Reaction Timer FPGA Implementation

Justin Bui

September 2, 2018

Contents

1 Introduction

This document outlines the Reaction Timer project designed for ELC 5396. This project makes use of the Digilent NEXYS 4 Artix-7 FPGA development board to implement a reaction timer, which measures the response to a randomly generated stimulus. The document below includes the ASMD diagram, functional discription, and thought process behind my programming decissions, as well as testing and performance verification notes.

2 Project Requirements

The Reaction Timer requirements are outlined in problem number 6.5.6 of FPGA Prototyping By SystemVerilog Examples, by Pong Chu. The project is designed to measure the reaction time to a visual stimulus. The design uses the following criteria:

1. Three Input Buttons: Clear, Start, and Stop
2. Pressing the Clear button returns the program to the initial state.
3. Pressing the Start button initializes the test.
4. After a random time between 2 and 15 seconds, the stimulus LED is turned on and the reaction timer is started.
5. The player then has up to one second to press the Stop button. If the timer reaches one second, the player "loses".
6. Pressing the Stop button before the stimulus is generated results in a "false start", producing an error

In addition to the 7-Segment display, I have implemented one of the RGB LEDs to allow for state identification. This not only allows for a more visable game interface, but allows for debugging if necessary. The 7-Segment display will show different values, depending on the state of the game. The following table identifies the 7-Segment Display and RGB LED values for the particular states.

Table 1: State Vales of 7-Segment Display and RGB LED

State	**7-Segment Display**	**RGB LED**
Ready	H I 0 0	Blue
Start	0 0 0 0	Blue
Game	Run Time	Off
Error	9 9 9 9	Yellow
Win	Reflex Time	Green
Lose	1 0 0 0	Red

3 Thought Process and ASMD

At a high level, the reaction timer can be broken down in to a few core functions, those being the timing element(s) and the display element(s). The textbook provides great code examples for both of these elements, and I have implemented my own version of each for this design. In addition to the timing and display elements, a state machine can be used to better control the programs flow and operation. As you can see from the table on page 3, I have designed this system to make use of 6 states (Ready, Start, Game, Error, Win and Lose). A brief descrption of each state can be found below.

Ready: The Ready state is used as my default state, in which the game is held until the player presses the Start button (BTNR). If the player presses the Clear button (BTNC) at any time, the system returns to the Ready state.

Start: When the Start button is pressed, a random number is generated and the timer begins to count towards that random number. When the random number of seconds has been reached, the device moves to the Game state. If the user presses the stop buttong when in this state, the system registers a "false start", and the system moves in to the Error state.

Game: When the Stimulus LED (LED0) is first turned on, the player has up to 1 second to press the Stop button (BTNL). If the player hits the Stop buttong during this time, the system moves to the Win state. If 1 second has elapsed, the system moves in to the Lose state.

Win: When the player wins the game, the 7-Segment displays= shows their raction time (in ms) and the RGB LED turns green. In order to leave the Win state, the player must press the Clear button.

Lose: When the player loses the game, the 7-Segment display shows 1000 and the RGB LED turns red. As with the Win state, the Clear Button must be pressed in order to leave the Lose state.

Below are the two versions of the ASMD diagrams used to program the reaction timer. These diagrams provided a general outline of how the state machine controlling the reaction timer operates, as well as state transitions.

As you can see, I have emphasized simplicity and code reusability for this project. In the next section, I will go in to more detail regarding the functionality of the Reaction Timer.

