Homework set 2

(These problems should be done individually, not with project partner)

Due – As announced or on Canvas

Instructions for submission

1) All answers are to be submitted in a **single** word/pdf file and as separate .vhd files for each entity required.

2) Include Quartus Prime, Modelsim/Altera vwf and observation screenshots whenever possible with appropriate annotations for better presentation in the word/pdf file.

Do not submit extra screenshots in the submission folder

3) Naming convention and files required

a) LastName_FirstName_HWSet2.docx/pdf - for Observations and Answers and all code listings

b) Also Submit <Codefilename>.vhd/.v - separately for each question externally.

c) Follow the module (.v) or Entity name (.vhd) and port Name as per guidelines in each question. These modules will be auto graded and will not compile if the names do not match.

** Commenting of code, references and presentation of material also carries weight*

1. Design a comparator

   Using VHDL, design a 2-bit version of the comparator in the video. Using the entity below also given in HW2P1.vhd, create the architecture.

```
library ieee;                                   -- line 1
use ieee.std_logic_1164.all;                    -- line 2
                                                -- line 3
entity comparator2 is port (                    -- line 4
    A, B: in std_logic_vector(1 downto 0);  -- line 5
    Equals: out std_logic);                     -- line 6
end comparator2;                                -- line 7
```

2. Find the Code Errors.
   Determine which lines have syntax errors in the accompanying VHDL code, which is also found in HW2P2.vhd:

```
library ieee;                          -- line 1
use ieee.std_logic_1164.all;           -- line 2
                                       -- line 3
entity find_errors is (                -- line 4
    a: bit_vector(0 to 3);             -- line 5
    b: out std_logic_vector(3 to 0);   -- line 6
    c: in bit_vector(5 downto 0))      -- line 7
end find_errors;                       -- line 8
                                       -- line 9
architecture not_good of find_errors   -- line 10
  begin                                -- line 11
  my_label: process                    -- line 12
    begin                              -- line 13
    if c = x"F" then                   -- line 14
      b <= a;                          -- line 15
    else                               -- line 16
     b <= '0101';                      -- line 17
    end if                             -- line 18
  end process;                         -- line 19
end not_good                           -- line 20
```

Line number, error 1 _____
Line number, error 2 _____
Line number, error 3 _____
Line number, error 4 _____
Line number, error 5 _____
Line number, error 6 _____
Line number, error 7 _____
Line number, error 8 _____
Line number, error 9 _____
Line number, error 10 _____

3. Correct the Code.
   Correct the VHDL code given in problem #2 and submit it as HW2P3.vhd. You may need to us a conversion function, like to_stdlogicvector() to complete this. Compile and **simulate** the results.

```
library ieee;                                    -- line 1
use ieee.std_logic_1164.all;                     -- line 2
                                                 -- line 3
entity find_errors is port (                     -- line 4
    a: in bit_vector(0 to 3);                       -- line 5
    b: out std_logic_vector(3 downto 0);         -- line 6
    c: in bit_vector(5 downto 0));               -- line 7
end find_errors;                                 -- line 8
                                                 -- line 9
```

4. Majority Vote

   Using VHDL, design a majority vote circuit, which outputs a logical 1 only if 2 or more of 3 inputs are a 1. Using the entity below also given in HW2P4.vhd, create the architecture. Simulate the compiled results.

```
library ieee;
use ieee.std_logic_1164.all;

entity Majority is port (
   A, B, C: in std_logic;
   Y: out std_logic);
end Majority;
```

5. Develop a VHDL model of the 74LS163 4-bit binary counter. The model should be **functionally equivalent**, but timing differences will be allowed.

   a. Follow 74LS163 datasheet for naming the input and output signals in accordance to IC. Also see the datasheet on Canvas for design information.
   b. Compare the timing differences using Fmax between the VHDL FGPA implementation and the TTL chip by building the counter in a MAX10. Use a databook or the datasheet on to get the TTL data for comparison.
   c. Simulate your circuit design.

