Health Monitor System - Technical Report

Angela Busheska & Laxman Poudel

Digital Circuits I

Prof. Naga Spandana Muppaneni

05/02/22

1. Abstract:

A health monitor system is a piece of essential equipment to keep track of individual health for their healthy and prosperous life. The report presents the design of a health monitor system implemented on an FPGA Board, including a high-level description of the invention, details on the internal operation, and information on the system performance, as well as its validation. The report shows that the final system meets all requirements of the specification.

2. Introduction:

Living a healthy lifestyle can help individuals prevent chronic diseases and long-term illnesses. As the fast-paced living rate increases, people give a lot of emphasis on health and lifestyle. Due to monopoly pricing, prescription drugs are expensive, and people prefer to remain healthy rather than get cured. So to reduce suffering and medical costs, the health monitor system is designed. Health monitors are equipped with different sensors that allow us to read the user's heartbeat in real-time, and are used in some of the most popular products of today's world: Apple Watch and Fitbit. In this project, we are using an FPGA in order to design a health monitor module.

The specifications of the health monitor system include:-

● A pulse monitor that reads the heartbeat of its user every 5 seconds.
● A seven-segment monitor to display the average measured heartbeat.
● A reaction timer to indicate the time in the seven-segment display.
● An option for the user to toggle between two modes via a switch: reaction timer or pulse counter to present simple controls such as a start, enter, reset, and LED signals.

The technical report will examine the operation of the health monitor by first exploring the high-level organization and then individual submodules.

3. System Design

3.1 High-Level Design

Figure 1 demonstrates a diagram of the complete health monitor. This module presents the connection between the pulse monitor and the reaction timer. It uses a switch mode to select between the reaction timer and pulse monitor. We can choose which functionality we are going to use through SW0. The output of each operation will be displayed in a 7-segment display as well as in a tri LED.

