Axelou Olympia, 2161
oaxelou@uth.gr
Verilog, Spartan 3 FPGA
ce430
25/10/2017

Lab1: Circular Message on a 7-Segment Display

The purpose of this assignment is to create a driver in order to display a 16bit circular message on a seven-segment display of a Spartan 3 FPGA (Xilinx). The driver is implemented in Verilog. The tool for the FPGA programming I used is ISE 14.7 for Linux, Xilinx.

The design process I followed consists of the following steps: Circuit & Verilog Code -> Behavioral Simulation -> Post Route Simulation -> Bit File & test on FPGA board. When a bug occurred, I went back to previous steps to fix it.

The assignment had to be done in 4 parts:

- Part A: The LEDdecoder which connects the char to display and the LEDs of the anode that need to be open.
- Part B: The driver of the four digits which chooses the anode to open and the char to send to this particular anode.
- Part C: The circular motion of the message with the use of a button.
- Part D: Finally, the circular motion of the message using a constant delay.

I successfully implemented the four parts of the assignment. In the following paragraphs, there is a detailed explanation for every part.

Part A – BCD to 7-segment coding Decoder

Implementation

For this part, the circuit is a decoder. It's a combinational logic circuit which takes as input the characters (in binary coding) and generates the appropriate outputs for the segments to display the digit. E.g. for the char 'F': BCD: 4'b1111 & 7-segm coding: 7'b0111000.

The main parts of the circuit:

Modules: One module (the decoder), it has one always block (dataflow).

Verification

Testbench: It tests all of the 16 cases of the digit [0-F]. For each of these:

It waits 100ns, sets the input and the stdout (what we expect), it waits another 100ns and then checks the output with the stdout. It displays on the monitor updates on results. There weren't any problems.

The following screenshot shows a simulation of the circuit. The first signal is the output(7-segment coding), the second is the input(BCD), the third is the standard output and it's fixed and the signal correct_cases is a counter from the testbench.

