CSEE 4270 Report: Final Project

Victor Philippe
School of Electrical and Computer Engineering
University of Georgia
Vlp47226@uga.edu

Contributions: Please list contributions (in estimated percentages) of each member in the following categories.

- Pre-lab design and analysis: Philippe
- In-lab module and testbench design: Philippe
- In-lab testbench simulation and analysis: Philippe
- In-lab FPGA synthesis and analysis: Philippe
- Lab report writing: Philippe

Abstract

The final project provides a new extension of state machines with a look at High Level State Machine (HLSM) to simulate a visual reaction timer. Here, multiple components including a clock divider and a pseudo random number generator are designed and simulated using a testbench in Verilog in the Xilinx environment. This lab introduces new concepts like pseudo random number generation as well as another state machine using HLSM through its design and simulation.

1. INTRODUCTION

The reaction timer will measure a user's reaction time by counting the time elapsing after the user has reacted and pushed a button after a pseudo random amount of time has elapsed. The reaction has two smaller components that are the clock divider to slow the speed of the device down as well as a module dedicated to generating random numbers. The HLSM will be implemented using one-procedure and feature a behavioral approach as well. The sequence of events that governs this particular HLSM can be seen in Figure 1. Similarly, the high level diagram of the project is shown in Figure 2.

Figure 1: Sequence of events of HLSM

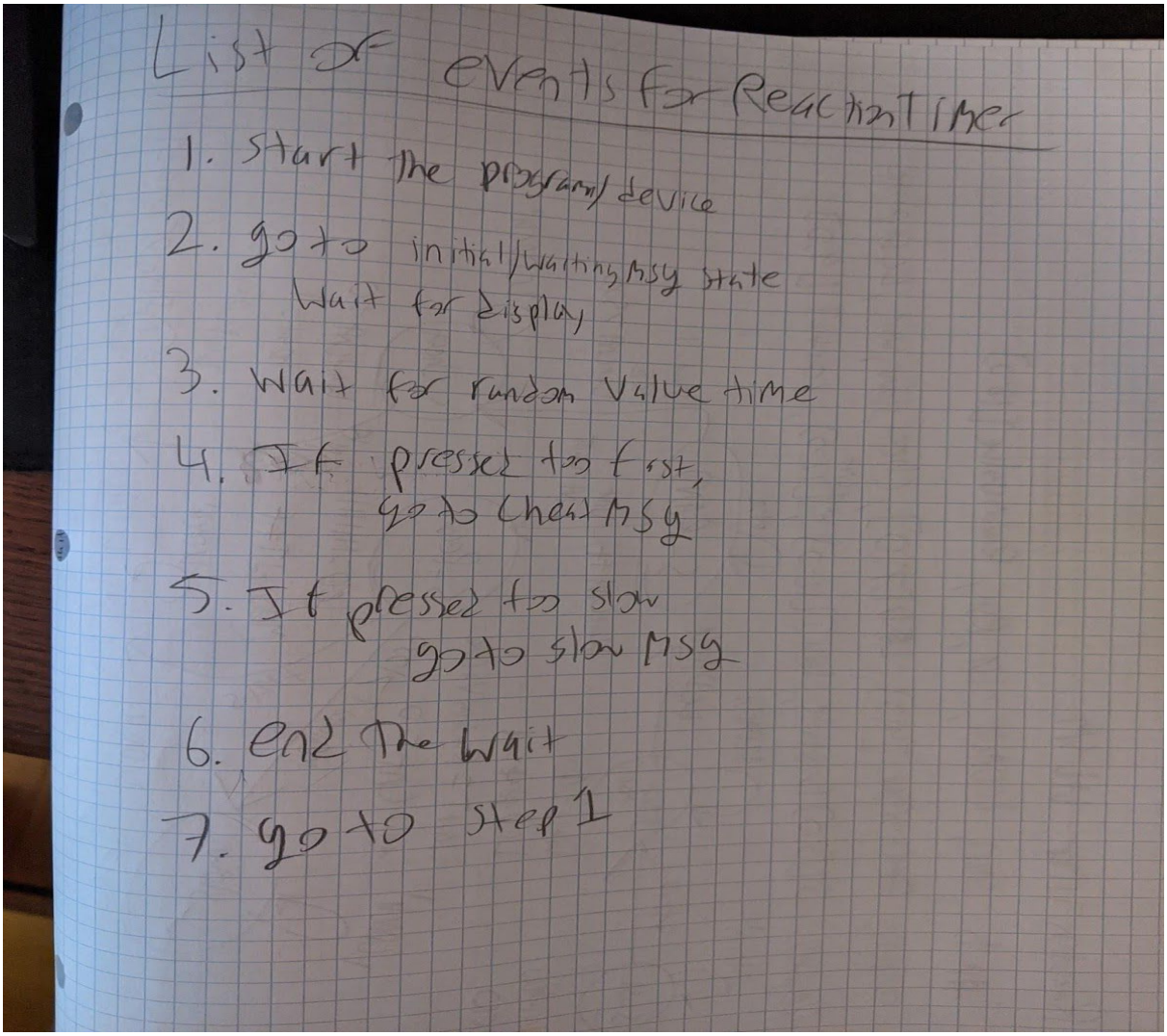
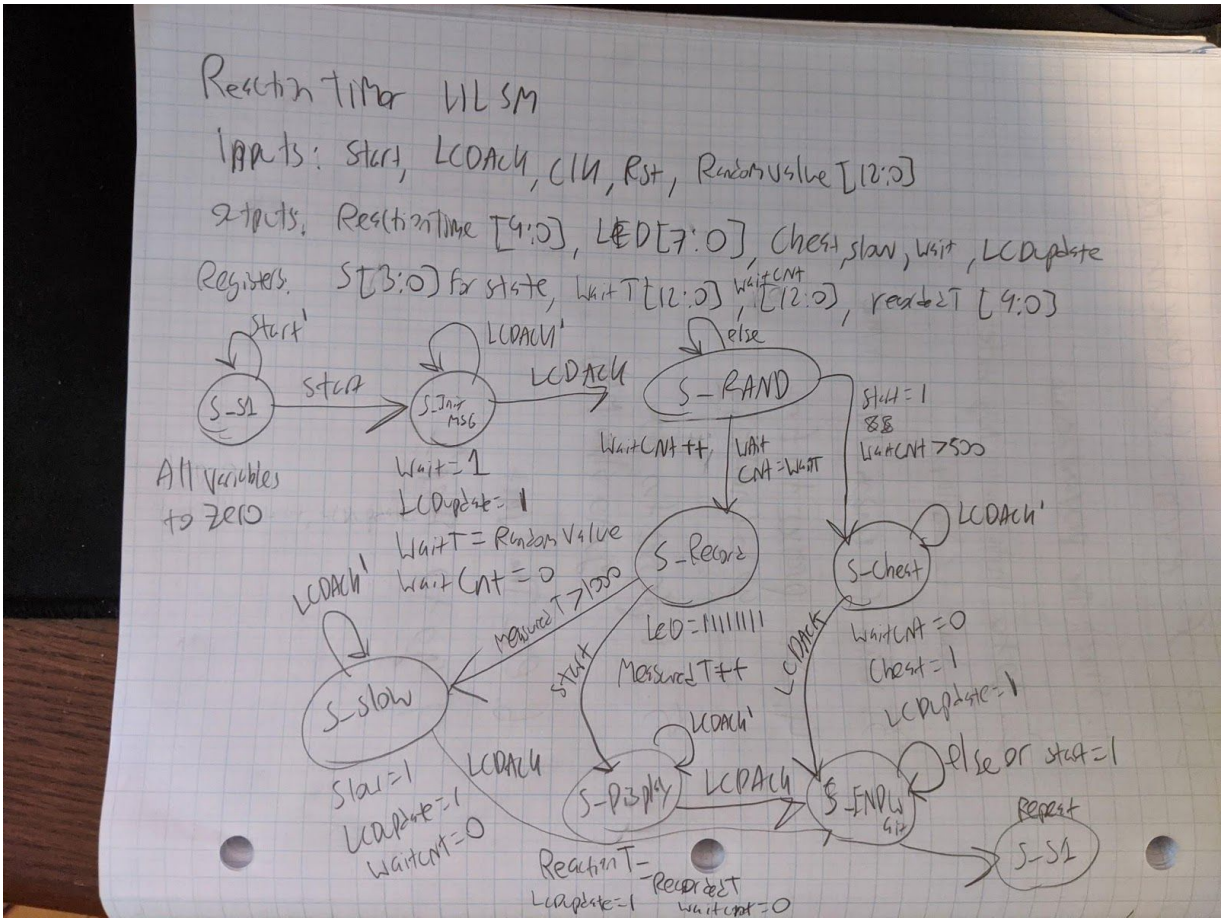


