Project 1: Reaction Timer

Jacob Boline

September 7, 2018

1 Introduction

In this project I was tasked with creating and testing a design that would measure a humans response to visual stimulus. The device is required to have 3 buttons that correspond to start, stop, and clear. The basic premise of the design is to have a random delay of 2-15 seconds after the start button is pushed, but before the LED is lit. Once the LED is lit, a 4-digit display will show the number of milliseconds it takes a person to press the stop button. In addition, there are some extra features added to make the design more robust such as a state if the stop button is pushed too early, or not pushed at all. To accomplish this project Vivado 2017.2 was used to simulate, synthesize, and implement System Verilog code for a Basys 4 DDR development board. The project design files can be found at: https://github.com/txjacob/SoC_FPGA

2 Experimental Plan

For this design five fundamental Modules are needed:

1. 7-seg Driver
2. Random Number Generator
3. Universal Counter
4. Stopwatch
5. State Machine

2.1 7-Seg Driver Module

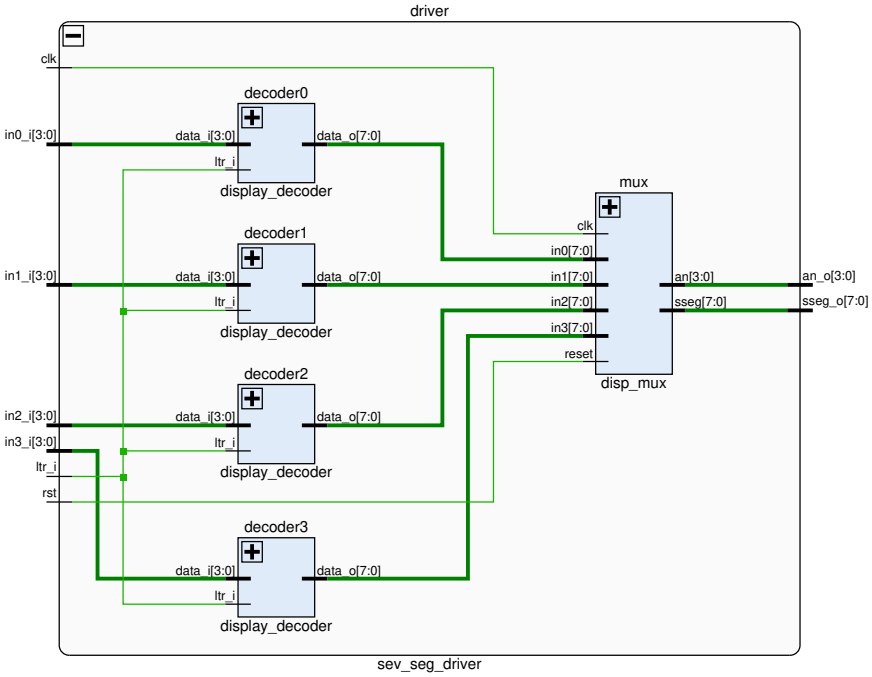

Figure 1: Block diagram of 7-seg driver module.

The first step to building this design was to create a method for driving the 7-segment display. This was accomplished by creating a decoder that had a 4-bit data in port, a 1-bit flag to indicate whether the data corresponded to numerical values, or alphabetic values, and an 8-bit data out port that corresponds to what portions of 7-segment display to turn on. Using 4 instantiations of this decoder, and the display mux from listing 4.15 in the class textbook I created a driver module (as seen in Figure 1) that had 4 separately controlled digits, and could display the numbers 0-9 and the letters H and I.

2.2 Random Number Generator Module

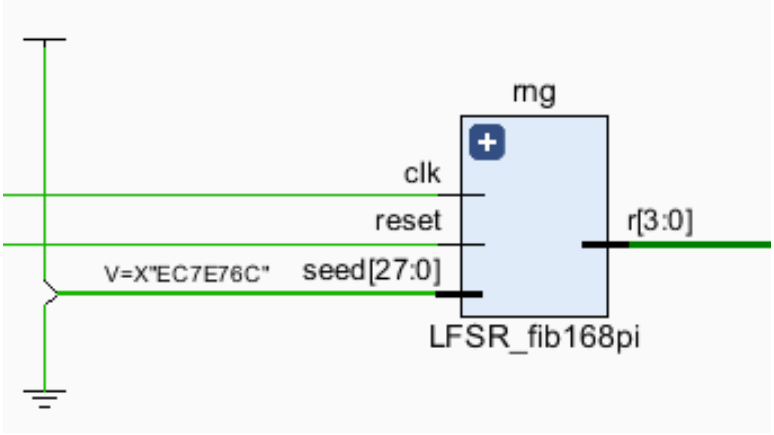

Figure 2: Block diagram of LFSR module.

