EEL 3701C – Digital Logic & Computer System

LAB 2
(35 Pts, 12 Pts Bonus, Due October 02 2022, 11:59 PM)

Instructional Objectives
At the end of this lab, you should be able to understand:

- synchronous and clock divider circuits
- configuration of the seven-segment display
- encoding BCD numbers

Goal: In this lab, we will first introduce synchronous circuits. These are circuits that depend on a clock signal for proper operation. Outputs are considered valid in a synchronous circuit only when the clock cycle changes. The second part of the lab will deal with combinational circuits, namely the design of a washing machine controller and a seven-segment display controller.

Problem 1: Design and Implementation of a synchronous (clocked) circuit on FPGA (25 Pts)
A synchronous circuit takes the clock's particular signal as input and makes the output dependent on that signal. The clock signal generates a sequence of repetitive pulses at a certain speed. The distance between two clock pulses is called the delay, which can be computed as the inverse of the frequency.

Part I

Pre-Lab Homework (5 pts)
1- Design a VHDL circuit that takes two inputs and a clock and generates a signal that is a random logic combination of the two inputs. The choice of the logic combination is left at your discretion. Make your output dependent on the clock. Compile your circuit and make sure there are no errors. **(5 pts)**

In-Lab Implementation (5 pts)

1- Use the following signal mapping (ignore reset signal) to implement and test your code on the FPGA board. **(5 pts)**

Node Name	Direction	Location	I/O Bank	/REF Grou	'O Standar
A	Input	PIN_J2	2	B2_N0	2.5 V...ault)
B	Input	PIN_K2	2	B2_N0	2.5 V...ault)
clk	Input	PIN_H4	2	B2_N0	2.5 V...ault)
LED	Output	PIN_L12	5	B5_N0	2.5 V...ault)

Figure 1: Signal mapping of Part 1 logic combination

Observe the output signal for some inputs and comment.

Part II

You can notice that the signal state is not altering at the output. The clock frequency may be too fast for you to see anything. The LED is probably on but not solid on, which indicates a high-speed clock. An external oscillator drives the MAX10 device that we are using with a clock frequency of 4MHz on pin PIN_H4 (see MXDBManual.pdf page 21 on Canvas). If a 4MHz clock drives an LED on the MXDB board, the changes on the LED will not be viewable as transitions will happen faster than the human eye can see.

In the second part of this design, we want to use a much slower clock and observe the changes at the output. We will therefore use a module called clock divider to slow down the clock.

The schematic of the resulting circuit is shown in the following figure (*Figure 2*).

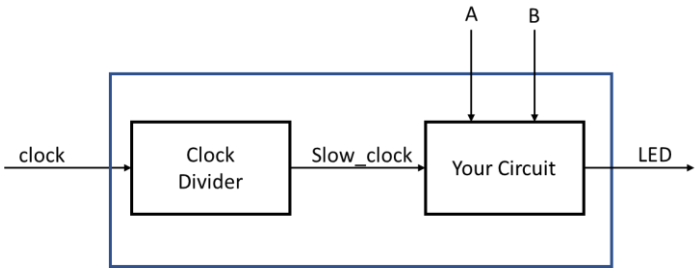

Figure 2: Synchronous circuit with clock divider

The VDHL code of the overall circuit is provided below in figure Figure 4. We use a process clock divider to generate a slower clock. The clock divider is a process that takes as input the base clock and generates a slower clock.