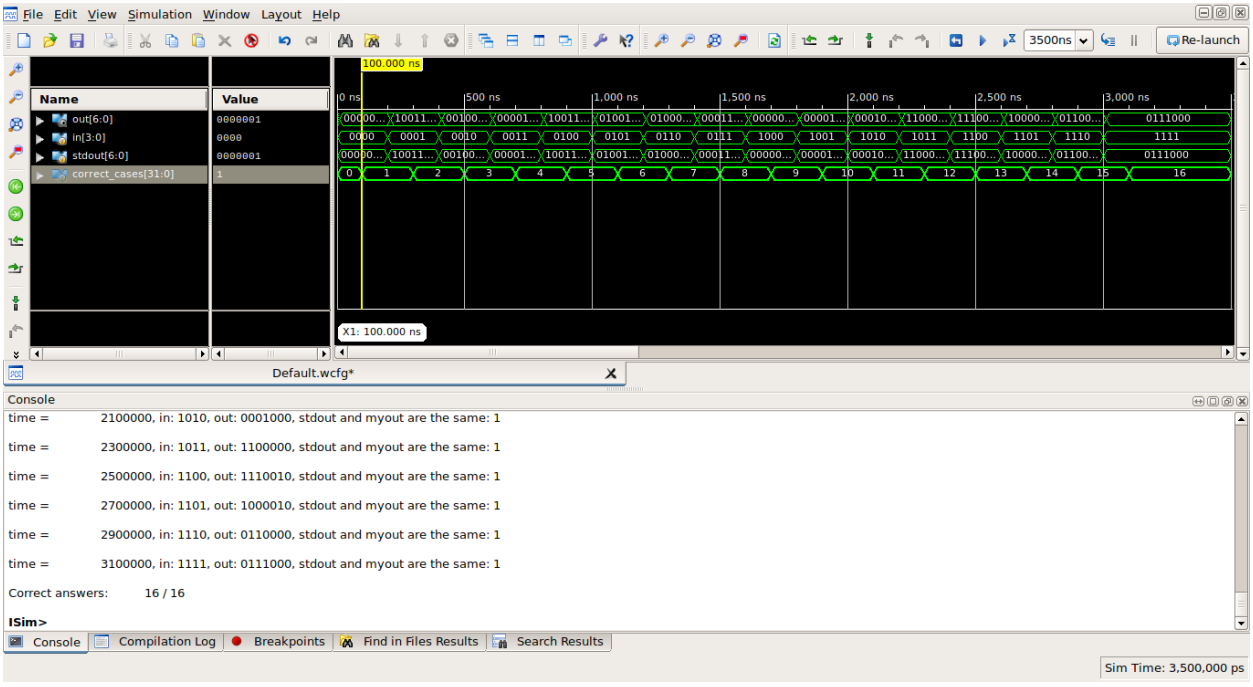

Figure 1: Part A simulation

Experiment / Final implementation

The first part was not tested on the FPGA, only B, C and D.

Part B – Static Message

Implementation

For this part, the circuit is composed of the following sub circuits – modules:

- The top-level module (FourDigitLEDdriver.v) which includes the instantiations of the sub-modules and the wired connection between them and between the inputs and the outputs. It also has the instantiation of the DCM module for the Spartan 3.
- The synchronizer module (reset_synchronizer.v) which has 2 always blocks (2 flip flops)
- The debouncer module (debounce_circuit.v) which has 3 always blocks (2 flip flops for the data collection – sampling) and another for the counter and the output connection.
- The finite-state-machine module (fsm.v) which is a control unit and it drives the chars to the corresponding anodes. It is composed of one always block with a case statement.
- The decoder module (from partA).

The main parts of the circuit:

- The circuit has as input the fpga clock (20ns) and delays it through the DCM module by dividing the clock frequency. The new clock has 320ns period.
- The synchronizer and the anti-bounce circuit use the fpga clock since the reset needs to be synchronized and checked for debouncing before the DCM. The rest of the circuit uses the new delayed clock.
- In the anti-bounce circuit, the sufficient cycles that the counter has to reach to consider the signal as stable is set to 2 cycles (2*20=40ns) for both the simulation and the testing on the FPGA since there is not a problem if we reset the circuit too many times.
- In the fsm circuit, the counter if a 4bit reg so there are 16 possible states. For every 4 states we focus on one anode. E.g. state 0: assign the char to display, state 1: do nothing, state2: open the anode (1->0) , state 3: close the anode(0->1).

Verification

Main testbench: With the inputs, it tests the reset button and then by observing the waveforms we check the circuit. For the reset, it first sends a simple signal (with no bouncing) for several fpga clock periods, then waits for some cycles and sends again a signal (with bounce on both positive edge and negative edge). It covers the two possible inputs for the reset. I also used a temporary testbench to check the synchronizer circuit and one for the anti-bouncer.

There weren't any major problems when testing it in Behavioral Simulation and Post Route Simulation. Only some in FSM (some regs were not being initialized when reset pressed).

This screenshot shows the correct function of the anodes and the characters (the cursor) and the correct function of the reset signal (especially at 3.2μs with the bouncing input).