The random number generator module (Figure 2) is a modified version of the lfsr_fib168pi design provided. In order to fit better into the project I modified the output to be a 4-bit number instead of a single bit. This was done by reading the first four bits of the state variable instead of using the straight polynomial variable. The 4-bits from the state variable were then passed through some combinational logic to limit the range of random numbers to be between 2 and 15. This was done by checking to see if the top 3-bits were all 1s, if they were then the two's place bit was zeroed out, if not the data was allowed to pass through. Finally, the data was summed with the number 2, and output from the module.

2.3 Universal Counter Module

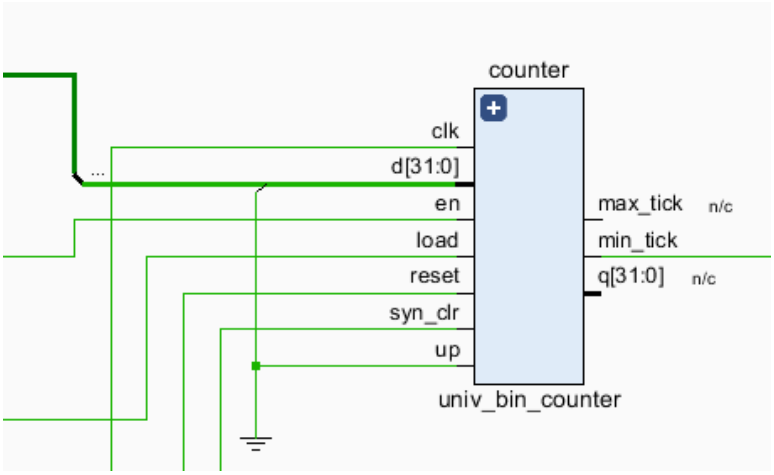

Figure 3: Block diagram of the Universal Counter module.

The universal counter module (Figure 3) is taken from Listing 4.12. It was configured to be in countdown mode so that the state machine only has to wait for the zero flag to go high. The input of the counter was connected to the output of the random number generator multiplied by 100,000,000. This was done to convert the value in seconds to the number of 100 MHz clock cycles. Finally, three flags were created

to control the counter: clear_counter, load_counter, and enable_counter. Using these 3 flags the counter configuration can be individualized for each state.

2.4 Stopwatch Module

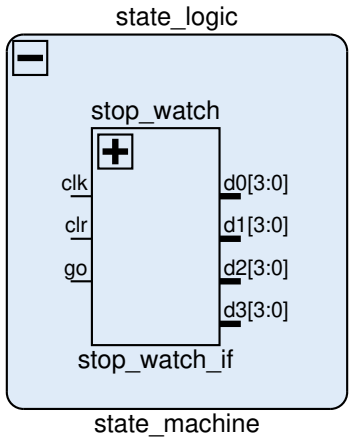

Figure 4: Block diagram of the stopwatch module.

The stopwatch module (Figure 4) is a modified version of the stop watch design in listing 4.20 of the class text book. To make the module more applicable for my design I extended the output to use all 4 digits on the 7-segment display, and adjusted the 1 ms tick generator factor. The factor was derived in the following manner:

$$factor = \frac{100 \times 10^6 \; cycles/s}{1 \times 10^{-3} \; ms/s} = 100 \times 10^3 \; cycles/ms$$

Using the clear and go pins of the module, the stopwatch can be started, stopped, and cleared independently in each state. In addition, the digit outputs are either connected to the the display driver, or left open depending on the current state.

2.5 State Machine Module

For the state machine I elected to use states that corresponded to six different states in the design's sequence. The six states are listed below and an ASMD diagram is shown in Figure 5.

1. init
2. start
3. delay
4. count
5. stop
6. bad_stop

2.5.1 init state

The function of the init state is simple. While in this state "HI" is printed to the 7-segment, the stopwatch is cleared, and the universal counter is cleared. This state can only be reached either by pressing the global reset button, or from the stop and stop_bad states. This state can only be exited by pressing the start button on the board, at which point the start state is entered.