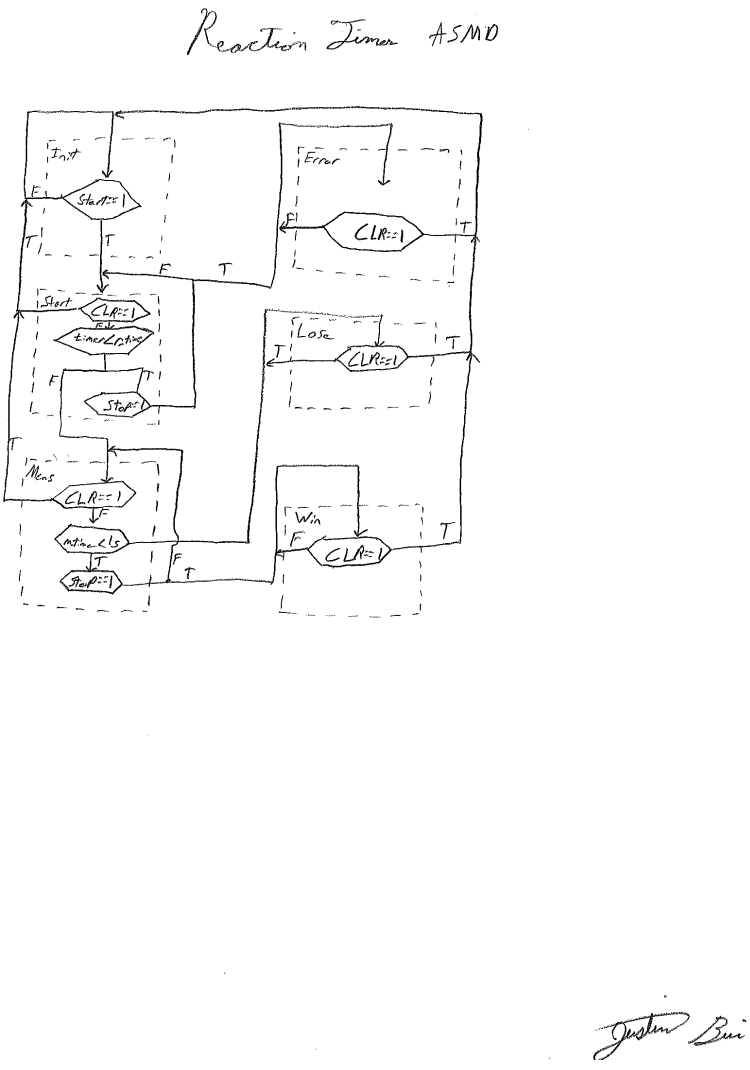

Figure 1: Initial ASMD with basic transitions

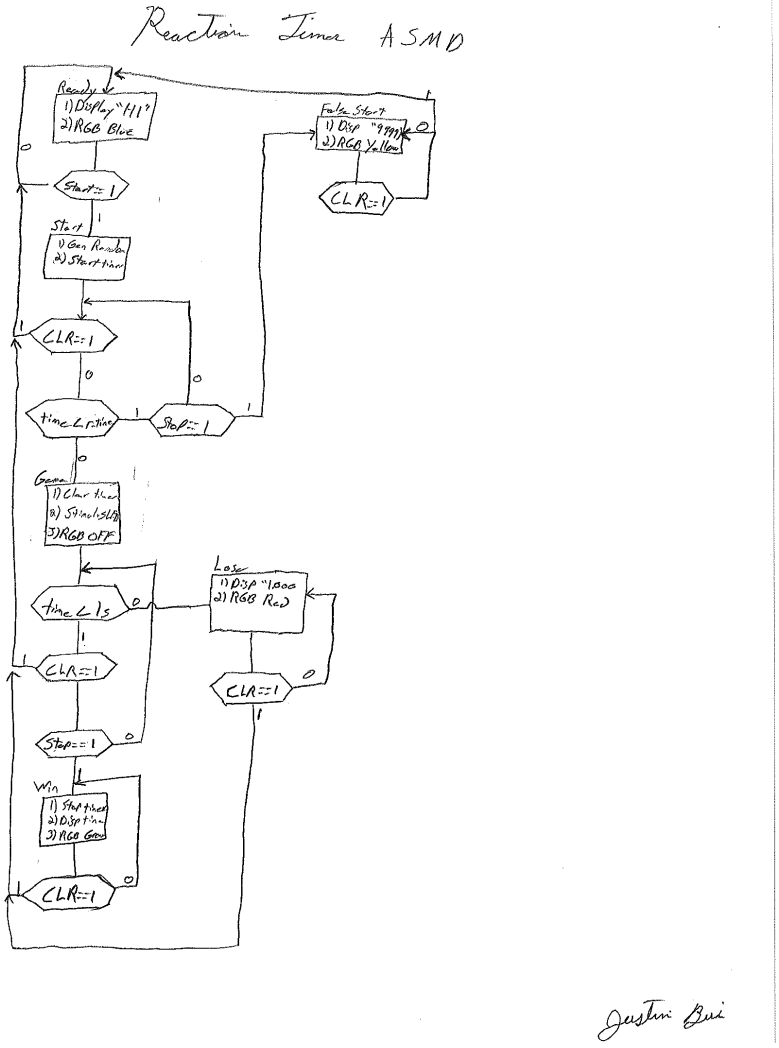

Figure 2: Second Draft ASMD with More Detail

4 Reaction Timer Functionality

As previously mentioned, the Reaction Timer makes use of a number of different elements, including the stopwatch and display modules. I have designed the stopwatch module to emulate the stopwatches used by coaches for athletic events. I've designed the stopwatch to work as a Binary Coded Decimal timer, counting up to 9 and incrementing the next decade. I have set each digit to be a 4 bit number (as seen in the code snipit below. This allows for 0-9 on the ms digits, and 0-F for the seconds digit. I have not capped the seconds digit at 9 to allow for the random 2-15 second time interval to make use of that value (maximizing code reusability).

```
//Declarations
localparam DVSR = 100000;
//milisecond registers
logic [24:0] ms_reg;
logic [24:0] ms_next;

//decade registers (seconds, ds, cs, ms)
logic [3:0] d3_reg, d2_reg, d1_reg, d0_reg;
logic [3:0] d3_next, d2_next, d1_next, d0_next;
```

For the 7-Segment display control, I have modified the values for Hex codes A and B (1010 and 1011) to display H and I. The code segment below shows the case statement used to "decode" the number to the required display settings.

```
always_comb
begin
        case(val)
            4'h0: sseg[6:0]=7'b1000000; // '0'
            4'h1: sseg[6:0]=7'b1111001; // '1'
            4'h2: sseg[6:0]=7'b0100100; // '2'
            4'h3: sseg[6:0]=7'b0110000; // '3'
            4'h4: sseg[6:0]=7'b0011001; // '4'
            4'h5: sseg[6:0]=7'b0010010; // '5'
            4'h6: sseg[6:0]=7'b0000010; // '6'
            4'h7: sseg[6:0]=7'b1111000; // '7'
            4'h8: sseg[6:0]=7'b0000000; // '8'
            4'h9: sseg[6:0]=7'b0010000; // '9'
            4'ha: sseg[6:0]=7'b0001001; // 'H'
            4'hb: sseg[6:0]=7'b1001111; // 'I'