Figure 2: Diagram of HLSM



2. IMPLEMENTATION

The first lower level module to be designed and implemented was the clock divider. This clock divider was tasked with taking in a 50 MHz clock signal and converting it to 1KHz. It's behavioral design is shown in figure 3. Figure 4 demonstrates the test bench for the clock cycles as it shows a complete cycle in 1ms. Figure 5 and 6 show the behavioral design and testbench for the pseudo random number generator. Figure 7 and 8 show the design of the HLSM as well its testbench without the number generator and with it. The simulations of these test benches will be shown in the Experimental Results section.

Figure 3: Verilog module for clock divider

```
module ClkDiv(Clk, DivReset, ClkOut);
        input Clk, DivReset;
  output reg ClkOut;
  reg ClkInt;
  reg[24:0] Count;
  parameter Value = 24999;

  always @(posedge Clk) begin
    if( DivReset == 1 )begin
      Count <= 0;
      ClkOut <= 0;
      ClkInt <= 0;
```

```
    end
    else begin
      if( Count == Value ) begin
        ClkOut <= ~ClkInt;
        ClkInt <= ~ClkInt;
        Count <= 0;
      end
      else begin
        ClkOut <= ClkInt;
        ClkInt <= ClkInt;
        Count <= Count + 1;
      end
    end
  end
endmodule
```

Figure 4: Test bench module for clock divider

```
module ClkDiv_tb();
        reg Clk_s, ClkReset_s;
  wire ClkOut_s;
  ClkDiv ClkDiv_1(Clk_s,ClkReset_s,ClkOut_s);
  // Clock Procedure
  always begin
    Clk_s <= 0;
    #10;
    Clk_s <= 1;
    #10;
  end
  initial begin
    ClkReset_s <= 1;
    @(posedge Clk_s);
    #5 ClkReset_s <= 0;
    @(posedge Clk_s);
    @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
    @(posedge Clk_s);
```

```
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
    @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
    @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
    @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
    @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
    @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
```

```
                    @(posedge Clk_s);
                    @(posedge Clk_s);
        @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
        @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
        @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
        @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
                    @(posedge Clk_s);
  end
endmodule
```

Figure 5: Pseudo Random Number generator

```
module RandomGen(Clk, Rst, RandomValue);
        //variables
  input Clk, Rst;
  output [12:0] RandomValue;
        //constants
        parameter MinValue = 1000;
        parameter MaxValue = 5000;
```

```
        //local variables
        reg [12:0] Val;
        reg [12:0] Counter = 0;

        always @(posedge Clk) begin
                if (Rst == 1) begin
                        Val = MinValue;
                end
                else begin
                        if (Counter % 2 == 0) begin //if Counter is even
                                Counter = 2* Counter + 3;
                        end else begin //if Counter is odd
                        Counter = 3 * Counter + 4;
                        end
                        Val = Val + Counter; //makes a psuedo random number
                        if (Val > MaxValue) begin //if the number too big
                                Val = MinValue;
                        end
                        if (Val < MinValue) begin //if the bumber too small
                                Val = MaxValue;
                        end
                end
        end
        assign RandomValue = Val;
endmodule
```

Figure 6: Testbench for PRNG

```
module RandomGen_tb();

        reg Clk_s, Rst_s;
  wire [12:0] RandomValue_s;

        RandomGen RandomGen_1(Clk_s, Rst_s, RandomValue_s);

        always begin
    Clk_s <= 0;
    #10;
    Clk_s <= 1;
    #10;
  end
```

```
    initial begin
Rst_s <= 1;
@(posedge Clk_s);
#5 Rst_s <= 0;
@(posedge Clk_s);
@(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
@(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
@(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
@(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
@(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
            @(posedge Clk_s);
```

```
                @(posedge Clk_s);
                @(posedge Clk_s);
@(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
@(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
@(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
@(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
@(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
@(posedge Clk_s);
                @(posedge Clk_s);
```

```
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
                @(posedge Clk_s);
  end




endmodule
```

