TALLINN UNIVERSITY OF TECHNOLOGY

COURSE CODE: **IAS0600**

(DIGITAL SYSTEMS DESIGN WITH VHDL)

LABORATORY REPORT NO 3

(TOPIC: COUNTERS**)**

Student: Dismas EZECHUKWU (**184603IVEM**)

Instructors:

Alexander Sudnitson (Associate Professor)

Dmitri Mihhailov (Research Scientist)

DATE: 11/11/2018

1. INTRODUCTION

In digital logic and computing, a counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.[1]. In this laboratory work the event to be monitored is a push button. Hence, we will monitor the number of times which a pushbutton has been pressed and display it in binary form with LEDs (Light Emitting Diodes) present on the Nexys-4 FPGA board provided for the lab.

Due to contact bouncing of the pushbutton, the counter may increment more than once whenever the pushbutton is pressed only once. To overcome this, we will add a debouncing circuit to the design, simulate and implement as well on the Nexys-4 FPGA board provided.

The overall task is presented in the TASK section below in two steps.

1.2 TASK

- Implement a counter that is enabled for one clock cycle each time a pushbutton is pressed. Note, that at this step the counter may increment more than once after a single button press due to bouncing effect;

- Add debouncer for the employed pushbutton. Then the counter should increment only once after a single button press since bouncing effect is eliminated [2].

2. BACKGROUND

The counter in this case will be a for bit counter; it will be able to display the counted event (pushbutton) number on the Nexys-4 FPGA board in binary representation. This implies that the maximum number of presses to be displayed is 16 or F in decimal.

To detect the button press, we will employ a button detector circuitry; what this does is to detect the transition from LOW to HIGH of the pushbutton signal. A short description of this is shown below:

Edge Detection and Push Button Debounce Circuit

The most straightforward way to detect a button press is to detect a transition when button input goes from LOW (unpressed) to HIGH (pressed). This can be done by sampling the button input and storing the last two values. Whenever these two values are different, it is possible to conclude that there has been a transition in the button input signal. An example in Listing 1 shows a rising edge detector that generates a single one clock period long pulse when the button signal changes from LOW to HIGH. A falling edge detector (that shows

when the button is released) can be implemented in similar fashion by changing the button_pulse signal equation to detect button_buf1 = ’0’ and button_buf2 = ’1’ condition [2].

Listing 1. Rising Edge Single Pulse Generation

```
button_buf1 <= button_input_signal when rising_edge(clock_100MHz);

button_buf2 <= button_buf1 when rising_edge(clock_100MHz);

button_pulse <= button_buf1 and not button_buf2;
```

3. WORKFLOW

Describe the chosen method/approach for solving the task and the way it has been applied

The method chosen to do this task are of two types;

- Counter design
- Counter design with debouncing.

For the first past (That is, the counter design);

It involves the creation of the project in vivado, simulation and testing on the FPGA board.

- Specific parts in the VHDL code include declaring of the ports and signals, edge detection and then counting of the button. The Figure 1. below shows a section of the code.

```
--button detection
button_buf1 <= button when rising_edge(clock_100MHz);
button_buf2 <= button_buf1 when rising_edge(clock_100MHz);
button_pulse <= button_buf1 and not button_buf2;
```

Figure 1. Rising Edge Single Pulse Generation

For the debounced counter, the input button was properly filtered and all bouncing effect were removed using some delays in the form of a 4-bits register, snapshots of the codes sections is shown below;

```
27      -- clock division
28      process (clock_100MHz)
29      begin
30         if clock_100MHz'event and clock_100MHz = '1' then
31            if counter < 100 then
32               counter <= counter + 1;
33               clock_1MHz <= '0';
34            else
35               counter <= 0;
36               clock_1MHz <= '1';
37            end if;
38         end if;
39      end process;
```

Figure 2. Code for clock division

The clock division here was used to pass the pushbuttons input to the different delays used to filter out the bouncing effect.

```
41      button_debouncing:process(clock_100MHz)
42      begin
43        if reset ='1' then
44           delay1 <= '0';
45           delay2 <= '0';
46           delay3 <= '0';
47           delay4 <= '0';
48        elsif clock_100MHz'event and clock_100MHz = '1' then
49          if clock_1MHz = '1' then
50           delay1<=button;
51           delay2<=delay1;
52           delay3<=delay2;
53           delay4<=delay3;
54          end if;
55        end if;
56      end process;
57
```

Figure 3. Button debouncing with delays.

