CpE Lab

Section 002

Lab 8: Introduction to Sequential Logic-I

Prepared By:

Olumuyiwa Oyeleye

Prepared For:

Cletis Nicklow

October 20, 2010

**Introduction**

The main purpose of this lab is to introduce students to sequential logic, a type of logic circuit whose output depends not only on the present input but also on the history of the input. This is in contrast to combinational logic, whose output is a function of, and only of, the present input.

**Experiment**

Part 1: Clock Divider

The goal of this part of the lab is to slow down the internal clock of the Altera DE2 board which runs at 50 MHz, by dividing the clock signal

**Methodology**

The VHDL code to make this happen was provided in the lab hand-out, so the team created a new VHDL project and simply copied the provided code unto the VHDL editor. The code defined a clock divider which had 1-input and 1-output. The code was compiled and the output (clock\_out) of the clock divider was connected to pin AE22, which is the first LED on the board, and the input (clock\_in) was connected to pin N2 which is the internal 50 MHz clock of the board.The VHDL code for the clock divider can be seen below.

VHDL code

**LIBRARY ieee;**

**USE ieee.std\_logic\_1164.all;**

**entity slow\_clock is**

**PORT(**

**clock\_in : in std\_logic;**

**clock\_out : out std\_logic**

**);**

**end slow\_clock;**

**architecture behavior of slow\_clock is**

**signal clock\_tmp : std\_logic;**

**begin**

**process(clock\_in)**

**variable x : integer := 0;**

**begin**

**if(clock\_in'event and clock\_in='1') then**

**x:=x+1;**

**if x = 50000000 then**

**x:=0;**

**clock\_tmp <= not clock\_tmp;**

**clock\_out <= clock\_tmp;**

**end if;**

**end if;**

**end process;**

**end behavior;**

The code was re-compiled and programmed unto the FPGA.

**Result**

The value of the input clock\_in was changing with a frequency of 50 MHz and variable ‘x’ (whose initial value was 0) served as a counter for every time the value of clock\_in was. The counter reset itself to 0 once it had counted up to 50000000 (this represented one cycle) and the value of clock\_out changed after each cycle. **In short the group was dividing the internal clock by 50000000.**

**Part II – Building a D Flip Flop**

The goal of this part of the lab was to build a ’D Flip-Flop’ with inputs ‘clock’ and ‘d’, and outputs ‘q’ and ‘q not’, where the flip flop would set the output ‘q’ equal to the input ‘d’ each time a clock cycle was completed.

**Methodology**

To achieve this, the group created a new VHDL file for the design of the D Flip Flop. The aforementioned inputs and outputs were defined and just as requested in the problem statement, the group set the output ‘q’ equal to the input ‘d’ after the completion of a clock cycle and set output ‘q not’ to ‘not d’ . This was done by using an ‘if’ statement. This ‘if’ statement is shown below in the VHDL code for the flip flop.

(Note : Instead of “q not” the group used the name ‘qn’.)

VHDL Code for ‘D Flip Flop’

**LIBRARY ieee;**

**USE ieee.std\_logic\_1164.all;**

**entity flipflop is**

**PORT(**

**clock : in std\_logic;**

**d : in std\_logic;**

**q : out std\_logic;**

**qn : out std\_logic**

**);**

**end flipflop;**

**architecture behavior of flipflop is**

**begin**

**process(clock)**

**begin**

**if(clock'event and clock='1') then**

**q <= d;**

**qn <= not d;**

**end if;**

**end process;**

**end behavior;**

The code was compiled, and input ‘clock’ was connected to pin G26 (a pushbutton), input d to a switch while the outputs q and qn were connected to LEDs. The code was re-compiled and programmed unto the FPGA.

**Result**

Different input combinations were tested, and for each one the D Flip Flop worked as it was supposed to. When the value of input ‘clock’ was set to 1 and the switch corresponding to the input ‘d’ was flicked on, the LED for output ‘q’ expectedly lit up. This was just one of the combinations inspected.

**Part III- Building a slow D Flip Flop**

The objective this time was to create a slow D Flip Flop.

**Methodology**

This was done by combining the clock divider from part I and the D Flip Flop from part II in a new VHDL file. The inputs and outputs for the slow D Flip Flop were the same as the ones for the D Flip Flop, and the two signals defined were ‘clock\_tmp’ and ‘clock\_out’. The signal ‘clock out’ was defined as the output from the clock divider, and it served as the clock input to the D Flip Flop. Just as in part II, the signal ‘clock out’ had to have a value of 1 in order for output ‘q’ to equal input‘d’. Once again this was done by using an ‘if’ statement. The VHDL code for the slow D Flip Flop is shown below.