Figure 7: HSLM Verilog module

```
module ReactionTimer(Clk, Rst, Start, LED, ReactionTime, Cheat, Slow, Wait, RandomValue,
LCDUpdate, LCDAck);


        //variables
        input Start, LCDAck, Clk, Rst;
        input [12:0] RandomValue;
  output reg [9:0] ReactionTime;
        output reg [7:0] LED;
  output reg Cheat, Slow, Wait;
        output reg LCDUpdate;
        //local variables
        reg [3:0] S;
        reg [12:0] WaitT;
  reg [12:0] WaitCnt;
  reg [9:0] RecordedT;
        //constants
  parameter S_S1 = 0, S_InitMsg = 1, S_Rand = 2, S_Record = 3, S_Cheat = 4, S_Slow = 5,
        S_Display = 6, S_EndWait = 7;

  always @(posedge Clk) begin
    if (Rst == 1) begin
      S <= S_S1;
                        Cheat <= 0;
                        Slow <= 0;
                        Wait <= 0;
                        LED <= 0;
                        WaitT <= 0;
```

```
                WaitCnt <= 0;
                RecordedT <= 0;
                ReactionTime <= 0;
                LCDUpdate <= 0;
        end
else begin
  // Default Values
                Cheat <= 0;
                Slow <= 0;
                LED <= 0;
                Wait <= 0;
                ReactionTime <= 0;
                LCDUpdate <= 0;

                case (S)
                        S_S1: begin
                                RecordedT <= 0;
                                if (Start == 1) begin
                                        S <= S_InitMsg;
                                end
                                else begin
                                        S <= S_S1;
                                end
                        end

                        S_InitMsg: begin
                          Wait <= 1;
                                LCDUpdate <= 1;
                                if (LCDAck == 1) begin
                                  WaitT <= RandomValue;
                                  WaitCnt <= 0;
                                        S <= S_Rand;
                                end
                                else begin
                                        S <= S_InitMsg;
                                end
                        end

                        S_Rand: begin
```

```
        if (Start == 1 && WaitCnt > 500 ) begin
                S <= S_Cheat;
        end
        else if( WaitCnt == WaitT ) begin
          LED <= 8'hFF;
                S <= S_Record;
        end
        else begin
          WaitCnt <= WaitCnt + 1;
                S <= S_Rand;
        end
end

S_Record: begin
  LED <= 8'hFF;
        if (Start == 1) begin
                S <= S_Display;
        end
        else if( RecordedT >= 1000 ) begin
                S <= S_Slow;
        end
        else begin
          RecordedT <= RecordedT + 1;
                S <= S_Record;
        end
end

S_Cheat: begin
  Cheat <= 1;
        LCDUpdate <= 1;
        if (LCDAck == 1) begin
                WaitCnt <= 0;
                S <= S_EndWait;
        end
        else begin
                S <= S_Cheat;
        end
end
```

```
S_Slow: begin
  Slow <= 1;
        LCDUpdate <= 1;
        if (LCDAck == 1) begin
                WaitCnt <= 0;
                S <= S_EndWait;
        end
        else begin
                S <= S_Slow;
        end
end

S_Display: begin
  ReactionTime <= RecordedT;
        LCDUpdate <= 1;
        if (LCDAck == 1) begin
                WaitCnt <= 0;
                S <= S_EndWait;
        end
        else begin
                S <= S_Display;
        end
end

S_EndWait: begin
        if (Start == 1) begin
                S <= S_EndWait;
        end
        else if( WaitCnt == 1000 ) begin
                S <= S_S1;
        end
        else begin
          WaitCnt <= WaitCnt + 1;
                S <= S_EndWait;
        end
end

default: begin
        S <= S_S1;
```

```
                                    WaitT <= 0;
                                    RecordedT <= 0;
                                end
                            endcase
                        end
    end
endmodule
```

Figure 8: a) testbench without RandomGen

```
module ReactionTimer_tb();
        reg Clk_s, Rst_s, Start_s, LCDAck_s;
        reg [12:0] RandomValue_s;
    wire LCDUpdate_s, Cheat_s, Slow_s, Wait_s;
        wire [9:0]  ReactionTime_s;
        wire [7:0] LED_s;
        ReactionTimer ReactionTimer_1(Clk_s, Rst_s, Start_s, LED_s, ReactionTime_s, Cheat_s,
Slow_s, Wait_s, RandomValue_s, LCDUpdate_s, LCDAck_s);


        always begin
      Clk_s <= 0;
      #10;
      Clk_s <= 1;
      #10;
    end


        initial begin
                RandomValue_s <= 700;
                #5 Rst_s <= 1; Start_s <= 1;LCDAck_s <= 1;
                @(posedge Clk_s);
                #5 Rst_s <= 0;
                @(posedge Clk_s);
                #40 LCDAck_s <= 0;
                @(posedge Clk_s);
                #10135 LCDAck_s <= 1; Start_s <= 0;
                @(posedge Clk_s);
                #10000 RandomValue_s <= 300;
                @(posedge Clk_s);
                #24815 Start_s <= 1;
                @(posedge Clk_s);
```

```
                #6500 Start_s <= 0;
                #5 LCDAck_s <= 0;
                @(posedge Clk_s);
                #20495 Start_s <= 1; LCDAck_s <= 1;
                @(posedge Clk_s);
                #100 Start_s <= 0; LCDAck_s <= 0;
                @(posedge Clk_s);
                #26100 LCDAck_s <= 1;
                @(posedge Clk_s);
        end

endmodule
```

