

Final Project: Reaction Timer

Course: Design of Digital Systems CSEE 4270

University of Georgia

Date: November 2025

1. Student Information

Names: Jordan Williams

Emails: jrw22710@uga.edu

Contributions:

- Pre-lab design and analysis: 100%
- In-lab module and testbench design: 100%
- In-lab testbench simulation and analysis: 100%
- FPGA synthesis and analysis: 100%
- Report writing: 100%

2. Lab Purpose

The purpose of this final project is to design, implement, and simulate a Reaction Timer using Verilog on the Basys3 FPGA board. The Reaction Timer measures the user's response time between LED activation and a button press, while displaying information on an LCD module. The design demonstrates the use of High-Level State Machines (HLSM), clock division, debouncing, random delay generation, and synchronization with an LCD display.

3. Implementation Details

The Reaction Timer system consists of three major custom modules: ClkDiv, RandomGen, and ReactionTimer. These modules interact to generate a stable 1 kHz clock, produce a random delay between 1–3 seconds, and control the overall timing sequence through an HLSM.

- ClkDiv: Generates a 1 kHz clock signal from a 100 MHz input clock using a 17-bit counter.
- RandomGen: Generates pseudo-random delays between 1000–3000 ms (1-3 seconds) by incrementing a counter in steps of 10 ms.
- ReactionTimer: Implements the main FSM to handle reset, waiting for a random delay, lighting LEDs, measuring user reaction time, and identifying cheating or slow responses. The module also manages LCD handshaking signals using LCDAck and LCDUpdate.

