Rodrigo Becerril Ferreyra
CECS 301 Section 03
Lab 2
Due 25 February 2020

1 Introduction

In this lab, we used the language Verilog to be able to use the seven-segment display on the Nexys A7 FPGA board. A seven-segment display is a tool to display values internal to the FPGA. It can be used to display the contents of a register, but in this lab, we are using it to display the value presented on the switches. We can later reuse the code in order to display values on an internal register to aid with debugging a module.

Displaying a value on the seven-segment display is more complex than it sounds, because many different factors come into play. First, it is necessary to choose which four switches are going to be displayed. This is because only one set of four bits can be displayed at a time. This is achieved using a multiplexer, which will take four of the bits that are presented via the switches and feed it into a decoder which will decode it into eight bits; these decoded eight bits will go into the cathode inputs of the seven-segment display. The LEDs only have one set of eight cathode inputs, so we need a multiplexer in order to select which four switches go to which eight inputs. Since there are only eight inputs, there is an enable switch to display which LED the eight inputs go into. This is controlled by a rotating shift register, which only allows one active signal to pass through it. Lastly, in order to see the changes happening, it is necessary to slow the frequency of the FPGA's internal clock down. Since this cannot be done directly, it is necessary to create a one- clock-wide pulse every x amount of clocks. In this lab, we are creating a pulse every 5000 clock cycles, and assuming that the FPGA's clock is set to 100 MHz, our pulse will be 20 kHz.

2 Process

In total, there are five different modules to make: a 4-bit 8:1 multiplexer, a custom 4:8 decoder, a pulse generator, a 3-bit counter, and a shift register. At first, I made all five modules separately and instantiated them in a separate top-level module, but then cut and pasted the code from the five modules into one, for easy re-use.

2.1 Multiplexer, Decoder, and Counter

The multiplexer was simple to construct: it is a purely combinatorial circuit, and all it takes is one `always @(*)` block in order to create one. Because the multiplexer is 32 bits wide total, and the top-level input is only 16 bits, it was necessary to copy the same inputs into the top 16 bits of the mux.

The decoder is another purely combinatorial circuit. It takes a 4-bit input from the mux and turns it into an 8-bit output. The output logic is in accordance to standard seven-segment display procedures. The output of this decoder is connected to the `cathode` output of the top-level module.

The select signal of the mux is controlled by a counter; this counter counts from 0 to 7 using three bits; these bits are fed into the mux. The counter increments synchronously with the 8-bit shift register, because the counter's "increment" input and the register's "shift" input are tied to the pusle generator's "pulse" output. This allows the the mux's n-th input to coincide with having a zero in the register's n-th position.

2.2 Pulse Generator

The pulse generator is an integral part of this design, as it allows the user to see the numbers. If no pulse generator were present, and the "increment" and "shift" inputs of the counter and shift register respectively were directly connected to the clock, the LEDs of the seven-segment display would turn on and off too quickly, and no number would show.

The pulse generator works by having an internal counter which counts clock cycles. When a fixed number of clock cycles x is reached, the pulse generator outputs a pulse and the counter gets set to zero. x is calculated by dividing the internal clock frequency of the FPGA by the desired clock frequency. Originally, my desired clock frequency was 2 kHz, but I experienced annoying flashing of the LEDs, so I increased it by a factor of ten. Therefore, my new desired frequency is 20 kHz, and the number of clocks $x = 5000$.

The output of the pulse generator is hooked up to the "increment" and "shift" inputs of the counter and shift register, respectively.

2.3 Shift Register

The last piece of the puzzle is the shift register. Specifically, it is a rotating left shift register; this means that the register only shifts left, and whatever bit is at the MSB position will be inserted into the LSB position. This behavior can be achieved in a variety of ways, such as using an `if` statement, but I decided to use the concatenation operator. The output of this register is connected to the `anode` output of the module.

3 Conclusion

In this lab, I learned how to efficiently model sequential logic and separate it from combinatorial logic. I also learned how to write a top-level module in two ways—by making all the modules separately and instantiating them in a separate module, or by writing all the code in a single module—and the pros and cons of each format. Overall, this lab will help me with future labs, as I gained much experience.