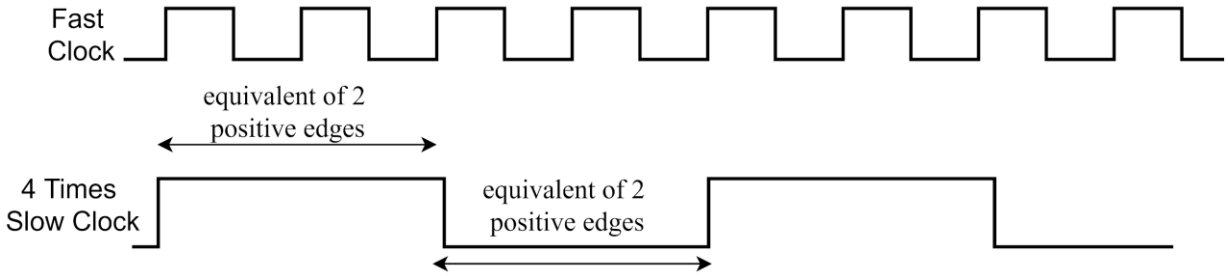

Figure 3: Fast and slow clock

The clock divider uses a binary counter to count the number of main clock pulses. Upon reaching the dividing value, the slower clock signal is generated. *Figure 3* illustrates the behavior of the

clock divider that slows down the base clock by 4, i.e., each slow clock occurs after four base clock cycles.

In VHDL coding, two ways are followed to detect the clock signal's rising edge/falling edge.

1. **if(rising_edge(**clock**)) then**
2. **if(**clock **=** '1' **and** clock'**event**) **then**

After detecting a rising edge/falling edge on the base clock cycle, we can start an internal counter and count till our desired clock cycles of slower clock. After finishing the counting, we enable the slower clock. For example, if we have a base clock of 4 MHz (4 million clock cycles per second) and want to generate a clock of 1 Hz (1 clock cycle per second), we must count 4 million clock cycles to achieve a one Hz clock. Remember, every clock cycle has one positive cycle (rising edge) and one negative cycle (falling edge). Thus, 4 MHz (4 million clock cycles per second) clock signal has 4 million positive and 4 million negative cycles.

It is very common to make mistakes in counting the desired clock cycles for the smaller clock. Often students count 4 million positive edges of the base clock to get desired 1 Hz clock, which is incorrect. As stated above, if we want to generate a 1 Hz clock (1 clock cycle per second), we must count for 4 million cycles. But in VHDL implementation, we count for 2 million. Because half of the slower clock period is positive edge, and the rest of the half period is negative. After counting 2 million edges, we force the slower clock to change its current state as we have already counted the half period of the slower clock.
One easy way of counting the required number is to follow this equation:

$$\frac{\frac{Base\ clock\ Frequency}{2}}{Slower\ Clock\ \ Frequency}$$

If we take a 50 MHz base clock and wants to generate a 50 Hz slower clock, the counting number will be,

$$\frac{\frac{50 * 10^6}{2}}{50} = 500{,}000\ count$$

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.round;
use IEEE.std_logic_unsigned.all;

entity Lab3 is
port ( A,B , clk,reset: in std_logic;


LED : out std_logic;
--seg_left : out std_logic_vector(7 downto 0);
--seg_right : out std_logic_vector(7 downto 0)

);

end Lab3;

architecture bhv of Lab3 is

  signal count : std_logic_vector(31 downto 0) := (others =>'0') ;
  signal tmp : std_logic := '0';
begin

  process(clk)
  begin
  if(rising_edge(clk)) then

   if(unsigned(count) = to_unsigned(2000000,32)) then
      tmp <= not tmp;
      count <= (others =>'0');


   else
      count <= count + 1;
   end if;

  end if;
  end process;
  LED <= tmp;