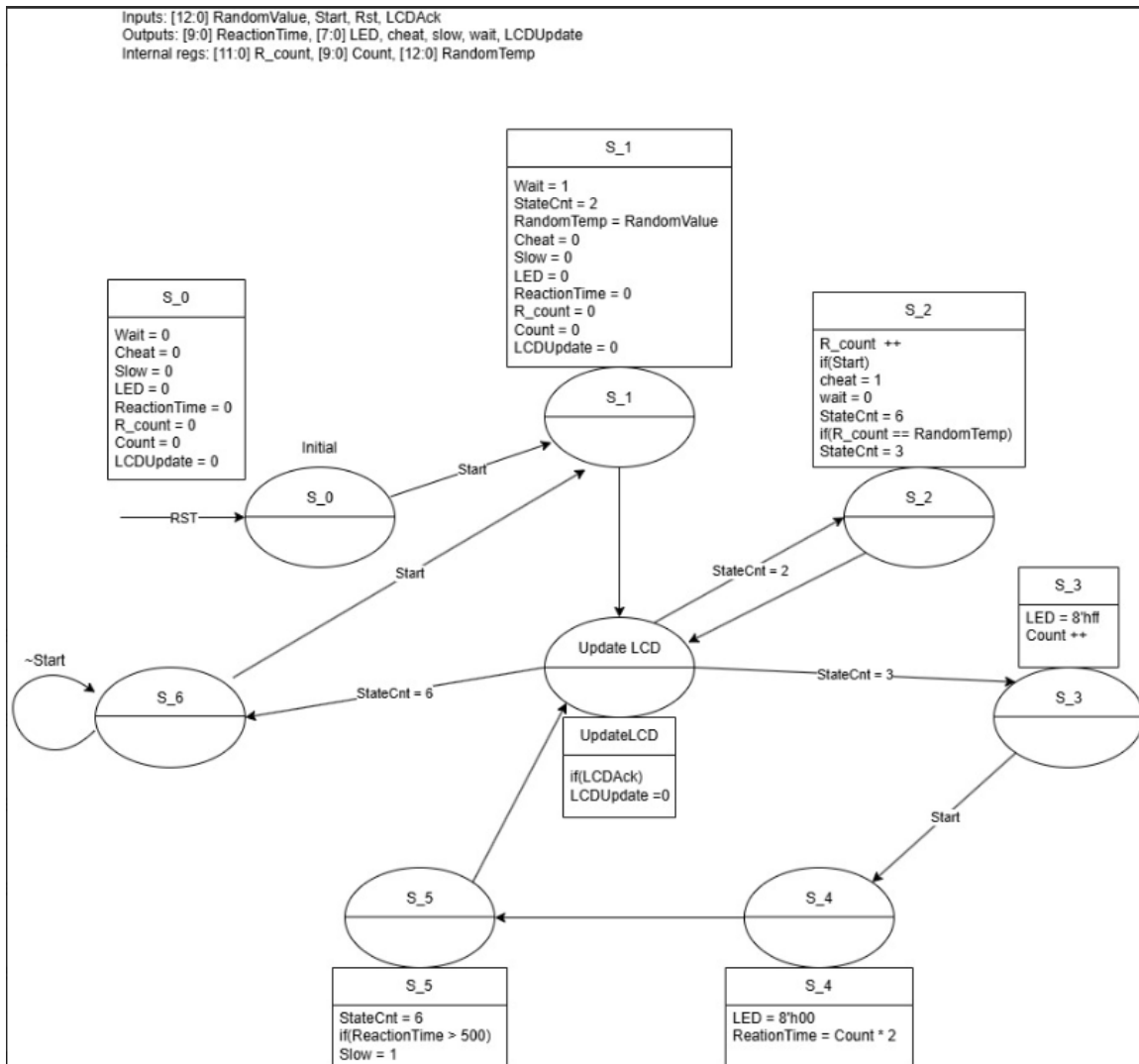


Figure 1: Reaction Timer State Diagram

- Verilog Modules:

ClkDiv.v

Generates a 1 kHz clock from a 100 MHz source by dividing the frequency by 100,000.

```

module ClkDiv(Clk, Rst, ClkMS);
    input Clk, Rst;
    output reg ClkMS = 0;
    reg [16:0] count = 0;
    parameter CLK_Div = 100_000;
    always @(posedge Clk) begin
        if (Rst) begin
            count <= 0;
            ClkMS <= 0;
        end
    end
endmodule

```

```

    end else begin
        if (count >= (CLK_Div/2 - 1)) begin
            count <= 0;
            ClkMS <= ~ClkMS;
        end else count <= count + 1;
        end
    end
end
endmodule

```

RandomGen.v

Provides a pseudo-random time delay by cycling between 1000–3000 ms.

```

module RandomGen(Clk, Rst, RandomValue);
    input Clk, Rst;
    output [12:0] RandomValue;
    reg [12:0] Counter;
    parameter MAX_mS = 3000, MIN_mS = 1000;
    always @(posedge Clk) begin
        if (Counter < MAX_mS && ~Rst) Counter <= Counter + 10;
        else Counter <= MIN_mS;
    end
    assign RandomValue = Counter;
endmodule

```

ReactionTimer.v

The ReactionTimer FSM includes states for initialization, waiting, random delay counting, reaction timing, slow and cheat detection, and LCD handshaking. A debounce circuit ensures single button activation, and the LCD handshake protocol for display updates.

Key FSM States:

S_0: Reset and initialize variables

S_1: Display 'Wait for LEDs...'

S_2: Count random delay and detect cheating

S_3: Turn on LEDs and measure reaction

S_4: Calculate reaction time

S_5: Determine slow or valid response

S_6: Wait for next start

Update_LCD / WaitAck: Handle LCD communication handshake

```

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

