EGR680 High Level Implementation on FPGA

Laboratory 02

Seven-Segment Display Applications Using PYNQ

Professor: Dr. C. Parikh

Student: Dimitri Häring

September 12, 2018

Contents

1 Introduction

The goal of the lab 2 is to familiarize the student with Vivado in further depth and to introduce and teach top level hierarchy. The lab is structured the way that the students learns the design flow to realize a top down model with less instructions for each part.

2 Design

In this section the design and decisions that where made to achieve the laboratory are discussed.

2.1 Part I - Behavioral Modeling of a Seven-Segment Display

Verilog is used to describe a decoder that shall control a seven-segment display with two switches and one button. The given task is as followed:

In this part, you will simulate a pattern decoder using the buttons, switches, and a single seven-segment display. Using switches, SW0 and SW1 as the pattern input, write a Verilog program that takes the binary input combinations of the switches and displays the decimal value of the binary switch combination on the seven-segment display. Table 1 shows the switch combinations with the expected output value on the seven-segment display.

Table 1: Input Combinations and Expected Output

SW1	SW0	BTN0	Display
0	0	0	OFF
0	0	1	0
0	1	1	1
1	0	1	2
1	1	1	3

As software package to implement the decoder in Verilog Vivado 2017.2 is used.

2.2 Part II - Hardware Implementation & Modular Design

For the second part a hierarchical design is achieved including the implementation and generation of a bit stream to use hardware or more specific the PYNQ development board. Therfore the folowing task is given:

In this part, you will modify your code from Part I. Instead of only using one button and one switch, you will design your Verilog code to display on four seven-segment displays by pressing the four buttons on the PYNQ board. Each seven-segment display will correspond to one button (i.e., BTN0 lights up the right-most seven-segment display and BTN1 will light up the adjacent seven-segment display and so on). Make sure that when a button is released, the corresponding seven-segment display turns OFF. Only one seven-segment display should be on at any given time.

To realize the constrained file the PYNQ reference manual was used to allocate the right pins for the switches and buttons, figure 1 shows the basic I/O pins assignment of the PYNQ development board.

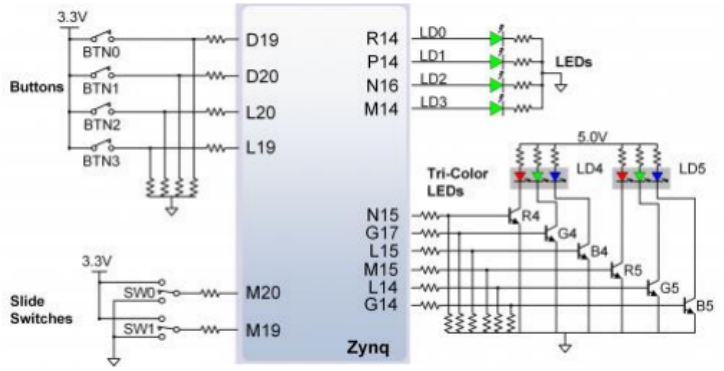

Figure 1: Schematic PYNQ Basic I/O. [1]

The one of solution to part II is two concatenated case statements. The first case is used to check for the button press and the second to switch the seven-segment display to show the number as defined by the two switches SW0 and SW1. The used code is listed in the following three listening **??**, **??**, and **??**.

2.3 Part III - Laboratory Exercise

The third part of the lab combines part one and part two to a final design that uses a modified decoder version of part two and a clock divider to print out the words, PYNQ, FPGA, IS, and COOL with the seven-segment display. The clock divider is used to slow down the 125 MHz clock to 125 Hz, the relation ship of clock divider an decoder is shown in figure 2. Switches SW0 and SW1 defines which word will be shown on the displays if the button BTN0 is pressed, as shown in figure 3. The task is described as followed in the lab report:

In this part, you will create a time multiplexed seven-segment decoder using the block diagram shown in Figure 8 as a guide. You will need to create a multiplexer in order to display more than one seven-segment simultaneously. Design the multiplexer to enable each digit for 8ms, or a 125Hz refresh rate. Remember, these Pmods have a shared digit select pin. Most multi-digit displays would have a separate digit select pin for each digit. Table 3 shows the required outputs for the required input combinations.

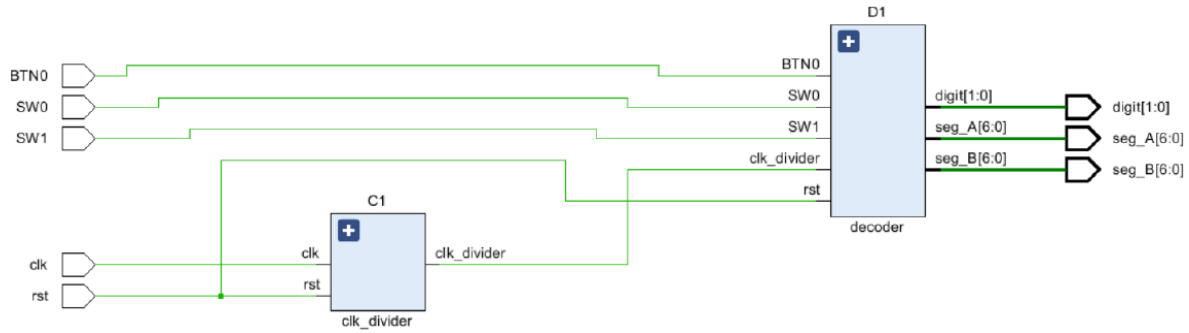

Figure 2: Part III block diagram. [1]