The Test Bench File

The purposes of the testbench file is for exercising and verifying the functional correctness of my counter model (including the debounced counter). Its structure is described below;

```
46  button: process
47  begin
48  --button off
49  button_tb <= '0';
50  wait for 5 ms;
51
52  --adding noise
53  button_tb <= '0';
54  wait for 1 ms;
55  button_tb <= '1';
56  wait for 1 us;
57  button_tb <= '0';
58  wait for 3 us;
59  button_tb <= '1';
60  wait for 1 us;
61  button_tb <= '0';
62  wait for 2 us;
63  button_tb <= '1';
64  wait for 1 us;
65
```

Figure 4. Testbench for the noise (bounce) signal generation

```
38  clock : process
39  begin
40  clock_100MHz_tb <= '0';
41  wait for 10 ns;
42  clock_100MHz_tb <= '1';
43  wait for 10 ns;
44  end process;
```

Figure 5. Testbench for the clock signal generation

```
125  reset_proc: process
126  begin
127  reset_tb <= '1';
128  wait for 30 ns;
129  reset_tb <= '0';
130  wait for 50 ms;
131  wait;
132  end process;
133  end Behavioral;
134
```

Figure 6. Testbench for the reset signal generation

4. RESULTS AND DISCUSSION

The simulation results from the counters are presented below;

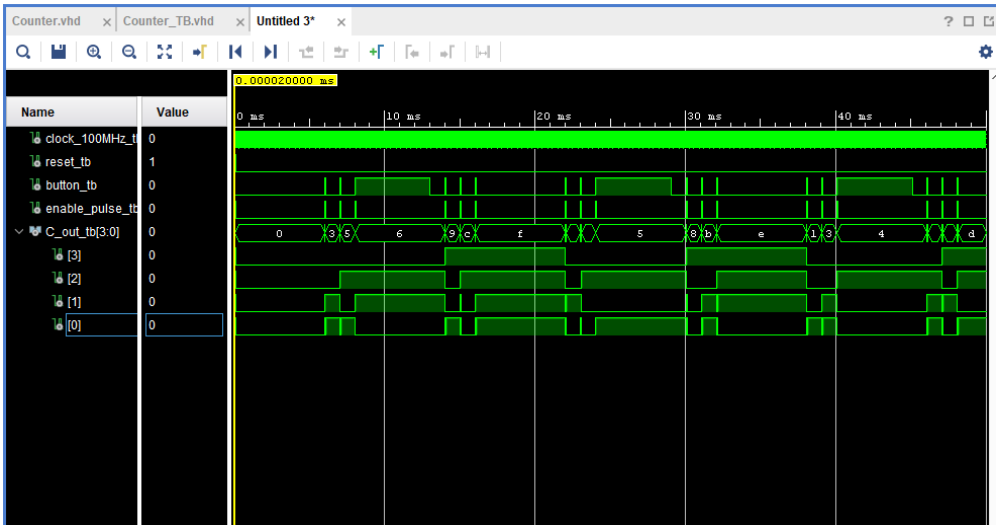

Figure 7. Simulation result for the counter without debouncing

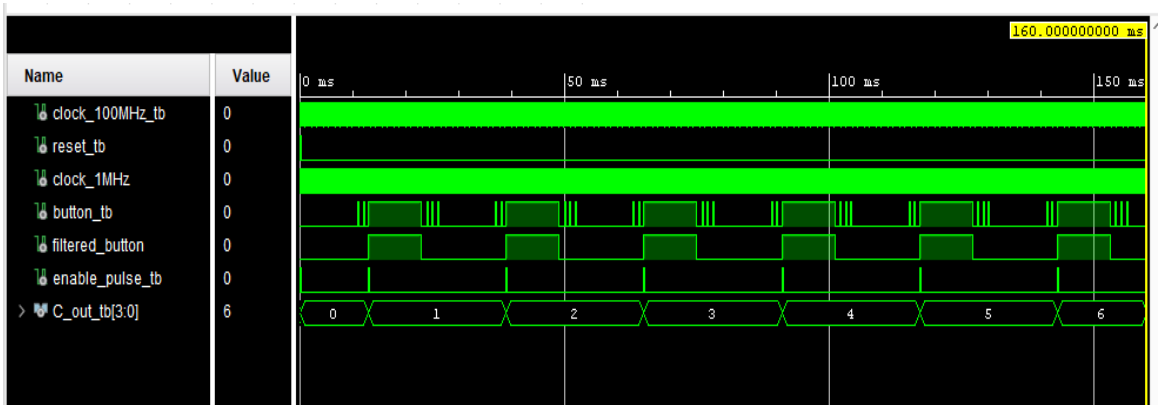

Figure 8. Simulation result for the counter with debouncing

The simulation for the counter without debounce shows that the enable pulse counted several times due to bouncing effect and hence, giving a wrong counting sequence to the output. While for the debounced counter, its simulation filtered out the noise signals and hence, gave a correct result.

5. CONCLUSION

In this project, Counter with and without debouncing effect to the input button have been developed, simulated and implemented on the FPGA board provided. The project required the use of clock divider to slow down the rate at which the shifting delays work in order to remove bouncing effect.

The following observations have also been observed in the simulated resulted;

It was seen that in Figure 7. The counter counted the noise present in the button input as an actual press and hence, this resulted in an erroneous counting by the counter.

In the Figure 8. It was seen that the effect of the bouncing what was removed and the filtered button is clean (without bouncing), making the output to be correct, incrementing only when an actual button was pressed and not responding to noise.

6. References

[1] https://en.wikipedia.org/wiki/Counter_(digital)

[2] http://ati.ttu.ee/~alsu/DE_Counter.pdf