b) testbench with randomGen

```
module RandomTimerWGen_tb();

reg Clk_s, Rst_s, Start_s, LCDAck_s;
        wire [12:0] RandomValue_s;
  wire LCDUpdate_s, Cheat_s, Slow_s, Wait_s;
        wire [9:0]  ReactionTime_s;
        wire [7:0] LED_s;
        RandomGen RandomGen_1(Clk_s, Rst_s, RandomValue_s);
        ReactionTimer ReactionTimer_1(Clk_s, Rst_s, Start_s, LED_s, ReactionTime_s, Cheat_s,
Slow_s, Wait_s, RandomValue_s, LCDUpdate_s, LCDAck_s);

        always begin
    Clk_s <= 0;
    #10;
    Clk_s <= 1;
    #10;
  end

        initial begin
                #5 Rst_s <= 1; Start_s <= 1;LCDAck_s <= 1;
                @(posedge Clk_s);
                #5 Rst_s <= 0;
                @(posedge Clk_s);
                #40 LCDAck_s <= 0;
                @(posedge Clk_s);
```

```
        #20 Start_s <= 0;
        @(posedge Clk_s);
        #20 LCDAck_s <= 1;
        @(posedge Clk_s);
        #20 Start_s <= 1;
        @(posedge Clk_s);
        #20 LCDAck_s <= 0;
        @(posedge Clk_s);
        #20 Start_s <= 1;
        @(posedge Clk_s);
        #20 LCDAck_s <= 1;
        @(posedge Clk_s);
        #20 Start_s <= 1;
        @(posedge Clk_s);
        #10000 LCDAck_s <= 1; Start_s <= 0;
        @(posedge Clk_s);
        #20020 Start_s <= 1;
        #40 Start_s <= 0;
        @(posedge Clk_s);
        #40000 LCDAck_s <= 0; Start_s <= 1;
        @(posedge Clk_s);
        #5000 LCDAck_s <= 1; Start_s <= 0;
        @(posedge Clk_s);
        #24815 Start_s <= 0;
        @(posedge Clk_s);
        #6500 Start_s <= 0;
        #5 LCDAck_s <= 0;
        @(posedge Clk_s);
        #40 Start_s <= 0;
        @(posedge Clk_s);
        #60 Start_s <= 1;
        @(posedge Clk_s);
        #20495 Start_s <= 0; LCDAck_s <= 1;
        @(posedge Clk_s);
        #100 Start_s <= 0; LCDAck_s <= 0;
        @(posedge Clk_s);
        #25000 LCDAck_s <= 1; Start_s <= 0;
        @(posedge Clk_s);
end
```

endmodule

3. EXPERIMENTAL RESULTS

The first tested component was the clock divider which needed to show a complete cycle of 1ms over the waveform. Figure 9 shows that from the start of the simulation the clock out does not go high until around 0.5ms (500us) and then goes to 0 shortly after 1ms(1000us). This shows a complete cycle of low then high then low for the waveform, proving its competence. Figure 10 moves onto the next module to test which is the random number generator. The test bench needed to show that random or effectively random numbers were generated. This is the case as my alternating equations and separate counter resulted in this phenomena. Figure 11 goes into the reaction timer elements and starts with the waveform without the random gen. Here, the waveform starts with a complete reset of the system. Then, it starts the first test which is to provide a cheating response. This is done with a random value of greater than 500 and start being high. Once it goes through the cheating sequence and through the end wait, it starts the next test. This is to correctly measure the reaction time. After it comes back to the start, it does the last test of measuring a slow reaction