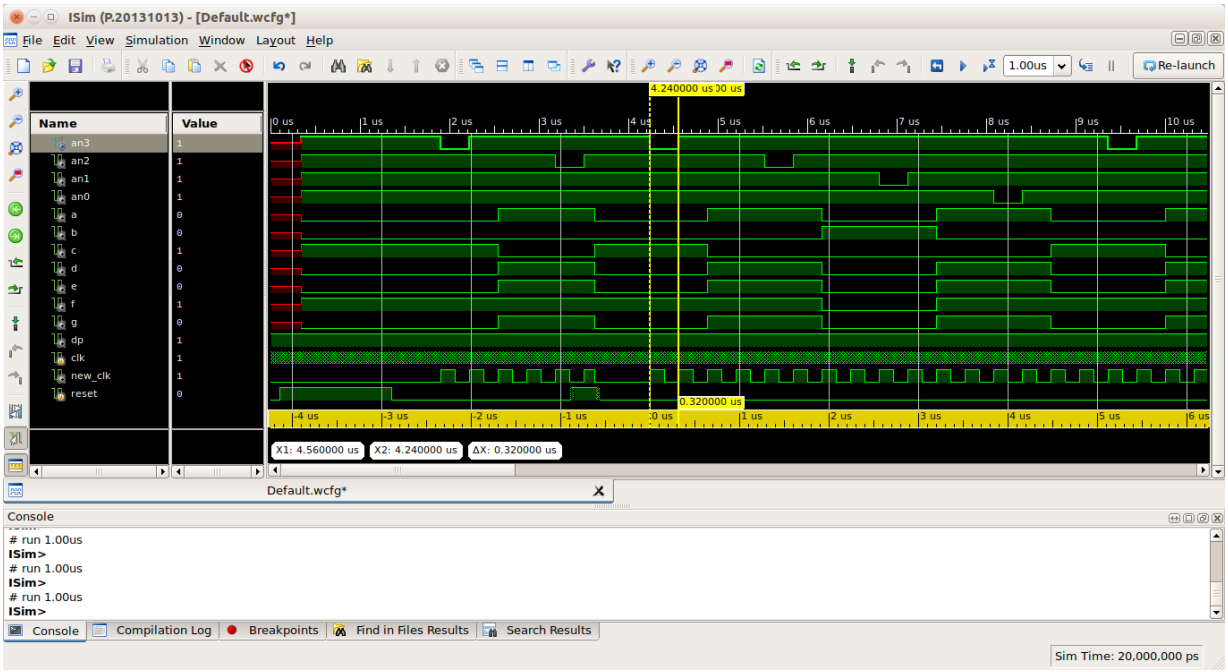

Figure2: Part B simulation

Experiment / Final implementation

Because of the small problems mentioned in the previous paragraph, the FPGA was not programmed with the original code. There had to be 2-3 corrections and then the Bit File was produced. The first time we loaded the circuit on the board, the reset button was working correctly but the digits displayed were only 4 eights (8888). The problem was that the assignment to the character to display was near the driving of the anode (an* = 0;) and it didn't have the time to change the value to display. I moved the character assignment one state behind and it worked.

The following photo was taken in the lab when testing part B. I displayed my student ID number.