```

```

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;

parameter S_0 = 0, //State 0
           S_1 = 1, //State 1
           S_2 = 2, //State 2
           S_3 = 3, //State 3
           S_4 = 4, //State 4
           S_5 = 5, //State 5
           S_6 = 6, //State 6
           Update_LCD = 7, //State 7
           WaitAck = 8; // State 8

// INTERNAL REGISTERS
reg [9:0] Count = 0; // used to calculate ReactionTime
reg [11:0] R_count = 0; // used to count to random gen value
reg [12:0] RandomTemp; // used to latch the random gen value

// regs used to debounce start button
reg start_sync_0 = 0; // sync buffer
reg start_sync_1 = 0; // sync buffer
reg start_db = 0; // debounced level
reg start_db_prev = 0; // previous debounced level
reg start_pulse = 0; // one clock pulse on rising edge

parameter integer DEBOUNCE_TICKS = 10; // 1kHz ticks for debounce buffer
reg [19:0] DebounceCnt = 0; // for counting to debounce ticks

integer State = 0; // current state
integer StateCnt = 0; // holds the next state after LCD is updated

always @(posedge Clk) begin
    // DEBOUNCER AND SYNCHRONIZER
    // two flop synchronizer to "cross clock domains" from user input
    start_sync_0 <= Start;
    start_sync_1 <= start_sync_0;

    //require stable high for "DEBOUNCE_TICKS" number of ticks
    if (start_sync_1) begin
        if (DebounceCnt < DEBOUNCE_TICKS) DebounceCnt <= DebounceCnt + 1;
        if (DebounceCnt >= DEBOUNCE_TICKS) start_db <= 1;
    end else begin
        DebounceCnt <= 0;
        start_db <= 0;
    end
end

```

```

// produce a single pulse on rising edge
start_pulse <= (start_db & ~start_db_prev); // detect edges
start_db_prev <= start_db;

if (Rst) begin
    State <= S_0;
    // clear internal registers and debounce state on reset
    Count <= 0;
    R_count <= 0;
    DebounceCnt <= 0;
    start_sync_0 <= 0;
    start_sync_1 <= 0;
    start_db <= 0;
    start_db_prev <= 0;
    start_pulse <= 0;
end
else begin
    case(State)
    S_0: begin //initialize start values
        Wait <= 0;
        R_count <= 0;
        Count <= 0;
        LED <= 8'h00;
        Cheat <= 0;
        Slow <= 0;
        ReactionTime <= 0;
        LCDUpdate <= 0;

        if(start_pulse) begin
            RandomTemp <= RandomValue; // latches in random value
            State <= S_1;
        end
    end
    S_1: begin // display wait
        R_count <= 0;
        Count <= 0;
        LED <= 8'h00;
        Cheat <= 0;
        Slow <= 0;
        ReactionTime <= 0;
        Wait <= 1;
        StateCnt <= 2;
        State <= Update_LCD;
    end
    S_2: begin // count to random value
        if(R_count >= RandomTemp) begin
            Wait <= 0;
            StateCnt <= 3;
            State <= Update_LCD;
        end
    end
end

```

```

        else if(start_pulse) begin // display cheat if start pressed in this state
            Wait <= 0;
            Cheat <= 1;
            StateCnt <= 6;
            State <= Update_LCD;
        end
        else R_count <= R_count+1;
    end
S_3: begin // light LEDs and increment reaction timer
    LED <= 8'hFF;
    Count <= Count +1;
    if(start_pulse) State <= S_4;

end
S_4: begin // calculate reaction time
    LED <= 8'h00;
    ReactionTime <= Count << 1; // = count * 2
    State <= S_5;
end
S_5: begin
    if(ReactionTime < 500) begin
        StateCnt <= 6;
        State <= Update_LCD;
    end
    if(ReactionTime > 500) begin // display slow
        Slow <= 1;
        StateCnt <= 6;
        State <= Update_LCD;
    end
end
S_6: begin
    if(start_pulse) State <= S_1; // after running wait for start press
end
Update_LCD: begin // updates LCD
    LCDUpdate <= 1;
    if (LCDAck) begin
        LCDUpdate <= 0;
        State <= WaitAck;
    end
end
WaitAck: begin // waits for LCDAck
    if (~LCDAck) State <= StateCnt;
end
endcase
end
end
endmodule