Figure 9: Waveform for clock divider

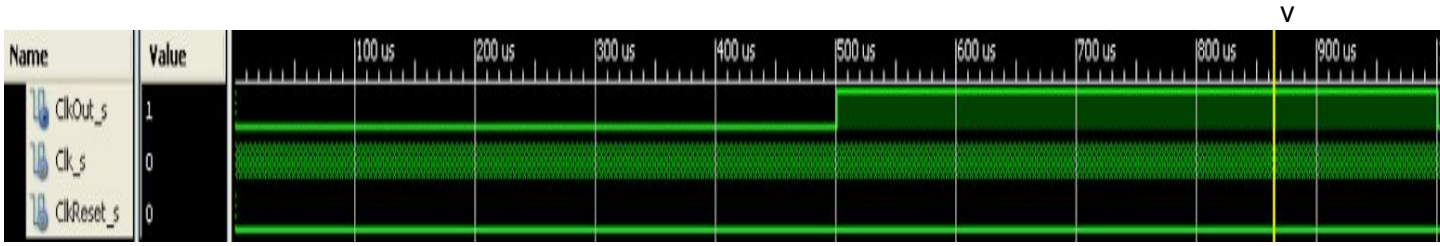


Figure 10: Waveform for PRNG

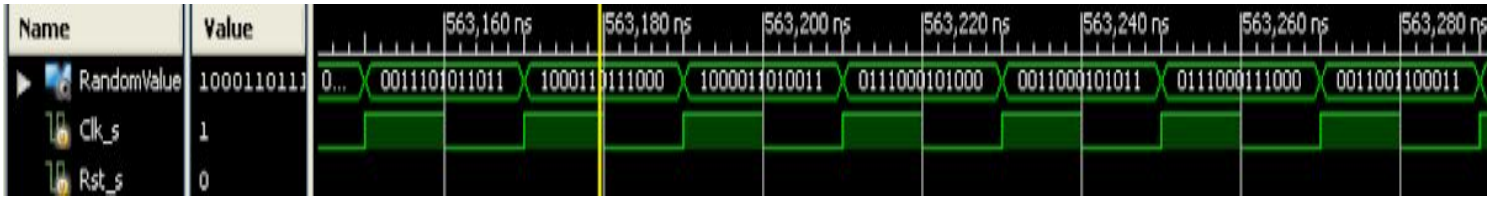
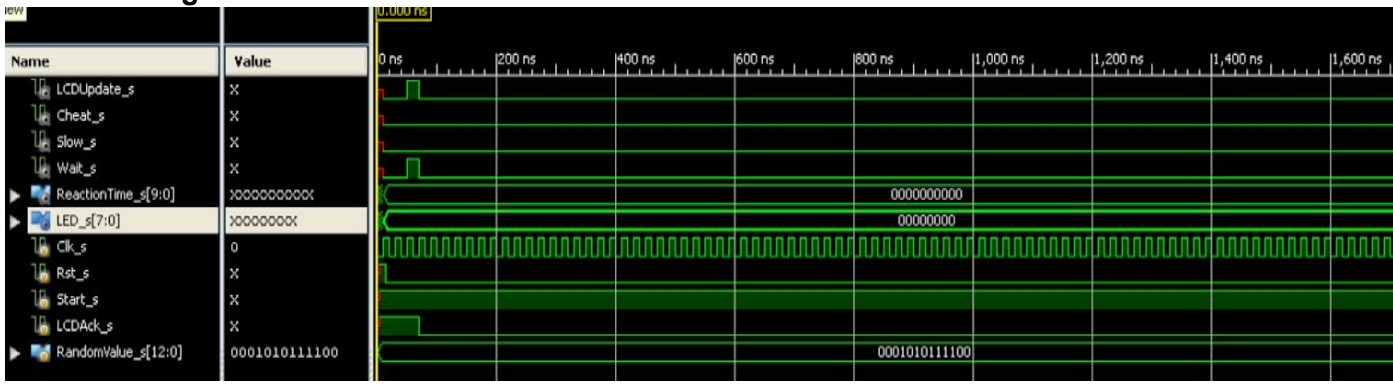
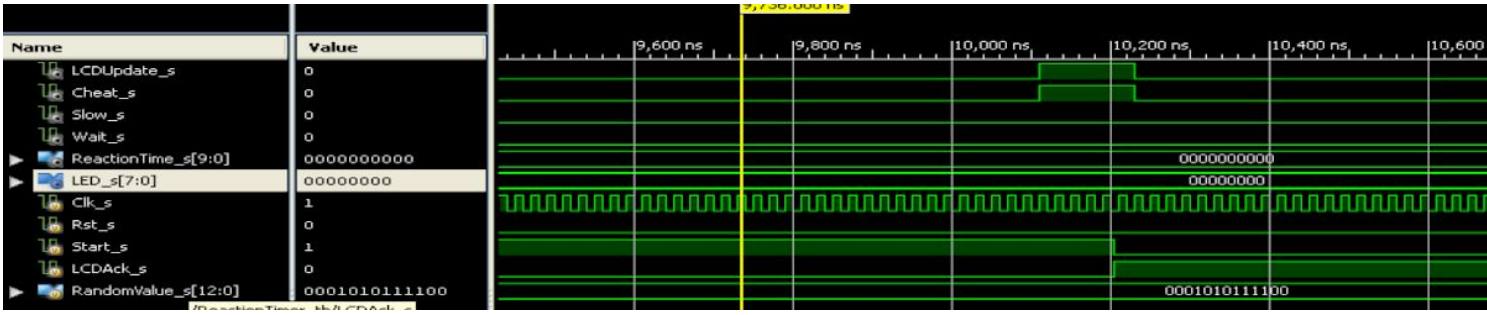


Figure 11: Waveform for ReactionTimer without random Gen

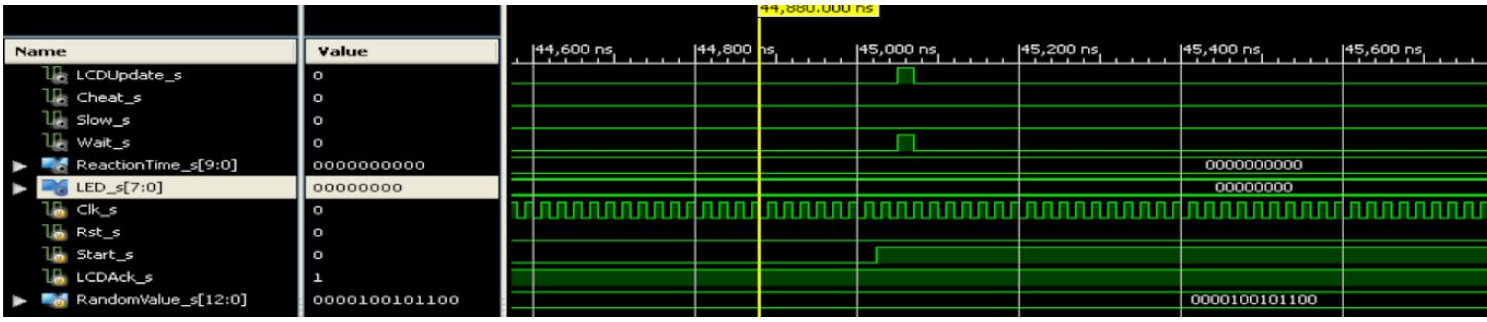


The first reset is done and the first steps toward calculating the WaitCnt is done toward the cheat section

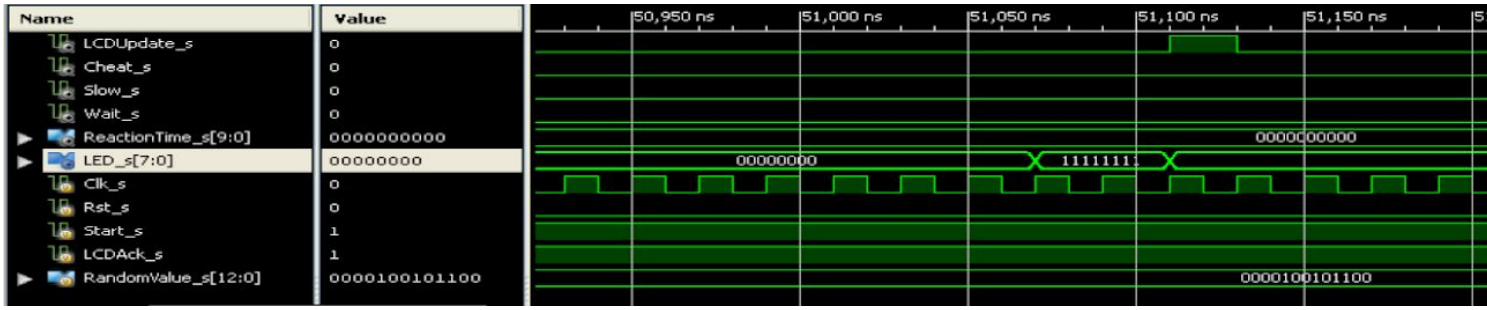


Cheat has been accomplished and now we are in the end wait state and will do another measurement