SW1	SW0	BTN0	Display
0	0	0	OFF
0	0	1	PYNQ
0	1	1	FPGA
1	0	1	IS
1	1	1	COOL

Figure 3: Part III inputs and outputs. [1]

The clock divider is realized with a counter that counts up until the divider ratio is reached of 500,000 as calculated with equation 1.

$$Divider\ ratio = \frac{f_1}{f_2 * 2} = \frac{125MHz}{125Hz * 2} = 500,000 \tag{1}$$

Therefore, a clk_temp register is used to count up until the counter value equal or great than 500,000 is reached. In the top module the divided clk_out of the clk_divider is wired into the decoder. The reason is why the clock needs to be divided at the first placed is that a two fast clock would prevent the viability of the seven-segment displays. The decoder is time multiplexed because each seven-segment display has a common carriage line to display ever a number on display A or display B. Therefor, a time multiplexer was written which is clock depended that each display one after another. The time multiplexer is realized with a case statement and a two bit value that iterates trough and each case displays one character of a word.

Further improvements that would increase the code quality and re-usability of the modules would be to change the clk_divider module that the divider ratio is given as input to the clock divider.

3 Simulation

Describes the result of the behavioral simulation based on the synthesized hardware description language.

3.1 Part I: behavioral simulation of decoder

After a successful simulation of the synthesized hardware description language that implements an decoder a test bench was written, see listening **??**. The time variant simulation is shown in figure 4 which shows steps trough the different possible switch positions and shows the output on the decoder bus seg. In comparison to the simulation given in Lab 02 Part 1 it could be confirmed that there are mostly identical.

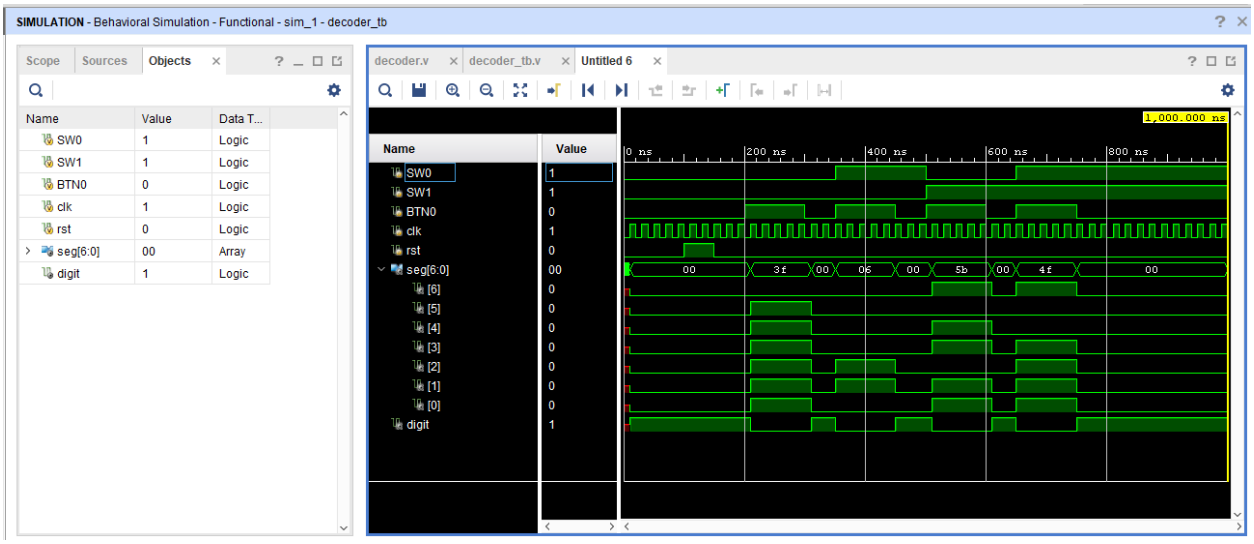

Figure 4: Vivado behavioral simulation of the decoder module.

3.2 Part III: behavioral simulation of the Time Multiplexer

The simulation was made to find bigs or logic faults in the code. Due to an extensive search of multiple hours to track down the issue stated in subsection 5.1.2 which lead to the simulation shown in figure 5. The simulation shows how the simulation of an instantiated time multiplex module behaviors over time which indicates that the clock divider is working. Furthermore, It shows how the decoder witches the display to the first word and time multiplexes the displays with the switching carries line cat0 and cat1.