```

- Testbench code:

tb_ReactionTimer_with_random.v

```
module tb_ReactionTimer_with_random;
```

```
reg Clk = 0;
reg Rst = 0;
reg Start = 0;
reg LCDAck = 0;
wire [12:0] RandomValue;
```

```
wire [7:0] LED;
wire [9:0] ReactionTime;
wire Cheat;
wire Slow;
wire Wait;
wire LCDUpdate;
```

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

```
always #5 Clk = ~Clk;
```

```
initial begin
  repeat(5)@(posedge Clk);
  #2 Rst = 1;
  repeat(10)@(posedge Clk);
  #2 Rst = 0;
  repeat(10)@(posedge Clk);
  // TEST 1: CHEAT
  #2 Start = 1;
  repeat(12)@(posedge Clk);
  #2 Start = 0;
  repeat(10)@(posedge Clk); // during wait period
  #2 Start = 1;
  repeat(12)@(posedge Clk);
  #2 Start = 0;
  repeat(20)@(posedge Clk);
  //TEST 2: MEASURE REACTION TIME
```



```

#2 Start = 1;
repeat(12)@(posedge Clk);
#2 Start = 0;
@(posedge LED);
repeat(10)@(posedge Clk);
#2 Start = 1;
repeat(12)@(posedge Clk);
#2 Start = 0;
repeat(20)@(posedge Clk);
// TEST 3: SLOW
#2 Start = 1;
repeat(12)@(posedge Clk);
#2 Start = 0;
@(posedge LED);
wait (uut.Count == 12'h140);
#2 Start = 1;
repeat(12)@(posedge Clk);
#2 Start = 0;
repeat(20)@(posedge Clk);
#2 Rst = 1;
repeat(10)@(posedge Clk);
#2 Rst = 0;
repeat(10)@(posedge Clk);
end

```

```

initial begin
  @(posedge Clk);
  forever begin
    // Simulate LCD handshake
    @(posedge LCDUpdate);
    #10;
    LCDAck = 1;
    repeat(2)@(posedge Clk);
    LCDAck = 0;

    end
  end
endmodule

```

tb_RandomGen.v

```

module tb_RandomGen();

  reg Clk;

```

```

reg Rst;
wire [12:0] RandomValue;

RandomGen uut(Clk, Rst, RandomValue);

initial Clk = 0;
always #5 Clk = ~Clk;

initial begin
    Rst = 0;

end
endmodule

tb_ClkDiv.v
module tb_ClkDiv();

    ClkDiv #(.CLK_Div(1))uut(Clk, Rst, ClkOut);

    reg Clk = 0;
    reg Rst = 0;
    wire ClkOut;

    initial Clk = 0;
    always #5 Clk = ~Clk;

    initial begin
        Rst = 0;
        repeat(50)@(posedge Clk);
        Rst = 1;
        repeat(10)@(posedge Clk);
        Rst = 0;
        repeat(1_000)@(posedge Clk);

    end
endmodule

```

4. Experimental Results

The testbench (tb_ReactionTimer_with_random) verified the ReactionTimer and RandomGen modules. It simulated three key scenarios: cheat detection, valid reaction, and

slow response. The LCD acknowledgment signal (LCDAck) was toggled manually to emulate LCD handshaking. All state transitions were confirmed in the waveform output.

- Test 1 (Cheat): Start button pressed before LEDs activated; LCD displays 'No Cheating!'.
- Test 2 (Reaction): LEDs illuminated and button pressed promptly; displays measured time such as '0.345s'.
- Test 3 (Slow): User pressed after >500 ms; displays 'Too Slow!'.