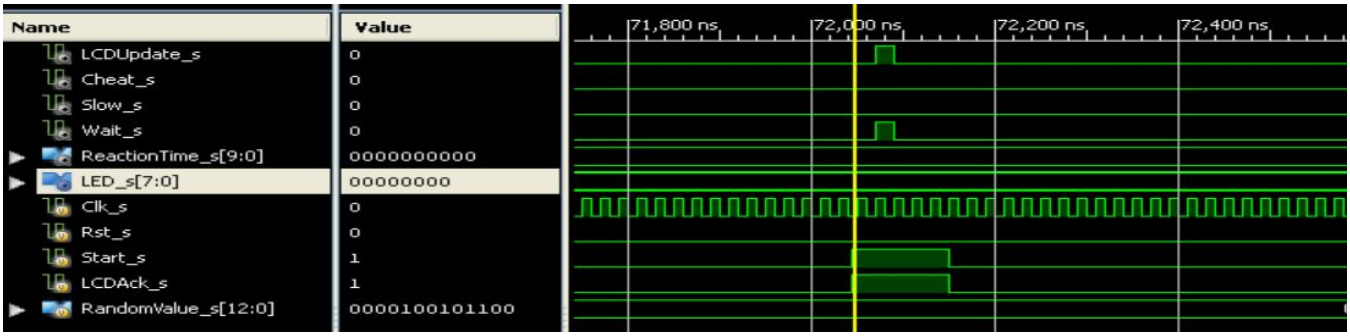
The randomValue value is then changed a short while later to do a measurement that will not be caught by another state



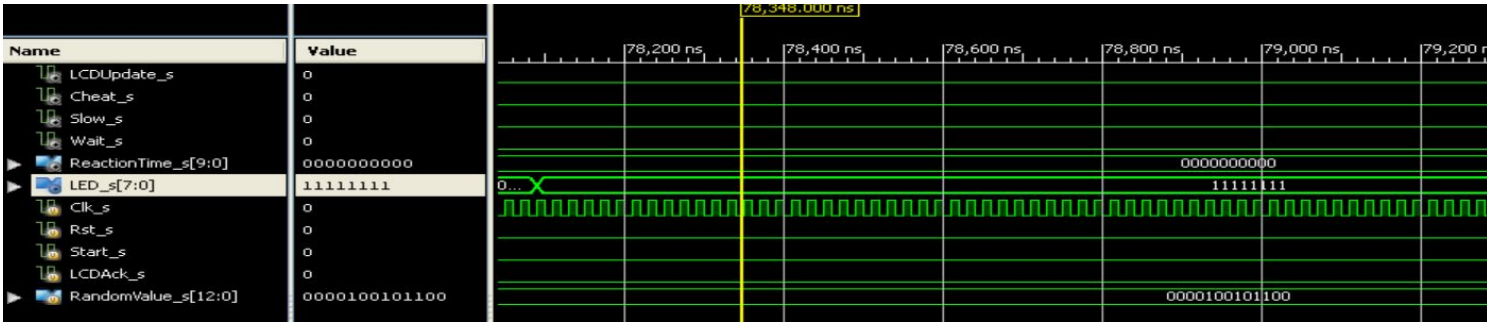
We reached the waiting state and are now in the random state



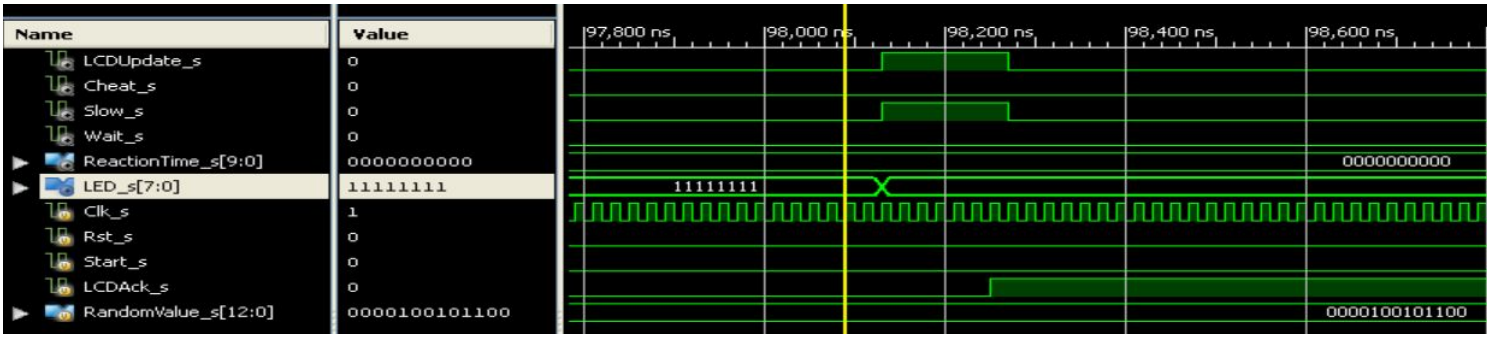
The LED lights up to indicate that the waitcnt reached the time needed to wait and LCDUpdate goes high to indicate that the time has been recorded.



The testbench made it back to the waiting state after recording and finishing. Now it is on to testing the slow function