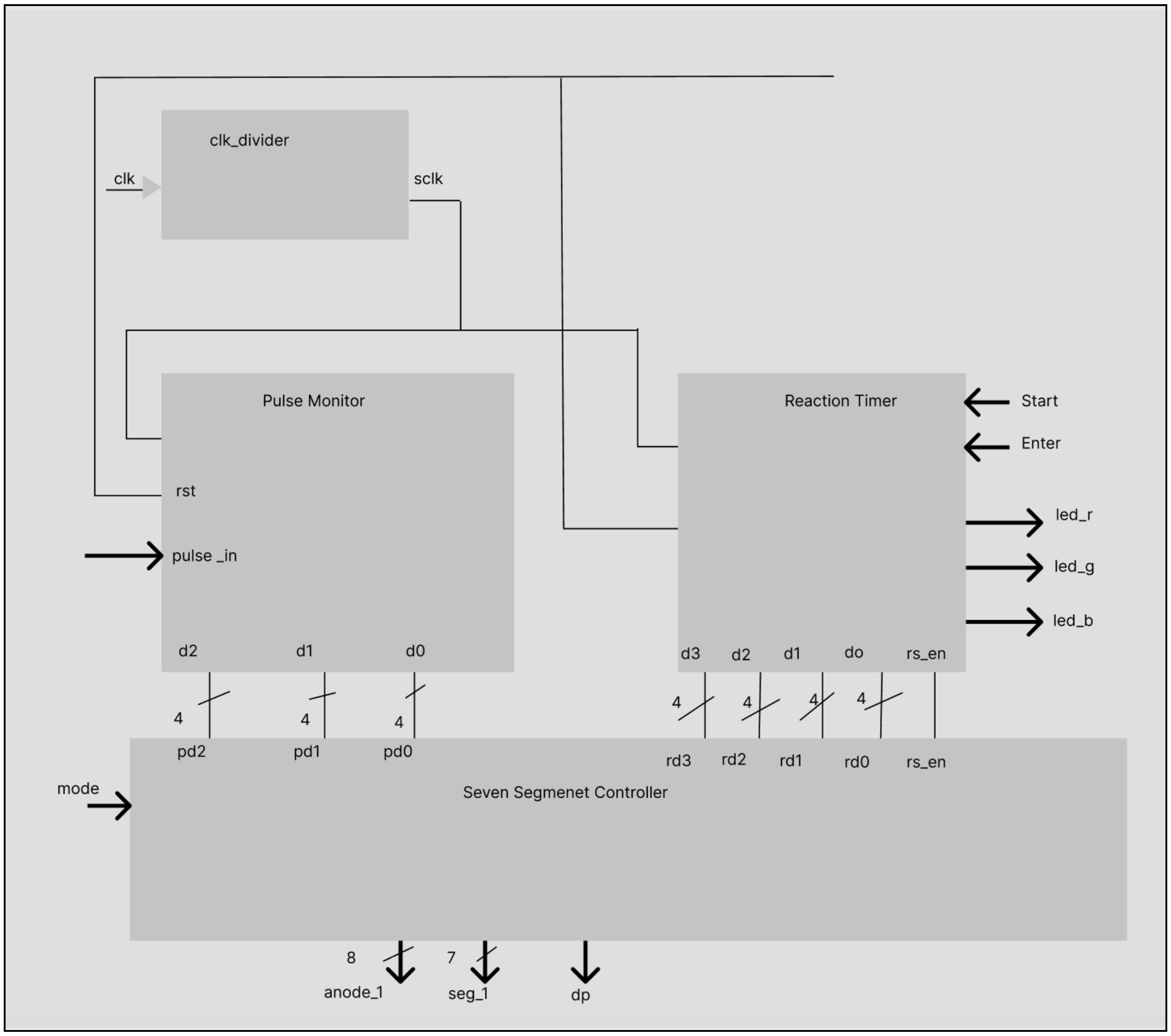

Figure 1: High-Level Design

3.2 Implementation

This section demonstrates the implementation of the modules to make a health monitor function. Combining all these modules at the top level we were able to implement them on an FPGA board.

3.3.1 Top-Level Module - Health Monitor System

3.3.1.1 **Inputs:**

- clk: The 100Mhz clock is provided by an external oscillator on the development board.
- reset. A push-button input that resets the health monitor back to an initial state.
- pulse_in: A signal read by a sensor that reads the user's heartbeat.

- start: A push-button input that starts the reaction timer operation.
- enter: A push-button input that performs the reaction of the user.