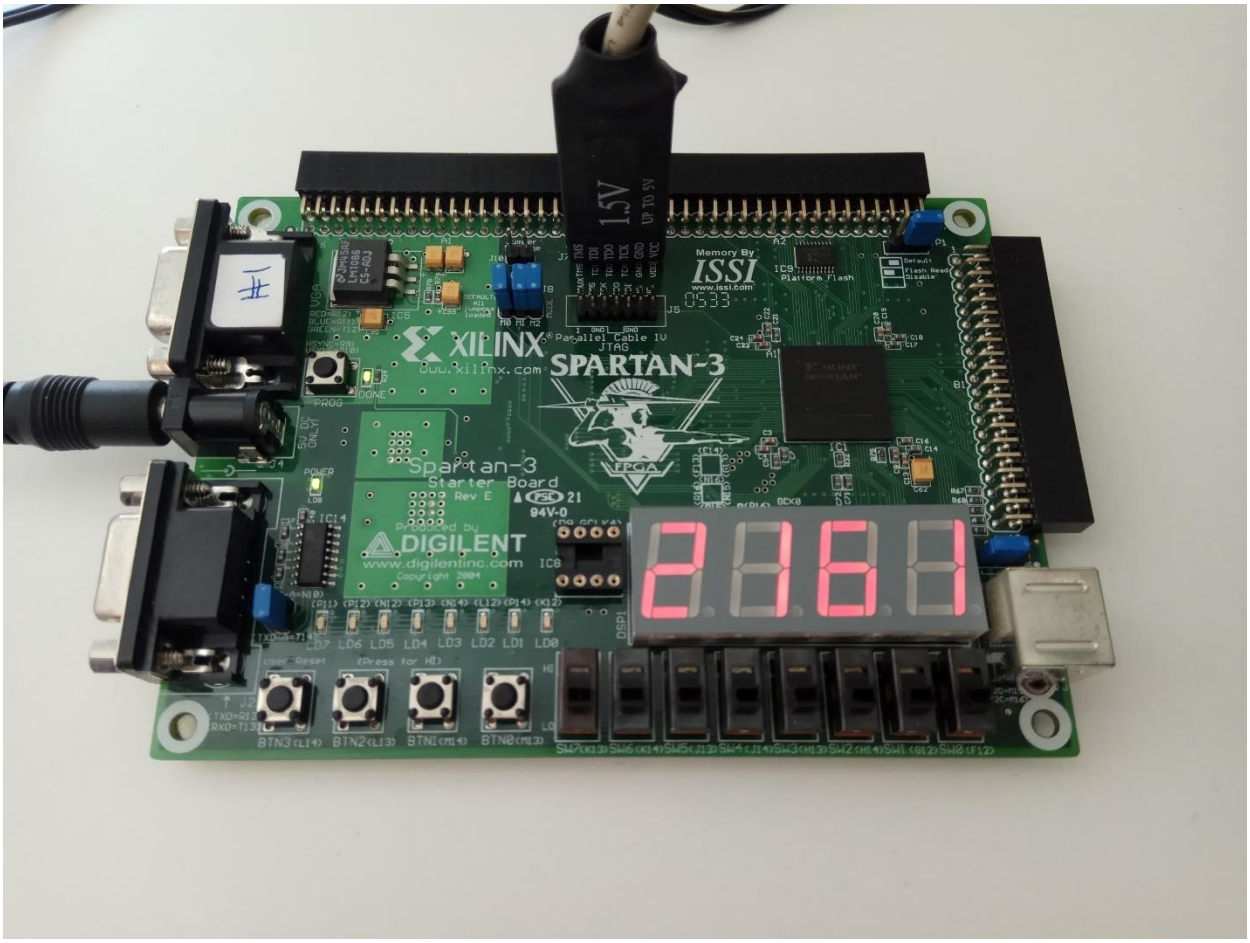

Figure 3: Part B: testing on the FPGA board

Part C – Circular message (button triggered)

Implementation

For this part, the circuit is composed of the following sub circuits – modules:

- The top-level module (FourDigitLEDdriver.v). The same as the one of the part B but with the only difference of the double instantiation of the synchronizer and the debouncer (one for the reset and the other for the button).
- The synchronizer module (from partB).
- The debouncer module (from partB).
- The finite-state-machine module (fsm.v). In this part it is composed of two always blocks: the one with the case statement (from partB) and the other one which is a sub-circuit for choosing the right index for the memory.
- The decoder module (from partA).

The main parts of the circuit:

- The two instances of the synchronizer and the anti-bouncer (4 in total) are in different clock domains: the one for the reset is in the clock domain of the fpga clock (20ns) and the other one is in the clock domain of the new one (320ns).
- In the anti-bounce circuit for the button, the sufficient cycles that the counter has to reach to consider the signal as stable is set to 2 cycles (2*320ns=640ns) for the simulation but for the testing on the FPGA is 62500 cycles (62500*320 = 0.02s).
- In the fsm circuit, in the first always block, it checks the old value of the button (the value it had in the previous cycle) and if it spots a positive edge it increments the index for the memory ([3:0]message[0:15]).
- Still in fsm, in the last version of the circuit, the index is a 5bit reg. The need to declare it as a 5bit and not a 4bit reg is explained in more details in the Verification section.

Verification

Testbench: First, it resets the circuit and then approximately every 20-25 cycles (enough time to open every anode at least once) is sends a button signal to move circularly the message. The total number of the button signals is 20, so that we can check what happens when it is near the end of the message.

The reset signal and the first two button signals have bounce but the other ones don't (since we have checked the circuit how it corresponds to bouncing).

In the beginning, in order to access the memory, I was simply using the index with an offset of +0, +1, +2 and +3 for each of the anodes. (e.g. char = message[index +2];). I did that having in mind that since the index is a 4bit reg the result of the addition 'index + 2' would also be a 4bit number. However, when index was 15, it was trying to access message[17] and did not return to 1. So, the correction I did, so that I do not have to create a circuit which explicitly checks if (index >= 13) and assigns specific values to the index of every node character, I declared the index as 5bit reg and use only the last [3:0] bits and I have connected index[4] to the ground so even if after the addition index == 17, it will discard the first '1' and it will become 1.

In the following screenshot, char is the character to display and the last 4 (index_an3-0) are the indices for the message for every anode. To the left of the cursor and to the right is noticeable that the index returns to zero and the message is properly displayed.