            4'hc: sseg[6:0]=7'b0000110; // 'E'
            4'hd: sseg[6:0]=7'b0000110; // 'E'
            4'he: sseg[6:0]=7'b0000110; // 'E'
            default: sseg[6:0]=7'b1111111;
         endcase
         sseg[7] = dp;
end
```

The final submodule used for the implementation of the Reaction Timer is the random number generator. I made use of Dr. Schubert's 168 bit Fibonacci Pi generator, modified to grab 4 arbitrary bits (in this case 110-113 of the machine state]. The random number generator is constantly running, making it more difficult to predict the timing.

To address the issue of the constantly running random number generator, I've made use of an additional register which stores the random number (in seconds). This prevents the random time from changing while the game is in the Start state.

The top level design, in addition to implementing the modules mentioned above, defines the remaining game logic. This logic covers the state transitions, display configurations (LEDs and 7-Segment Display values), memory registers (timer values, state values, etc, seen below). Since much of this code is heavily commented and exceeds the marigins for this document, I have posted the code on GitHub to be viewed at anytime. it may be found at: The complete code can be found on github.

5 Design Verification

The Reaction Timer has been verified through operation and manual testing. By taking advantage of the RGB LED, I am able to determine which state the device is in, as well as verify the timing with an external source. You can see in Table 1 that the RGB LED color indicates which state it is currently in. For future designs, I plan on taking advantage of the simulation techniques to quickly verify the design. Since this design was manually verified, I have provided video documentation showing the functionality of the system. To support the video, below you can see images of the game in it's various states.