VHDL Code:

**LIBRARY ieee;**

**USE ieee.std\_logic\_1164.all;**

**entity slowff is**

**PORT(**

**clock\_in : in std\_logic;**

**d\_in : in std\_logic;**

**q : out std\_logic;**

**qn : out std\_logic**

**);**

**end slowff;**

**architecture behavior of slowff is**

**signal clock\_tmp : std\_logic;**

**signal clock\_out : std\_logic;**

**begin**

**process(clock\_in)**

**variable x : integer := 0;**

**begin**

**if(clock\_in'event and clock\_in='1') then**

**x:=x+1;**

**if x = 50000000 then**

**x:=0;**

**clock\_tmp <= not clock\_tmp;**

**clock\_out <= clock\_tmp;**

**if (clock\_out='1') then**

**q <= d\_in;**

**qn <= not d\_in;**

**end if;**

**end if;**

**end if;**

**end process;**

**end behavior;**

The code was compiled, and input ‘clock’ was connected to pin N2, input ‘d’ to a switch while the outputs ‘q’ and ‘qn’ were connected to LEDs. The code was re-compiled and programmed unto the FPGA.

**Result**

The slow D Flip Flop was tested and it worked as expected as the action of the D Flip Flop was substantially slowed down.

**Part IV**

The goal of this part of the lab was to design a simple up down modulo 8 counter which counts up to 7 and repeats forever.

**Methodology**

The first step taken was to build a state table. This was easily done by adding the number 1 to the present state value when counting up and 0 when counting down. Next the group built an excitation table which was the same as the state table except that the numbers were now represented in 3-bit binary numbers, which each bit representing an input. After completing the excitation table, K-map was used to derive the equations that would govern the circuit to be built. The values used in the K-map were gotten from the excitation table.

**Result**



Figure 1 State table



Figure 2 Excitation table

The K-Maps and their corresponding Equations:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X|Q2 | | | |
| Q1|Q0 | 0|0 | 0|1 | 1|1 | 1|0 |
| 0|0 | 1 | 1 | 1 | 1 |
| 0|1 | 0 | 0 | 0 | 0 |
| 1|1 | 0 | 0 | 0 | 0 |
| 1|0 | 1 | 1 | 1 | 1 |

D0=> !Q0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X|Q2 | | | |
| Q1|Q0 | 0|0 | 0|1 | 1|1 | 1|0 |
| 0|0 | 0 | 0 | 1 | 1 |
| 0|1 | 1 | 1 | 0 | 0 |
| 1|1 | 0 | 0 | 1 | 1 |
| 1|0 | 1 | 1 | 0 | 0 |

D1 => (x!Q1!Q0)+(!x!Q1Q0)+(xQ1Q0)+(!xQ1Q0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X|Q2 | | | |
| Q1|Q0 | 0|0 | 0|1 | 1|1 | 1|0 |
| 0|0 | 0 | 1 | 0 | 1 |
| 0|1 | 0 | 1 | 1 | 0 |
| 1|1 | 1 | 0 | 1 | 0 |
| 1|0 | 1 | 1 | 1 | 0 |

D2 => (!x!Q1Q2)+(xQ0Q2)+(!Q0Q1Q2)+(x!Q0!Q1!Q2)+(!xQ0Q1!Q2)

**Pre/Post lab Question**

LIBRARY ieee;  
USE ieee.std\_logic\_1164.all;  
  
ENTITY 7-LED IS  
  
PORT (  
    W : IN STD\_LOGIC;  
    X : IN STD\_LOGIC;  
    Y : IN STD\_LOGIC; Z: IN STD\_LOGIC;  
    a : OUT STD\_LOGIC;  
    b : OUT STD\_LOGIC;  
    c: OUT STD\_LOGIC;  
    d : OUT STD\_LOGIC;  
    e : OUT STD\_LOGIC;  
    f : OUT STD\_LOGIC;  
    g : OUT STD\_LOGIC;  
  
);  
END 7-LED;  
      
ARCHITECTURE Behavior of 7-LED IS  
    begin  
        a <= (not W and not X and not Y and Z) or (not W and X and not Y and not Z) or (W and not X and Y and Z) or (W and X and not Y and Z);  
        b <= (not W and not X and not Y and Z) or (not W and X and not Y and Z) or (not W and X and Y and not Z) or (W and not X and Y and Z) or ( and X and not Y and not Z) or (W and X and not Y and not Z) or (W and X and Y and Z);  
        c <= (not W and not X and not Y and Z) or (not W and not X and Y and not Z) or (W and X and not Y and not Z) or (W and X and Y and Z);  
        d <= (not W and not X and not Y and Z) or (not W and X and not Y and not Z) or (not W and X and Y and Z) or (W and not X and not Y and Z) or (W and not X and Y and not Z) or (W and X and Y and not Z) or (W and X and Y and Z);  
        e <= (not W and not X and Y and Z) or ( not W and X and not Y and not Z) or (not W and X and not Y and Z) or (not W and X and Y and Z) or (W and not X and not X and Z);  
        f <= (not W and not X and Y and not Z) or (not W and not X and not Y and Z) or (not W and X and Y and Z) or (W and X and not Y and not Z);  
        g <= (not W and not X and not Y and not Z) or (not W and not X and not Y and Z) or (not W and X and Y and Z) or (W and X and not Y and not Z);  
  
END Behavior;

**Conclusion**

Considering the fact that this is just the first part in a two part lab about the introduction to sequential logics I cannot say much about the topic, but I did learn how to program a D Flip Flop and a slow D Flip Flop, and also how divide the internal clock in a FPGA.