Ai, Bi, and AND, OR ,XOR and PASS for each row.

Havew the detector assert a message if
ASSERT Y'stable(4 ns) = TBY provide a REPORT message that gives, A, B, LogicFN and Y, TBY.
also include a compare signal true is TBY != Y, simple to show in the waveforms.

Run the TB with a blank DUT to check that the stimulus is correct, and the TBY is correct.

To remove the errors, temporarily copy/paste the TBDUT code into the DUT architecture and check that the errors disappear.

3) **Verification - Running a Simulation:**

Run the simulation and ensure that the device behaves correctly.

Lab Worksheet 03

PART 3:

Objective:

1) To design & verify an RV64I Arithmetic Unit.

Specification:

behaviour : 18-bit unsigned input, output true i value is divisible by 7
resource constraint: Estimate the required LUTs. Synthesis and check.
Timing Constraints: None specified at present. To be defined later.

1) **Preparing the Testbench:**

2) **Preparing the Testbench:**

Lab Worksheet 03

Main Objective:

1) To enhance your ability to code Hierarchical Circuits in VHDL.

2) To verify (using simulation) the timing of various Addition Circuits.

Start the program ModelSim. Create a new folder to store your project files before creating a new project (called "LWS02"). Create a file called LuckyGates.vhd. When the "create" window appears you should add the existing file called "LuckyGates.vhd". Compile the code so that the entities appear in the Work Library. Check that the ENTITIES are now inside the Work library.

Please demonstrate to the Teaching Assistant the following.

FOR PART 1: (*deadline: beginning of Lab on Feb 7th 2020*)

Chain1:	VHDL architecture / Circuit Diagram (drawn in your lab notebook)
Tree1:	VHDL architecture / Circuit Diagram
Chain2:	VHDL architecture / Circuit Diagram
Tree1:	VHDL architecture / Circuit Diagram

Simulation results showing the output signal waveforms of all four circuits in ONE diagram.

FOR PART 2: (*deadline: beginning of Lab on Feb 14th 2020*)

Ripple:	VHDL architecture / Circuit Diagram and waveforms with expanded sum bits.
BookSkip:	VHDL architecture / Circuit Diagram and waveforms with expanded sum bits.
GoodSkip:	VHDL architecture / Circuit Diagram and waveforms with expanded sum bits.
Brent-Kung:	VHDL architecture / Circuit Diagram and waveforms with expanded sum bits.

Table of predicted and measured timing. (drawn in your lab notebook)

All circuit diagrams must be complete showing every element of the VHDL architecture.

- All internal signals must be labelled identically to the VHDL code.
- Do not omit elements simply because a structure is repetitive.
- If the circuit is recursive you should draw TWO circuit diagrams; one for the recursive pattern and one for the stopping condition. Both circuits should be labelled identically to your VHDL code.

Please refer to the Handout-10-252-1167.pdf for further information hierarchical circuit descriptions.