LAB #6

Follow the steps to reproduce the sequential circuit using advanced VHDL programming techniques

Objective

- To understand the working of clock in FPGA design
- To generate slow clocks from a given clock
- To understand the function of a RESET signal in digital circuits.
- To design an electronic wheel of the fortune, utilizing the package and procedure constructs

Pre-Lab

Clocks and reset signals in digital circuits

a) Clock signal

A process statement easily handles synchronous logic, by allowing us to specify a clock signal as one of the triggers to the circuit. Figure 1 shows a circuit that is sensitive only to changes in the clock signal.

```
ARCHITECTURE cct OF testing IS BEGIN
process (clk)
begin
...
end process;
END cct;
```

Figure 1: Code Fragment: A Clock-Sensitive Circuit

Let's think about what a clock signal, or any waveform for that matter, looks like. There will be a *rising edge* and a *falling edge*. Do we want our circuit to be triggered by any change in the clock signal, or *{positive1negative} edge-triggered*? How do we represent edge- triggering in VHDL?

Edge-triggering requires two conditions to be true:

1) the clock signal must change
2) it must change in the positive (negative) direction, as required.

Representing edge-triggering in VHDL therefore requires a compound statement, specifying each of these conditions.

To represent the change in a clock signal, we need some way to record or recognize that an event (corresponding to the change in clock value) has occurred. To do this, we consider the event attribute of the clock signal, given by clk'event .

Unfortunately, all that the clk'event attribute tells us is that we have encountered an edge in the clock signal, and not whether it was a rising or falling edge. To further specify the type of edge encountered, we also specify the value of the clock after the edge has "completed". Thus, a value of clk='1' would indicate that a rising edge had just occurred, and a value of clk='0' would indicate that a falling edge had just occurred.

Figure 2 shows the clock-sensitive circuit of Figure 1 re-written to specify a negative-edge triggered circuit.

```
ARCHITECTURE cct OF testing IS BEGIN
    process (clk) begin
        if (clk event and clk= 0 ) then
            ...
        end if; end process;
    END cct;
```

Figure 2: Code Fragment: A Clocked, Negative-Edge Triggered Circuit

A typical digital circuit usually have more than one clock in a circuit. These clocks are generated using the standard clock available in the circuit. The Altera DE2-115 FPGA board includes one oscillator that produces 50 MHz clock signal. A clock buffer is used to distribute 50 MHz clock signal with low jitter to FPGA. The distributing clock signals are connected to the FPGA that are used for clocking the user logic. A typical clock waveform is shown in Figure 3

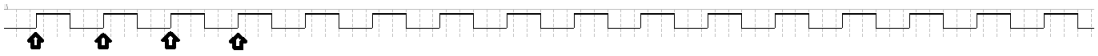

Figure 3: Clock waveform

The small arrows in Figure 3 indicates the positive edge of original clock. To generate slow clocks from original clock, we will use the analogy of a counter. At every rising-edge, the counter will increment, see the code in Figure 4.

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

entity slowclocks is port(
    clk: in std_logic;
    sclk: out std_logic
); end slowclocks;

architecture behv of slowclocks is
    signal count: std_logic_vector(31 downto 0) := X"00000000";

begin

    process(clk)

    begin

        if (clk'event and clk='1') then
            count <= count + '1';
        end if;

    end process;

    sclk <= count(20);

end behv;
```

Figure 4: Code for generating slow clocks

The key thing to note here is that the bit count (1) is half of the input clock frequency (25 MHz), count (2) will be 12.5MHz and so on. If we want to generate a fast clock then the bit value of count should be closer to zero and for slow clocks, it should be closer to bit number 31.

In-Lab Tasks

Lab Task 1: LED blinking

For our first lab task, you need to design a system that blinks 4 LEDs depending upon 3 available frequencies. We will use 3 slide switches to control the blinking frequency of LEDs. LEDs will blink according to the table given below

Selection Switches			**Blinking frequency of LEDs**
SW2	SW1	SW0	
OFF	OFF	OFF	Sclk <= count (20)
ON	X	X	Sclk <= count(31)
OFF	ON	X	Sclk <= count(15)
OFF	OFF	ON	Sclk <= count(2)

Please remember that all LEDs will turn ON and OFF together. Clk signal in our entity will use the clock available on our FPGA board, the pin number is PIN_Y2

b) Reset Signal

Resets are a generally very useful thing. A reset signal in a circuit can be used to restore initial conditions, such as resetting a counter to its initial count value. The problem with resets is that they are generally asynchronous signals. How do we work a reset into asynchronous circuit?

This is actually quite easy if we remember a couple of things about VHDL. The first is that we can specify in the process sensitivity a list of all signals that affect the circuit outputs. So, we can specify a sensitivity list that includes both the (synchronous) clock signal and the (asynchronous) reset signal.

The second thing that we must remember that in the specification of a sequential statement such as an if-then-else statement, there is an order of preference that is followed. The first conditions that are encountered are of higher precedence, even if subsequent conditions are also true. So, as long as the reset conditions are tested first, we should be okay.

Figure 5 shows how to incorporate an asynchronous reset signal into a synchronous circuit.

```
ARCHITECTURE cct OF testing IS BEGIN
    process (clk, reset)
    begin
        if (reset = 1 ) then
            -- do (asynch) reset actions
        elsif rising_edge(clk) then
            -- do (synch) rising edge clock triggered actions
        end if;
    end process;
END cct;
```

Figure 5: Code Fragment: Asynchronous Resets in Synchronous Circuits

The sensitivity list tells us that the circuit outputs are sensitive to changes in the ***clk*** signal and the ***reset*** signal. The first condition included in the if statement concerns the reset signal; if

this (asynchronous) signal is set, then the reset actions will be executed. If the (asynchronous) reset signal is not set, and the rising edge of the clock has occurred (note that rising edge == (clk'event and clk='1')) then the synchronous rising edge triggered actions occur.

Lab Task 2: Implementation of an LED wheel-of-fortune using a package/procedure

The LED wheel of fortune you are designing works as follows.

At the startup, the LED lights continuously rotate around using a bit pattern “00000001”, and at the same time the seven-segment display (SSD) digits continuously increments the four-digit hex number (16 bit) at a fast speed. The digit starts from 0000 and increments to FFFF and repeats.
When a user presses a button, the LED rotation stops, and the SSD displays the number caught at the positive transition of the button press. When the user presses the button again, it starts the whole process again, i.e., the LED lights rotates around, and SSD increments the four-digit hex number at a fast speed. The LED movements should be slow enough to be visible. However, the numbers on the SSD should be fast enough so that it is hard to recognize except when it is stopped.
In order to accomplish this, several things must happen.

1) Incorporate a reset signal in the design
2) Make a procedure/function for hex to ssd that can display 0-9 plus A-F on 7-segment display.
3) You will need to generate a fast clock ($1/60^{th}$ seconds) for SSDs and slow clock for LEDs
4) Rotate bits and display the bits on LED. Synchronize the LED pattern to a slow clock, so that the rotation would be visible. Rotating LED can be done using concatenation.
5) Write a push button code to detect positive transition and to take a proper action.
6) Please note that this design can be easily turned into a stopwatch or a timer.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date:_____