2.5.2 start state

While in the start state the the display is blanked and the universal counter is loaded by asserting the load_counter pin to HIGH. This state immediately transitions to the delay state on the next clock cycle.

2.5.3 delay state

In this state the load count pin is set to LOW, and the enable counter pin is set to HIGH. This allows the counter to latch in a random value and count down from that value. This state is exited once the universal counter hits zero (at which point the zero_count flag will be asserted) or if the stop button is pushed. If the stop button is pushed the bad_stop state will be entered, but if the button is never pushed before the universal counter reaches zero then the count state will be entered.

2.6 count state

Once the count state is entered the output of the stopwatch is connected to the 7-segment driver module, the LED is turned on, and the stop watch is started by asserting timer_go to HIGH. This state is stayed in until the stop button is pushed, or the stopwatch reaches 1000ms, whichever comes first.

2.6.1 stop state

In this state the timer is stopped by asserting timer_go to LOW. This causes the stopwatch to stop thus displaying the time it took for the stop button to be pressed after the LED was turned on. This state can only be exited by pressing the clear button, at which point the init state will be entered and the process will begin again.

2.6.2 bad_stop state

This state is only entered if the stop button is pushed while in the delay state (and the universal counter has not yet reached zero). Once in this state the counter will be disabled, and "9999" will be displayed on the 7-segment. The only was to exit this state is by pressing the clear button, which will return the user to the init state.