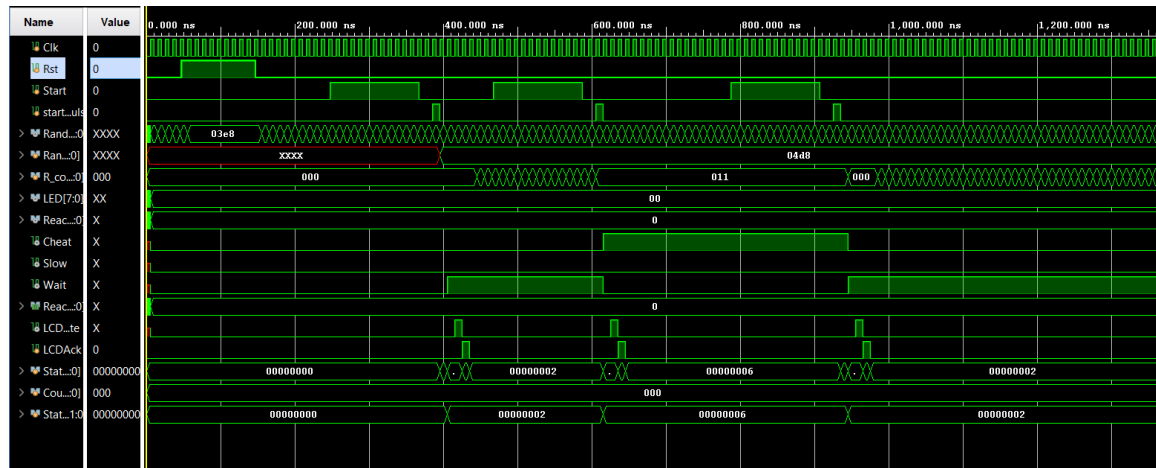


Figure 2: tb_Reaction_with_random Triggering CHEAT

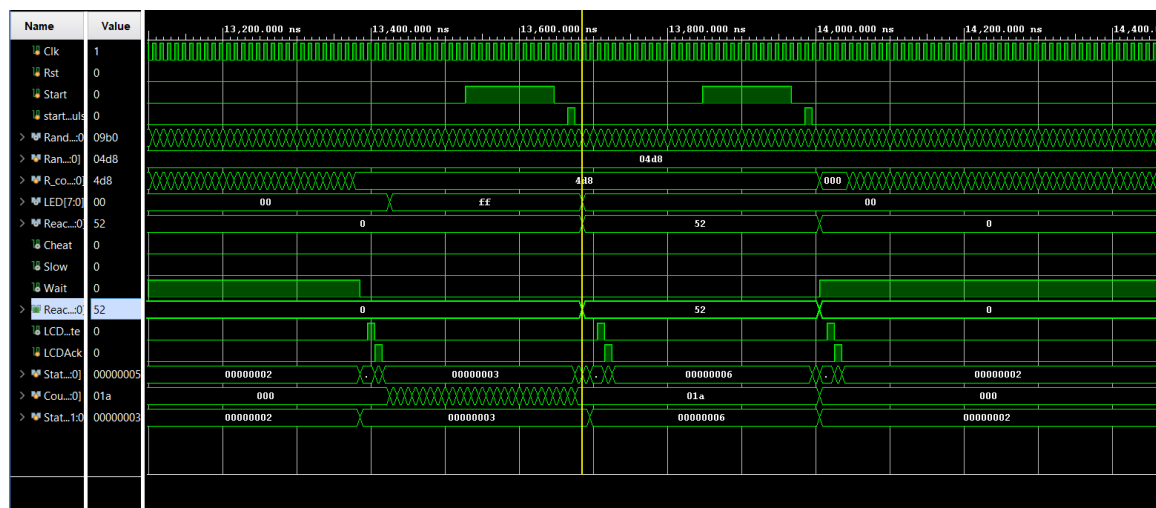


Figure 3: tb_Reaction_with_random Measuring Reaction Time

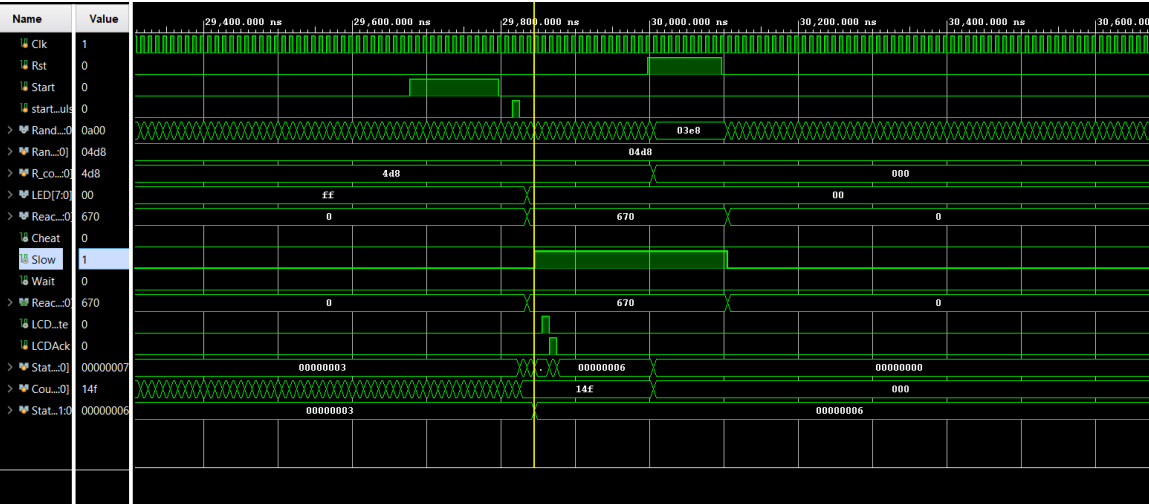


Figure 4: tb_Reaction_with_random Triggering SLOW then Rst

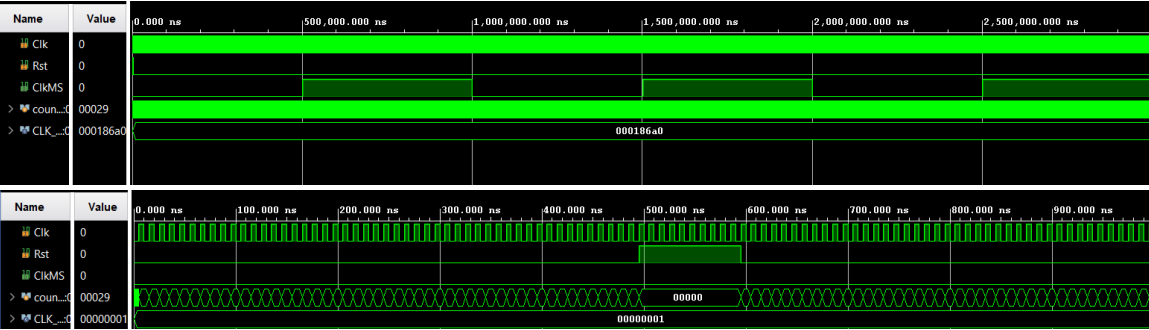


Figure 5: tb_ClkDiv waveforms

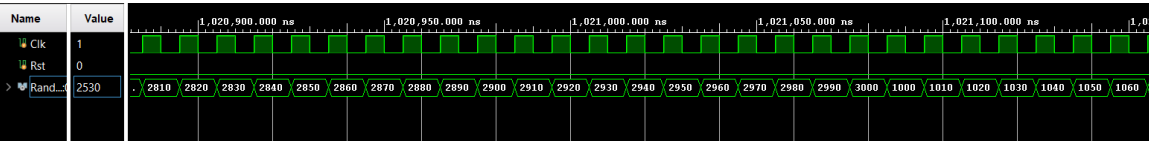


Figure 6: tb_RandomGen counting to 3000ms and resetting to 1000ms

5. Significance

This project demonstrates practical application of FSM design, input debouncing, clock division, and communication between modules. It highlights how synchronous digital logic can coordinate human input timing with hardware displays, which is valuable in embedded system design.