3.3.1.2 **Outputs:**

- led_r: Output signal to control the red color of an LED.
- led_g: Output signal to control the green color of an LED.
- led_b: Output signal to control the blue color of an LED.

```
module health_monitor_top_level( input logic clk100Mhz, rst, start,
enter, pulse_in, mode,
                       output logic [7:0] an_l ,
                       output logic [6:0] segs_l ,
                       output logic dp_l, led_r, led_b, led_g);

 // Logic Declarations
   logic clk ;
   logic clk_60, clk_go ; // Coming in from a second clock divider

   logic rs_en ;

   logic [3:0] d0,d1,d2,d3 ;

   logic [3:0] r0,r1,r2,r3 ;

   logic start_debounce , start_go ;

   logic enter_debounce, enter_go ;

   logic rst_debounce , rst_go ;

   logic [15:0] pulsemon_out , reaction_out ,q;


// Clock Divider for the clk signal of the entire circuit
clkdiv #(.DIVFREQ(1000)) U_CLKDIV(.clk(clk100Mhz), .reset(1'b0),
.sclk(clk));

//Clock Divider for simulated pulse
//clkdiv #(.DIVFREQ(60)) CLOCK_60(.clk(clk100Mhz), .reset(1'b0),
.sclk(clk_60));

// Debouncer and Single Pulser  START
   debounce START_1(.clk(clk), .pb(start) ,
.pb_debounced(start_debounce));
```

```
   single_pulser START_2(.clk(clk), .din(start_debounce),
.d_pulse(start_go));

// Debouncer and Single Pulser ENTER
   debounce ENTER_1(.clk(clk), .pb(enter) ,
.pb_debounced(enter_debounce));

    single_pulser ENTER_2(.clk(clk), .din(enter_debounce),
.d_pulse(enter_go));

 // Debouncer and Single Puler RESET
  debounce RST_1(.clk(clk), .pb(rst) , .pb_debounced(rst_debounce));

  single_pulser RST_2(.clk(clk), .din(rst_debounce),
.d_pulse(rst_go));


//Creating an instance of Pulse Monitor

pulse_moniter PULSE (.clk(clk),.rst(rst_go), .pulse_in(pulse_in),
.pd0(d0), .pd1(d1), .pd2(d2), .pd3(d3));

//pulse_moniter PULSE (.clk(clk),.rst(rst_go), .pulse_in(clk_60),
.pd0(d0), .pd1(d1), .pd2(d2), .pd3(d3));

//Creating an instance of Reaction Timer
reaction_timer REACT(.clk(clk), .rst(rst_go), .start(start_go),
.enter(enter_go), .led_r(led_r), .led_g(led_g),
.led_b(led_b),.rs_en(rs_en),.d0(r0),.d1(r1),.d2(r2),.d3(r3)) ;

// Concatenating the Outputs of the Pulse Monitor and the Reaction
Timer

assign pulsemon_out = {d3,d2,d1,d0} ;
assign reaction_out = {r3,r2,r1,r0};


// Creating an instance of 16 bit 2 to 1 Multiplexer
mux_16bit_2to1 SEL(.mode(mode) , .reaction_timer(reaction_out),
.pulse_mon(pulsemon_out) , .q(q)) ;


//Showing decimal 127  will c
```

```
sevenseg_control_hm U_C_5(.clk(clk),
.rst(rst),.mode(mode),.rs_en(rs_en),.d0(q[3:0]), .d1(q[7:4]),
.d2(q[11:8]), .d3(q[15:12]), .d4(4'd0), .d5(4'd0),.d6(4'd0),
.d7(4'd0),.segs_l(segs_l), .an_l(an_l), .dp_l(dp_l));

endmodule
```

3.3.1.2 Functionalities and Design:

This health monitor was designed with two modes available to the user.

The reaction timer is dedicated to recording the reaction time of the user and it is initialized by the user pressing the start button. The pulse monitor receives a pulse signal from an analog pulse sensor that is attached to the board. The monitor counts the number of heartbeats from the user over certain intervals while maintaining the samples from each five-second trial for conversion to beats per minute. The value is then displayed on the seven-seg display.

The detailed schematic is shown in figure 1.

3.3.2 High-Level Module - Pulse Monitor

The pulse monitor shown in Figure 2 involves all the submodules found in it. The pulse input signal coming from the pulse itself passes through both, the debouncer and single pulser, and is keeping track of the number of heartbeats at each clock edge.
The counted value of the pulse counter is passed through the registers and passed over through the adders. The sum is then passed over to the bpm module to execute the number in beats per minute so that they can be displayed on the seven-segment display.

3.3.2.1 Inputs:
- Pulse_in: The pulse input from the finger
- Clk: The clock signal
- Rst: The reset signal

3.3.2.2 Outputs:
- Pd0, Pd1, Pd2, Pd3: The display values of the pulse, shown on the seven-segment display

3.3.2.3 Functionalities and Design:

Figure 2 demonstrates the functionality of the pulse monitor top module to read the heart beat of its user every 5 seconds. It consists of various module as below :

a). delay_counter : It is used to count until 5 sec before it asserts high output.
b). pulse_counter :This module is used to count the input pulse from the sensor.
c). clkdiv :This module is used to divide the frequency for pulse monitors.
d). pulse_adder : It is used to sum up the pulse every 5 sec.
e). binary_to_bcd : This module turns a binary number into a number in BCD format.
f). registers : This module is used to perform shifting action.