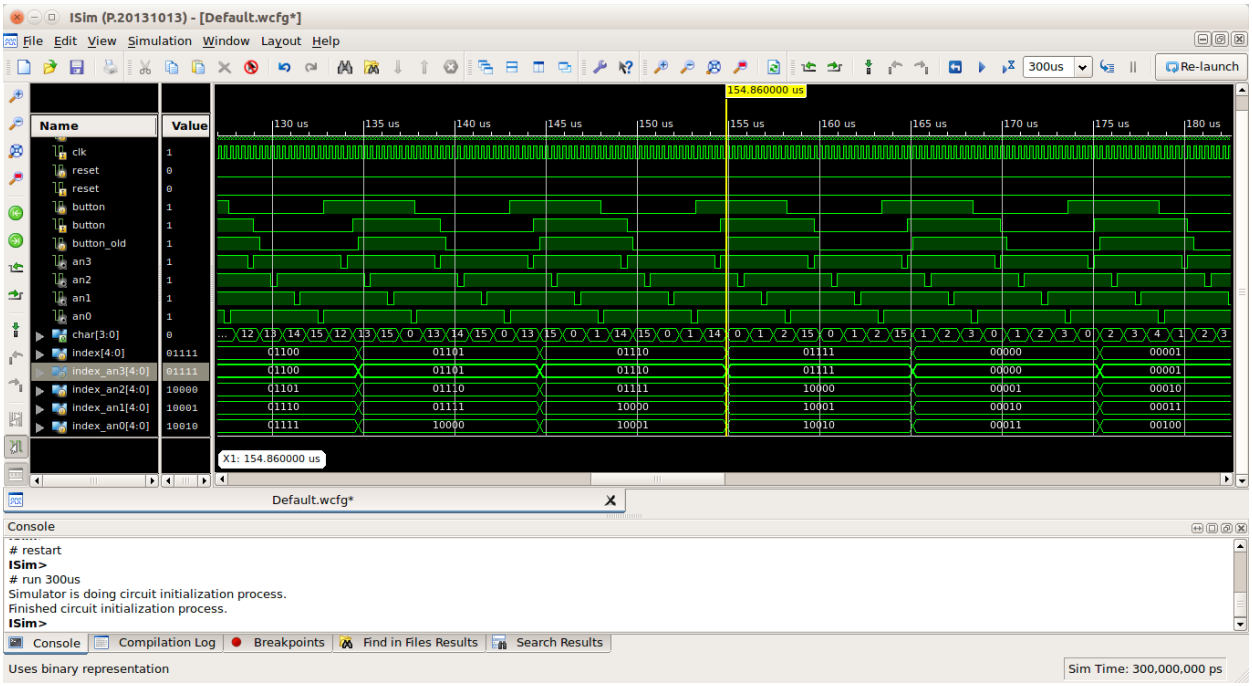

Figure 4: Part C simulation – circular motion of the message

Experiment / Final implementation

The first time we tested the code on the board, the button for the circular movement didn't respond well. In other words, we pressed it repeatedly and the message did not move all of the times. This bug was happening because I had originally set for the anti-bounce circuit the counter to reach 312500 cycles before considering it a stable signal. In real time, this is 0.2sec which is too long. I changed it to the final form of 62500 cycles (0.02sec) and everything was ok.

Part D – Circular message with a counter

Implementation

For this part, the circuit is composed of the following sub circuits – modules:

- The top-level module (FourDigitLEDdriver.v). The same as the one of the part B.
- The synchronizer module (from partB).
- The debouncer module (from partB).
- The finite-state-machine module (fsm.v). The same as the one of partC with the only difference that the sub-circuit for choosing the right index for the memory is not button triggered. Instead, we use a counter.
- The decoder module (from partA).

The main parts of the circuit:

- In the fsm circuit, we use a counter for the delay. In every cycle, this counter is incremented and when it reaches the maximum possible number (1_1) it increments the index for the message. This counter is a 22bit reg as it has to be of enough bits so that the message moves every second (2^22 * 320ns = 1.3421 sec). For the simulation I declared it as a 5bit reg.