end bhv;

```

Figure 4: VHDL Code 1 - LED blinker using clock divider

Understand the code in Figure 4 and verify that the VHDL code matches the behavior on the picture. The code has been written for 1 Hz slower clock when the base is 4MHz clock and the count for 1 Hz is 2,000,000.

One of the most used data types in FPGA coding is **std_logic** and **std_logic_vector**. While std_logic represents the bit 0 and 1 and a **std_logic_vector** is an ordered collection of **std_logic** elements. The right most position of the vector represents the least significant bit and next to the leftmost position represents the most significant bit. Conversion of **std_logic_vector** is often considered tedious. In the provided code, a very effective comparing of **std_logic_vector** were given which uses the **uncount** and **to_unsigned** function (Figure 4 Line 29). Students are advised to observe the data type comparison very carefully. To exploit the effectiveness of these two functions, the provided library headers must be included in the code.

Pre-Lab Homework (10 pts)

1- Calculate the value of the count if we want a 5 sec clock period if we use the 4 MHz on board clock as a base clock. **(5 pts)**
2- Modify the VHDL code of *Figure 4* below to include your previous design, according to *Figure 2*. Compile and make sure your code is free of errors. **(5 pts)**

In-Lab Implementation (5 pts)

Use the pin mapping from *Figure* 1 to implement and test your 5 sec clock design on the FPGA board. **(5 pts)**

Document your observation.

Use the schematic viewer (**Tools-> Netlist Viewers->RTL Viewer**) to visualize the structure of the circuit.

Problem 2: **Implementation of a 4-bit Binary to BCD converter with a seven-segment display on the MAX10 (10 Pts)**

Goal: In this experiment, you will learn about binary to BCD conversion and use the seven-segment display on the MXDB board to visually verify the correctness of your converter.

Description: BCD is a binary coded decimal number, where its equivalent binary number represents each digit of a decimal number. As an example,

Table 1 shows the conversion of binary and hexadecimal to BCD. You must show the BCD values for different binary numbers into the seven-segment display.

HEX	Binary	BCD
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9

Table 1 BCD conversion table

The configuration of the seven-segment display is shown in Table 2. Here we can assign different values to **a through g** (see Figure 7 LED Labels) to display the BCD numbers. Applying a logic '1' to a particular segment connection (a-g), the appropriate segment will light up in the enabled displays.

As shown in *Figure 5* if we want to light up c, we will assign logic '1' to c and the other inputs (a, b, d, e, f, g) to logic '0'. The proper configuration will turn on c (*c is turned to red on the right side of Figure 5*).

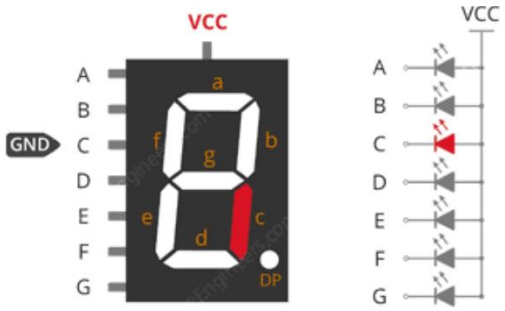

Figure 5: Enabling seven-segment display

Now you can apply this information to show BCD numbers in the seven-segment display. Referring to *Table 1*, you have to encode 4-bit binary carefully. For example, if we want to show "*4"* in the display, then the a-g signal needs "01100110".

The MXDB board has two seven segment LED displays highlighted below. The left display is U8 and right is U7.

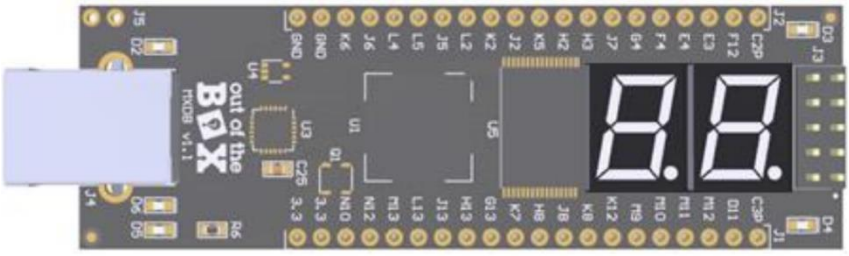

Figure 6 MXDB Seven Segment Display

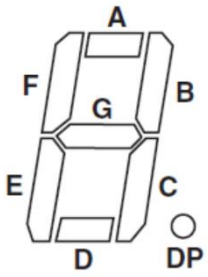

Figure 7 LED Labels

The display segment names are shown in the Figure 7 and related pin number is summarized in Table 2.

Table 2 Display LED Pins

Segment Signal (U8 Left)	Pins	Segment Signal (U8 Right)	Pins
A	H10	A	D9
B	F8	B	C9
C	E8	C	G12
D	J12	D	L10
E	H9	E	K10
F	J10	F	J9
G	L11	G	E6
DP	E9	DP	E12

Create a new process and put the **tmp** (slower clock) in the sensitivity list. This process will be executed when any change in the **tmp** signal occurs. Declare a **std_logic_vector** data type of width (4 bit) and name it as a **count_2**. Use the **count_2** data type as a counter and set the limit to 9. Change the values of the seven-segment LED according to the counter value. An easy way of assigning the LEDs is the use of a when statement. When statement in VHDL should be placed outside of the process block. The code of

this implementation is given below in Figure 8. This implementation will be an example of a 4 bit BCD counter, which will count from 0 to 9 and displaying in the seven segment display.