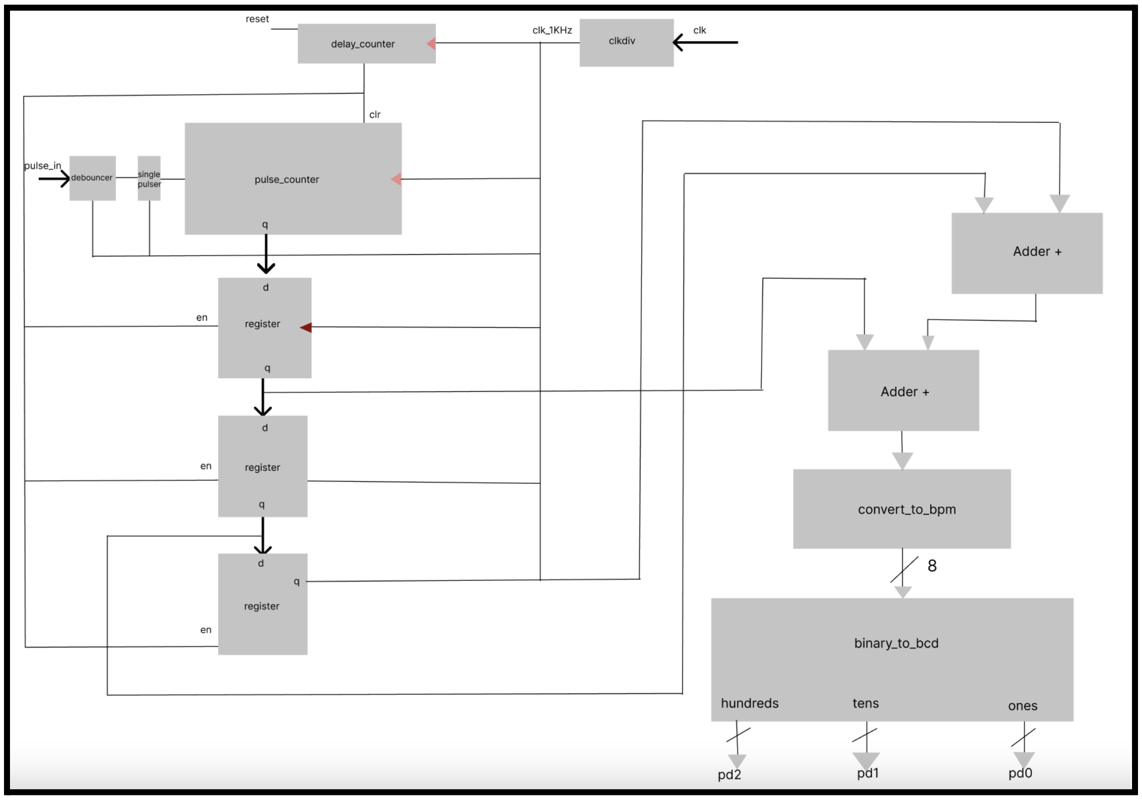

Figure 2: Pulse Monitor High-Level Design

```
module pulse_moniter( input logic clk,rst, pulse_in ,
                    output logic [3:0] pd0, pd1, pd2, pd3);

   //instantiations for the odule

    logic pulse_go;
    logic d_done; // a wire giving the output of the delay counter

  //a wire for the register
  logic[3:0] q0, q1, q2 , p_counter ;
  // adder
   logic [5:0] adder_sum;

  //bpm conversion
  logic [7:0] bpm_out;

  //Wire for 15 sec counter
  logic add_time ;

  // pd3 is connected to 0
  assign pd3 = 4'd0;

 // Single Pulser for Pulse_In input

  single_pulser S_PULSE(.clk(clk), .din(pulse_in),
.d_pulse(pulse_go)) ;

  //Creating an instance of delay counter

  delay_counter DELAY (.clk(clk), .delay_done(d_done));

    // Creating an instance of pulse counter

  pulse_counter PULSE (.clk(clk), .clr(d_done), .enb(pulse_go) ,
.q(p_counter));

  pcount_registers PCOUNT (.clk(clk),.iden(d_done), .rst(rst),
.q_in(p_counter), .c1(q0) , .c2(q1), .c3(q2)) ;


  // Instantiating the adder module to sum up the pulse
    pulse_adder ADD (.q1(q0), .q2(q1),.q3(q2), .sum(adder_sum));


  // Converting to BPM
  convert_to_bpm BPM (.sum(adder_sum), .bpm(bpm_out));

  //Instanstantiate Binary_to_bcd module

  binary_to_bcd BTBCD (.b(bpm_out), .hundreds(pd2), .tens(pd1) ,
.ones(pd0));

endmodule
```