   *****Please follow the below Entity name and port names as below, else it will not compile in the Autograder (Be careful it is case sensitive) *****

```
entity HW2Q5 is port (
   CP: in std_logic;  -- clock
   SR: in std_logic;  -- Active low, synchronous reset
   P:  in std_logic_vector[3 downto 0];  -- Parallel input
```

```
   PE: in std_logic;  -- Parallel Enable (Load)
   CEP: in std_logic;  -- Count enable parallel input
   CET: in std_logic; -- Count enable trickle input
   Q:   out std_logic_vector[3 downto 0];
    TC: out std_logic  -- Terminal Count
);
end HW2Q5;
```

6. Design a comparator

   Using Verilog, design a 2-bit version of the comparator in the video. Using the module beginning below also given in HW2P6.v, create remainder of the module and simulate it.

```
Module HW2P6(
   input[1:0] A, B;
   output Equals;
);
```

7. Develop a Verilog model of the 74LS163 4-bit binary counter. The model will be functionally equivalent, but there will be timing differences.

   a. Follow 74LS163 datasheet for naming the i/p and o/p signals in accordance to the IC
   b. Compare the timing differences between the Verilog FPGA implementation and the TTL chip by building the counter in a Cyclone IV. Use a databook or the datasheet on Canvas to get the TTL data for comparison.
   c. Simulate

*****Please follow the below module name and signal names as below, otherwise the test bench will not compile (Be careful it is case sensitive) ******

```
module HW2P7 (P, CP, SR, PE, CEP, CET, Q, TC);
    // Define input and output
    input [3:0] P;       // Parallel Input
    input CP;              // Clock
    input SR;              // Active Low Synchronous Reset
    input PE;              // Count Enable Parallel Input
    input CEP;            // Count Enable Parallel
    input CET;            // Count Enable Trickle
    output [3:0]Q;       // Parallel Output
    output TC;              // Terminal Count
```

8. Implement a 128 by 32 RAM using VHDL and the Altsyncram function. Do not use registered output options. Target the design to the Cyclone IV smallest device. Use the timing analyzer to determine the worst-case read and write access times for the memory. Use the IP Catalog of Quartus Prime/Quartus II (not Qsys) to instantiate the Altsyncram function to generate the memory. For Access time analysis you need to do gate level (post synthesis) simulation using ModelSim. Implement this also in a MAX10 and report what the timing simulation tells you.

9. Implement a 128 by 32 RAM using Verilog and the Altsyncram function. Do not use registered output options. Target the design to the Cyclone IV smallest device. Use the timing analyzer to determine the worst-case read and write access times for the memory. Use the IP Catalog of Quartus Prime/Quartus II (not Qsys) to instantiate the Altsyncram function to generate the memory. For Access time analysis you need to do gate level (post synthesis) simulation using ModelSim.

Grading Rubric

1) - [3 points]
    ----Auto Graded
    [2 points] VHDL Code
    [1 points] Implementation
    ----Screen Shot from submission
2) - [4 points]
    [0.4 point] each correct line number with error

3) - [3 points]
    ----Auto Graded
    [2 points] VHDL Code
    [1 point] Implementation and Simulation
    ----Screen Shot from submission

4) - [3 points]
    ----Auto Graded
    [2 points] VHDL Code
    [1 point] Implementation and Simulation
    ----Screen Shot from submission

5) - [8 points]

----Auto Graded

[2 points] VHDL Code

[2 points] Implementation

[2 points] Working functionality of 74LS163, All Scenarios covered, verified by simulation and timing analysis.

[2 points] Comparison

** Commenting of code, references and presentation of material also carries weight*

6) - [3 points]

[1 point] Verilog Code and Implementation

[1 point] Simulation using Modelsim

[1 point] Timing Analysis

7) - [8 points]

[3 point] Verilog Code, Implementation 74LS163 functionality

[3 point] Simulation -All Test Scenarios, using Modelsim

[2 point] Comparison

8) - [4 points]

[1.5 points] Implementation

[1.5 points] Simulation

[1 points] Access Time

9) - [4 points]

[1.5 points] Implementation

[1.5 points] Simulation

[1 points] Access Time

Guides for Modelsim and simulation

1) **Video for University Waveform for Quartus Prime**

https://www.youtube.com/watch?v=a8JAkKhxlQI

2) **ModelSim GUI -Quartus Guide**

a) ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Tutorials/VHDL/ModelSim_GUI_Introduction.pdf

b) http://staff.cs.upt.ro/~opritoiu/modelsim/simex1/en/index.html

3) **Modelsim post synthesis simulation guide**

http://cseweb.ucsd.edu/classes/fa10/cse140L/lab2/Lab2_tutorial_timinig_simulation.pdf