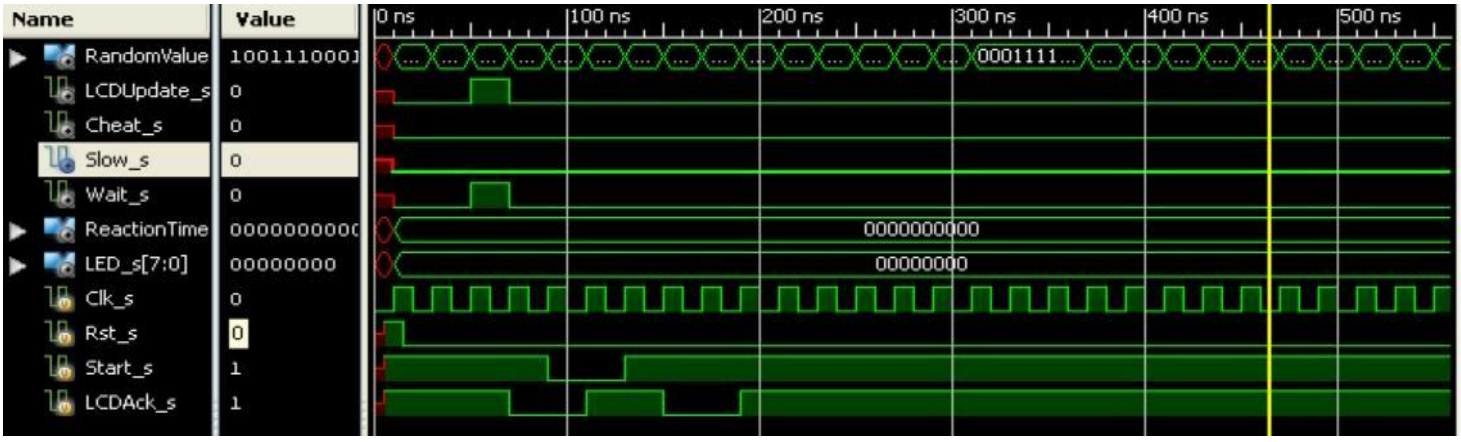


The LED has become lit to indicate we are in the measuring state

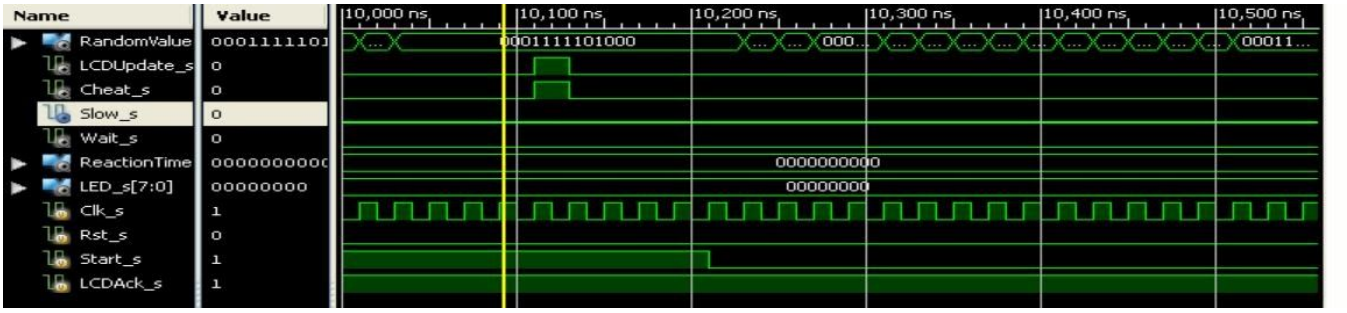


The LED turns off and the slow state is enacted and LCDAck is set back to high to go to the end wait and initial state eventually finishing the testing

B) ReactionTimer with RandomGen

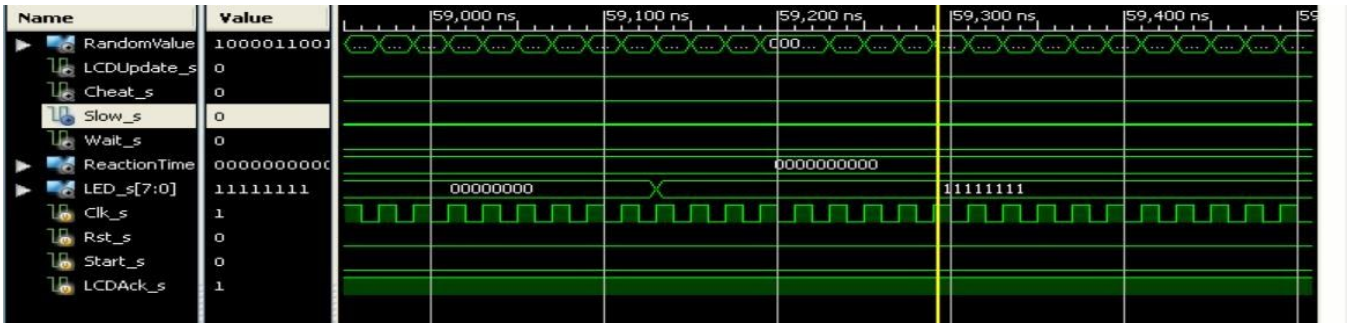


FIrst we start by getting to our wait state and starting our first test for cheat



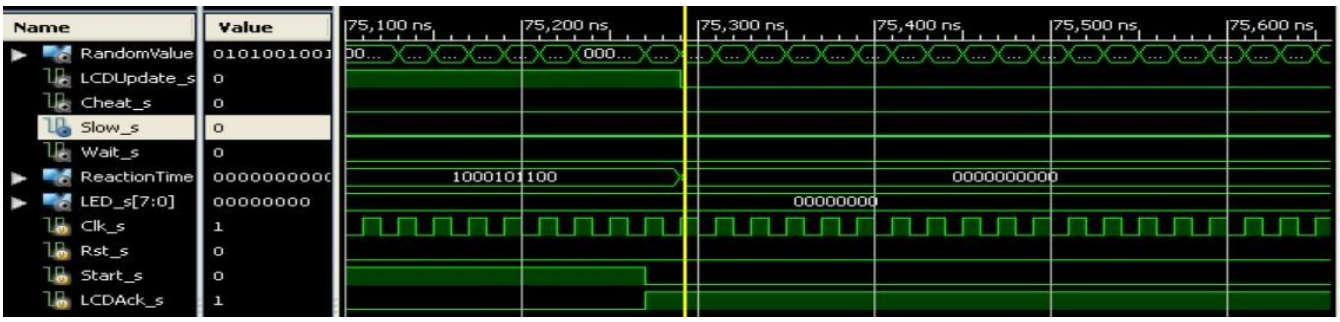
We get to cheat state so we exit through the end state and come back to wait eventually