```
   ----------------------------------------------------
   --- Slower clock of 1 Hz
   ----------------------------------------------------
   process(tmp)

   begin


   if(rising_edge(tmp)) then

      if(count_2 = "1001") then

         count_2<= (others =>'0');
      else

         count_2 <= count_2 + 1;
      end if;
   end if;


   end process;

   ----------------------------------------------------
   ----------------------------------------------------
   -- when statement in vhdl is concurrent and should be placed outside the process block
   ----------------------------------------------------
   seg_left <= "00000000" when (count_2 = "0000") else -- display 0
               "01100000" when (count_2 = "0001") else -- display 1
               "11011010" when (count_2 = "0010") else -- display 2
               "11110010" when (count_2 = "0011") else -- display 3
               "01100110" when (count_2 = "0100") else -- display 4
               "10110110" when (count_2 = "0101") else -- display 5
               "10111110" when (count_2 = "0110") else -- display 6
               "11100000" when (count_2 = "0111") else -- display 7
               "11111110" when (count_2 = "1000") else  -- display 8
               "11100110"  -- display 9;

end bhv;
```

Figure 8 BCD Implementation with 7 segment display

Pre-Lab Homework (4 pts)

1. Complete the VHDL code to represent the numbers 0-9 periodically in the display after every one second. **(4 pts)**

In-Lab Implementation (6 pts)

1. Using Quartus, assign pins to each of the inputs/outputs such that the signals are connected to the appropriate locations on the board. **(2 pts)**

2. Download your design to the board and test it. Upload the video of the implementation with the report. **(4 pts)**

Problem 2 (Bonus): Washing Machine Controller with logic gates (12 Pts)

Goal: This part of the lab is about designing a combinational circuit that implements a subset of the functionalities commonly found in washing machines. The design flow includes the manual specification of the circuit using logic gates. Then we perform circuit minimization and implementation on our MAX10 board.

Description: Assume we are designing a washing machine controller. The washing machine's motor turns ON when it has the right temperature, the correct water level, and obviously when the machine's door is closed. In addition, you have a heater and a water valve. If the water level goes down while the door is closed, we should open the valve to maintain the water level. Adding to this, you need to turn ON the heater if the temperature is low, the door is closed, and the machine has the correct water level.

Pre-Lab Homework (8 pts)

1. Write a truth table and the combinational logic to represent this problem and optimize to get the minimum amount of logic. **(4 pts)**
2. Implement the logic in VHDL without any error. **(4 pts)**

In-Lab Implementation (4 pts)

1. Using Quartus, assign pins declared I/O signals that the signals are connected to the appropriate locations on the board. **(2 pts)**
2. Download your design to the board and test it for different inputs and outputs. The TA will ask you to demonstrate at least one example for each possible selection. **(2 pts)**

Deliverables: Prepare and submit your report in canvas, using the template provided on the webpage. Include supporting documents (pictures, video, or compressed sources after cleaning up your project) in your report.

N.B All Pre-Lab HomeWorks must be completed before your lab session. You will not be admitted to the lab without completing this part.