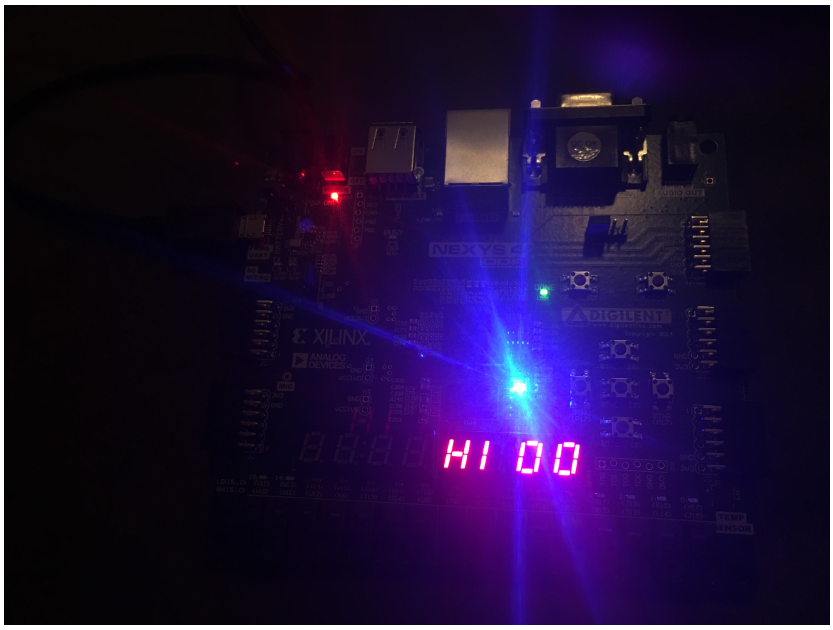

Figure 3: Ready State: Blue RGB LED and HI message

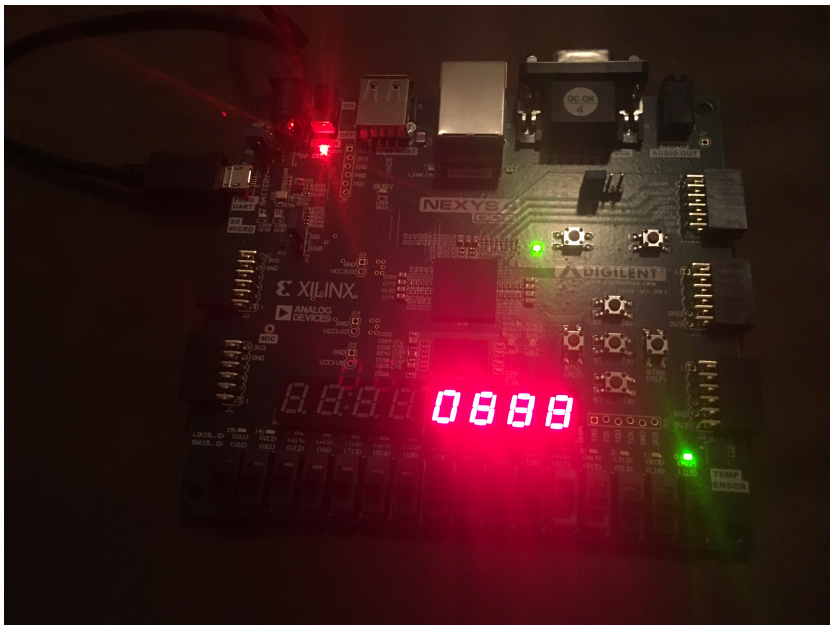

Figure 4: Game State: Stimulus LED, No RGB LED, and Current Time

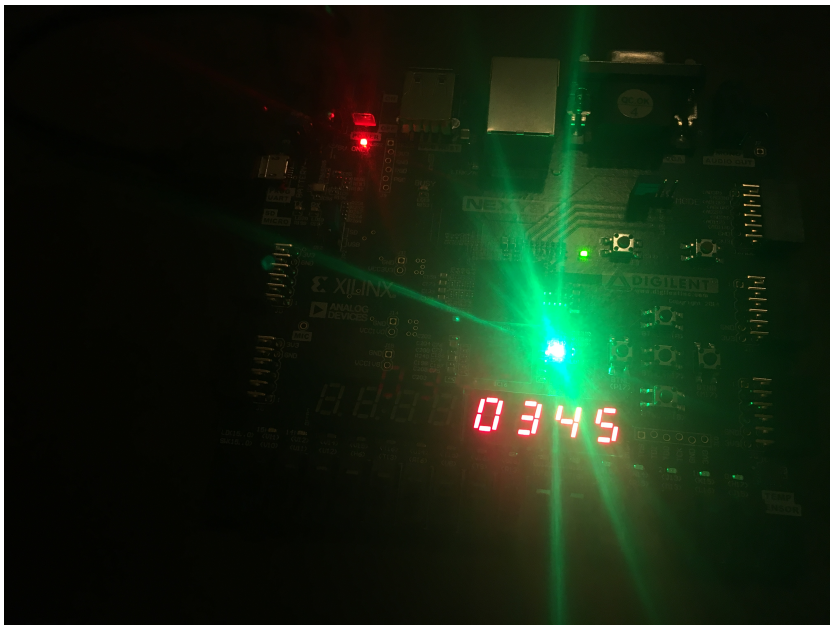