This is the wait state where we will now attempt to record the reaction time

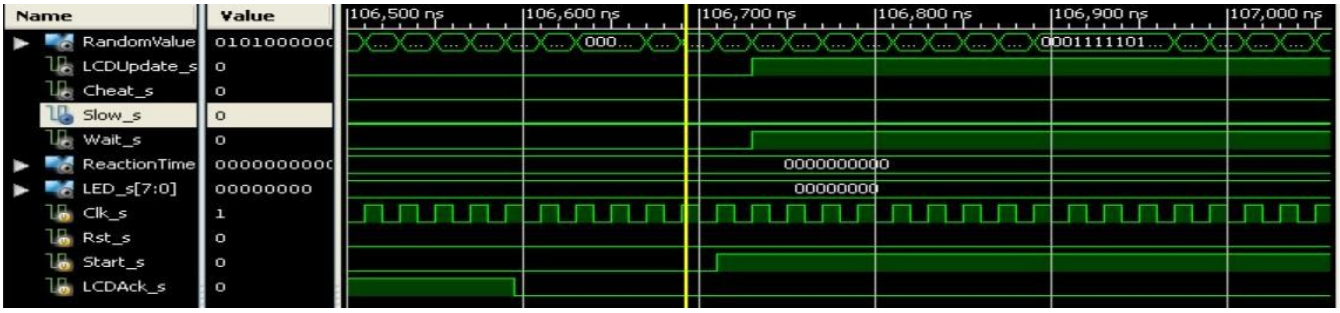


We successfully got to the LED turning on state

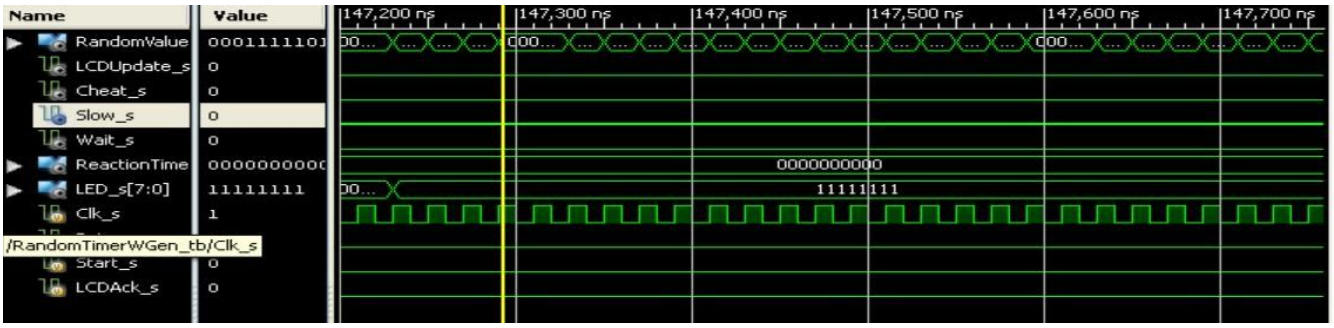
Reaction timer is set now and we update the LCD



We now end that state to test our last state of being too slow

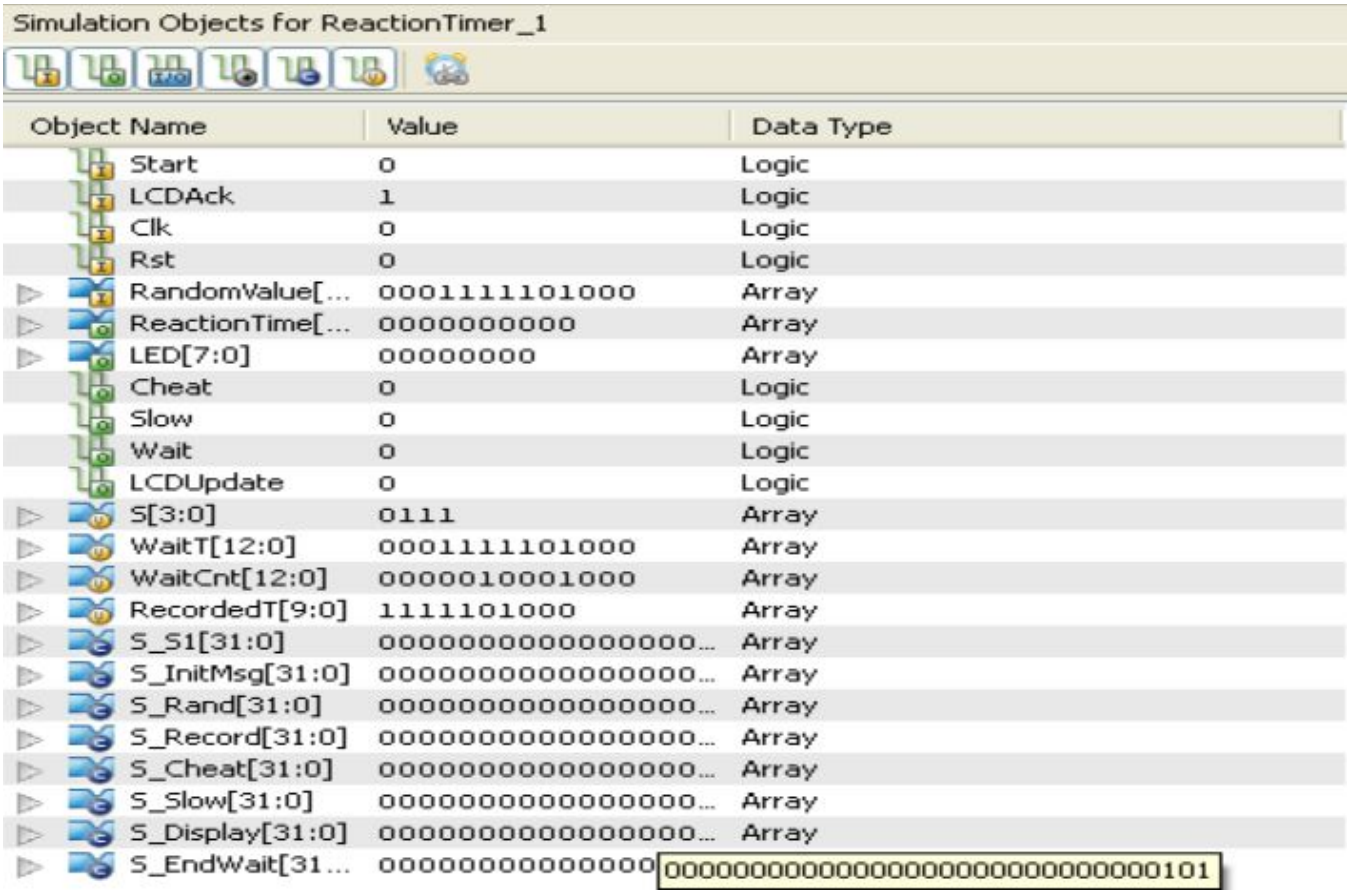


We get back to the wait state to prepare for slow



We are now measuring the reaction time

We were too slow so it displayed the slow state



This last image shows my Wait T being the same as the RandomValue which was shown to change alot during the simulation proving it used a pseudo randomly generated value as the timer

4. CONCLUSIONS

The HLSM Design was implemented in verilog along with the clock divider and the random number generator. These modules worked flawlessly as can be seen by the test benches which showed all their functionality. This final project helped teach the importance of keeping track of variables and creating effective test benches.