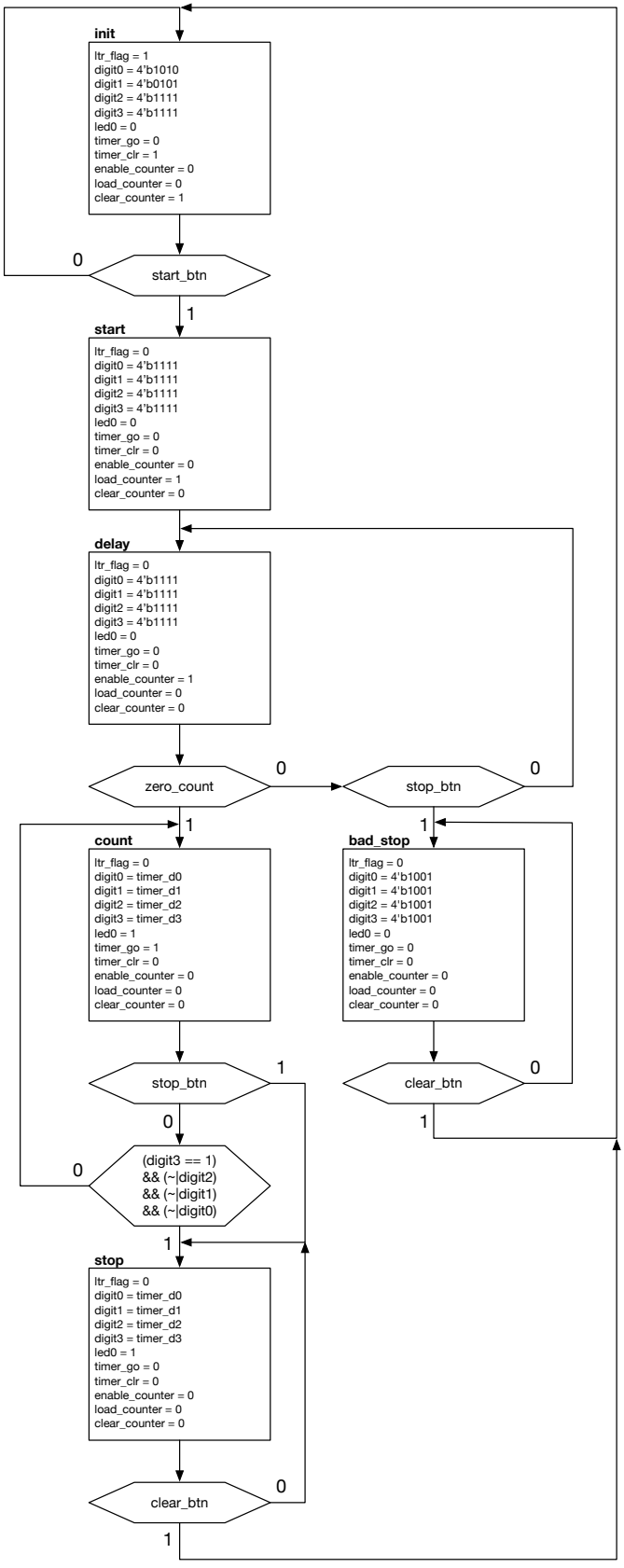

Figure 5: State diagram of the reaction timer design

3 Analysis

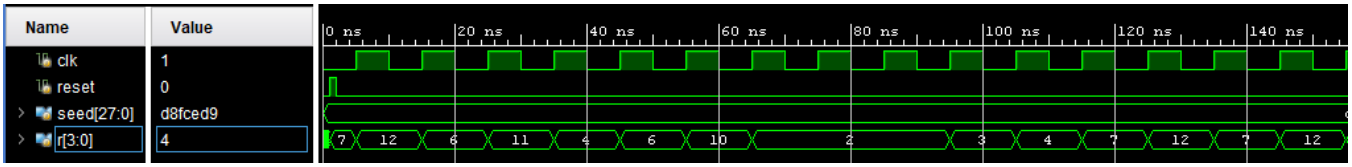

Figure 6: Simulation of LFSR Module

In Figure 6 the simulation for the LFSR is shown. In this simulation it is demonstrated that for every clock cycle a random number is generated.

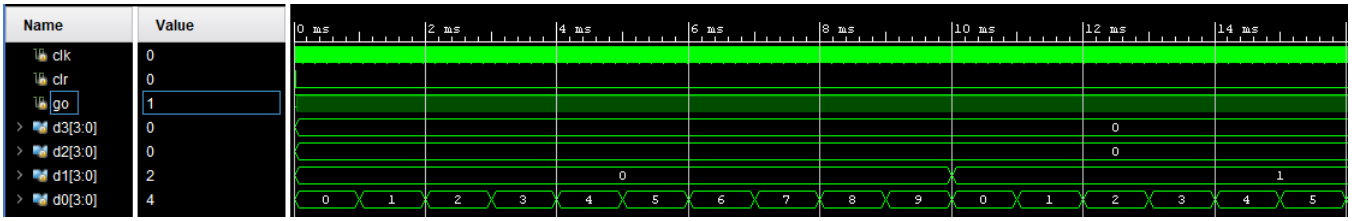

Figure 7: Simulation of the Stopwatch Module

In Figure 7 it is demonstrated that the stopwatch digits will increment once every millisecond. Since this module was pulled from the textbook, the only verification that was done was checking the conversion value for going from the 100 MHz clock, to a 1 ms increment.

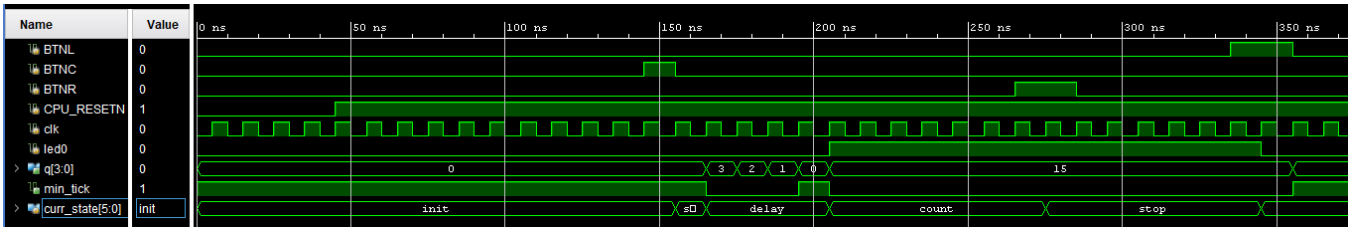

Figure 8: Simulation of the entire design showing states.

Figure 8 shows a simulation of the entire system in response to simulated button presses. Shown is the system sitting in the INIT state until the start button (BTNC) is pushed. Then the system goes into the START state in order to load the counter, and immediately enters the DELAY state until the min_tick line is asserted. Then it goes into the COUNT state where it increments the display for every millisecond until the stop button (BTNR) is pushed. Finally, the system sits in the STOP state until the clear button (BTNL) is pushed at which point it returns to the INIT state. Note: The delay state is sped up for the purposes of simulation.

4 Conclusion

From this report the processes for designing this system can be seen. The hope for this design was to create something that was simple, yet robust. By using pre-existing modules the design and verification

time was cut down, and the reliability of the system was increased. In addition, some design choices were made to simplify the design process, but at the expense of using more expensive resources. However, since we are designing for a chip with plentiful resources, I believed that this was an acceptable compromise.