Figure 5: Win State: Green RGB LED and Reaction Time

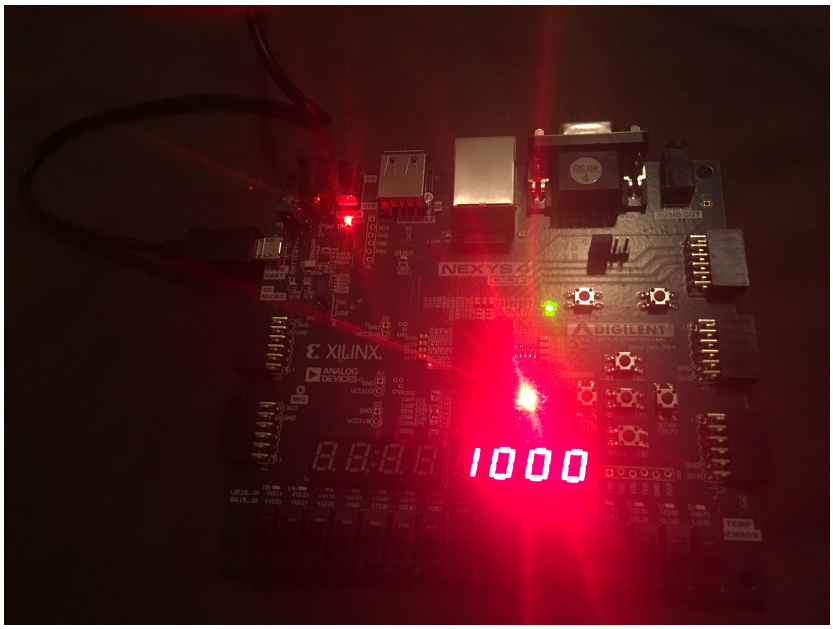

Figure 6: Lose State: Red RGB LED, and 1000 Count

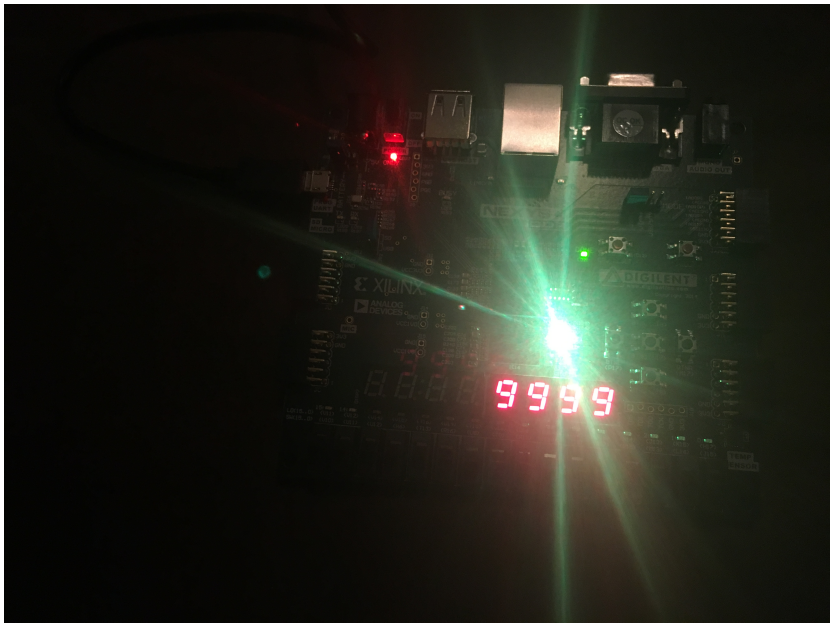

Figure 7: Error State: Yellow LED and 9999 Display Value