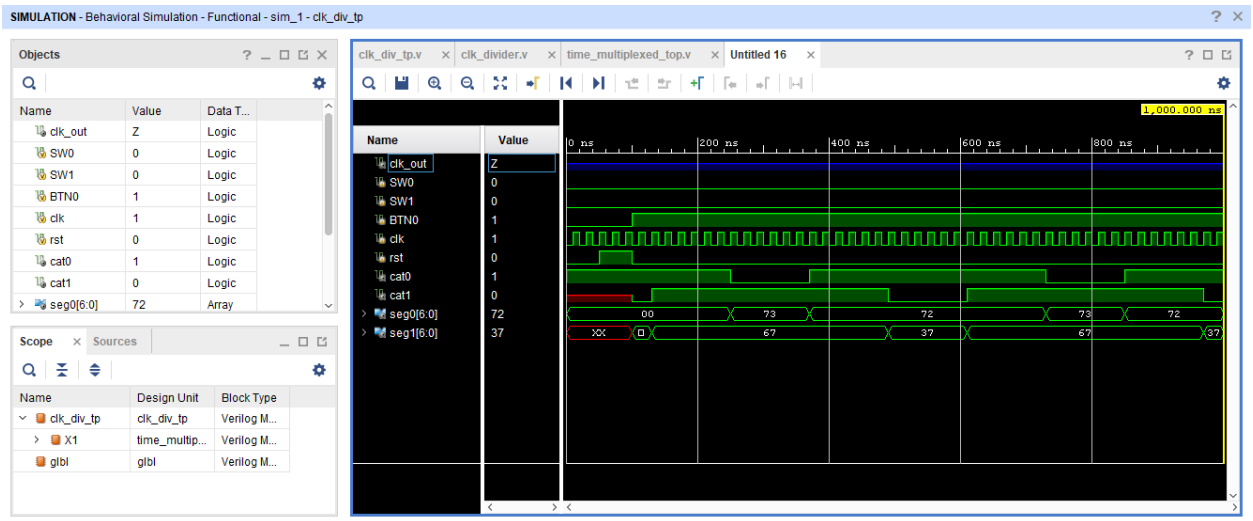

Figure 5: Vivado behavioral simulation of the decoder module.

4 Conclusion

The lab demonstrates in three parts how to build up in simple steps an top level hierarchy with multiple models by using simulations to proof behavior of a module based on set parameters in a test bench. Furthermore, the synthesized model was used to generate a bit stream and loaded on to the PYNQ development board for demonstration purpose.

5 Appendix

The appendix contains code listening and other large information parts that contain partial or complete relevance to the reports topic.

5.1 Lesson Learned

5.1.1 Vivado Language Templates

Figure 6 shows a handy tool build into Vivado software package that is called Language Templates and it seems to contain most of the verilog syntax.

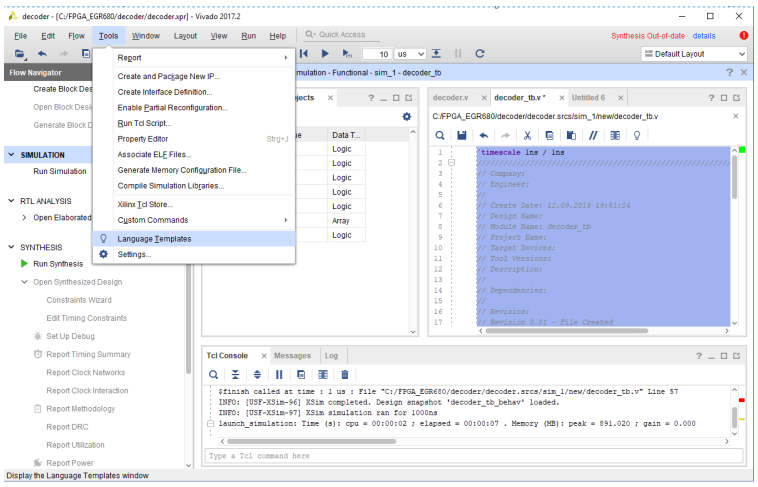

Figure 6: Vivado Language Templates.

5.1.2 Clock divider not working

Listening 1 shows a code snipe out of the clock divider that caused a lot of trouble by preventing the clock divider preventing dividing the clock. It seems that the last statement $clk_out <= clk_out$; overwrote the statement $clk_out <= clk_out$; which results either in an unknown state or an zero on the divided clock instead of the desired toggling clock behavior.

Listing 1: Clock divider issue.

```
else
begin

clk_temp <= clk_temp + 1;
if (clk_temp >= 500000)
//if (clk_temp >= 2) // Used for testbench
begin
clk_out <= ~clk_out; // no clue why this line does not toggle the clk_out
//    if (clk_out == 0)
//    begin
//    clk_out=1;
//    end
//    else
//    begin
//    clk_out=0;
//    end
```

```
clk_temp <= 0;
end
end // else rst
//clk_out <= clk_out; // could this lane of HDL be causong the problem of the not work
```

References

[1] "PYNQ-Z1 Board Reference Manual", Tech. Rep., 2017. [Online]. Available: www.pynq.io..