Verification

Testbench: Sends two reset signals at a long distance the one from the other to test the initialization of the regs at a random time. Since the only inputs are the clock and the reset there aren't many cases to examine, only the waveforms observation of the functionality of the counter.

There wasn't any "radical" change from part C to part D, so most (if not all) of the problems were common and were solved in part C.

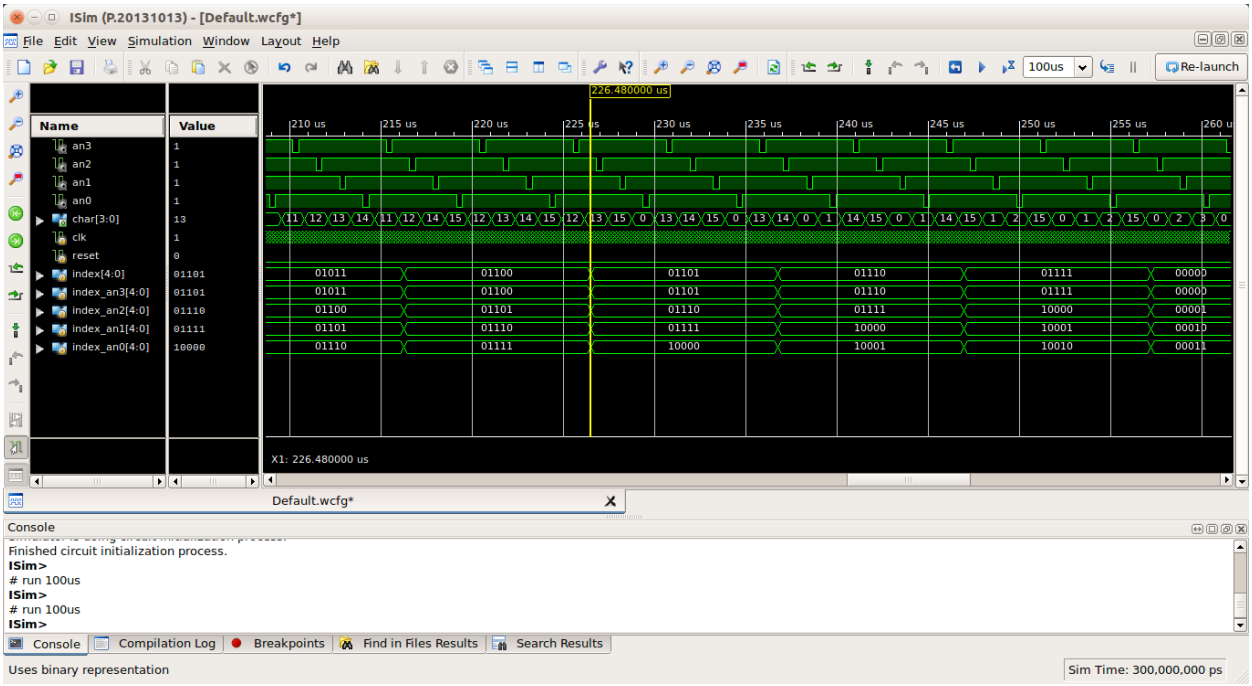

Figure 5: partD simulation

Experiment / Final implementation

The implementation of the parts C and D were checked on the board in the same laboratory hours, so the common issues were found and solved at the same time.

While correcting and testing in the Simulation the two parts and under the pressure of the laboratory I forgot to change the counter of part D for the delay from 5bit (simulation mode) to the final 22bit (for the 1.34sec real time delay) and the result was 4 eights (8888). I created a new Bit File with the correct counter and everything was working.

Apart from that I didn't face any other problems, so in total we tested this part 3 times.

Conclusion

The most crucial problem I faced during this assignment were the tools. In the first week I was trying to install and work on the ISE with ModelSim together but with no success. In the laboratory, the assistants recommended working on ISim.

After that, the assignment had officially begun. Part A was complete easily and quite fast. Then part B was a bit more complex (especially with the implementation of the debouncer). It was checked in the next lab and it was ok.

Then in part C and part D were those issues with the index out of bounds of the memory and some minor initialization issues (which were found and corrected from the simulation stage).

In conclusion, most of the issues were of minor importance and will not occur again in the next assignments.

~