3.3.4 Pulse Counter

The pulse counter takes the input coming from the pulse sensor. When the pulse is passed through the counter, a signal is received to pass the pulse through the registers and then further cleared for the next cycle of pulse counting.

Inputs:
- Clk: Clock Signal
- Clr: Input signal from the delay counter
- Enb: Transfers the input signal from the pulse

Outputs:
- Q: Outputs from the pulse on a five-second period

```
module pulse_counter( input logic clk, clr, enb ,
                      output logic [3:0] q ) ;


  always_ff @(posedge clk)

  if (clr) q <= 0 ;
    else if (enb) q <= q+1 ;

endmodule
```

3.3.5 Delay Counter

The delay counter is a counter that counts up. When it receives a delay signal, it is being asserted high for the pulse counter to reset its counter.

Inputs:
- Clk: Clock Signal

Outputs:
- Delay_Done: Signal of the counter after 5 seconds

```
module delay_counter( input logic clk,
                      output logic delay_done);

   // Logic Instantiation
  logic [12:0] q;

  always_ff @(posedge clk)
       begin
```

```
            // Increment the counter by 1
              q <= q +1;
           if (q == 13'd5000)
             begin
             // 5 sec counter is up and delay done is asserted to be
1
                 delay_done <= 1 ;
                 q <= 0;
             end

         else
            delay_done <= 0
       end

endmodule
```

3.3.6 Binary-To-BCD

The Binary-to-BCD counter turns a binary number into a number in BCD format.

Inputs: b: Receives the input from the existing minute to bpm transform module. **Outputs**: hundreds, tens, ones: Outputs the final result, which is displayed on the seven-segment display

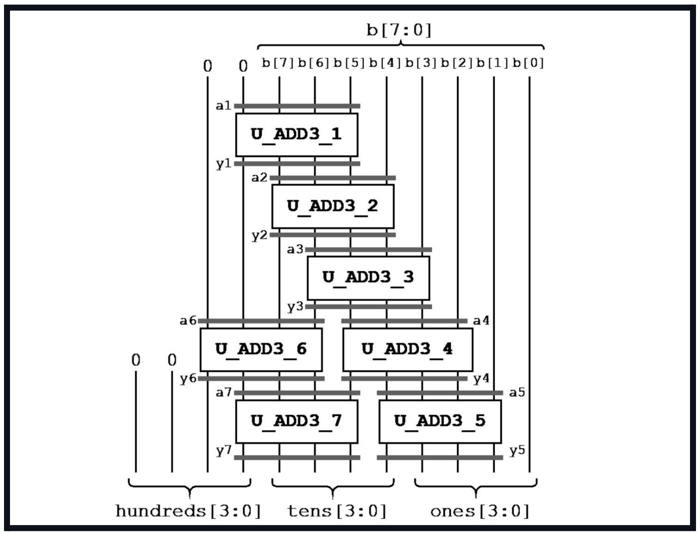

Figure 3: Binary to BCD Schematic

```
module binary_to_bcd ( input logic [7:0] b,
output logic [3:0] hundreds,
output logic [3:0] tens,
output logic [3:0] ones );

// Logic Instantiations
```

```
logic [3:0] a1, a2, a3, a4, a5, a6, a7;
logic [3:0] y1, y2, y3, y4, y5, y6, y7;

//Creating instances of add3
add3 U_ADD3_1 (.a(a1), .y(y1));
add3 U_ADD3_2 (.a(a2), .y(y2));
add3 U_ADD3_3 (.a(a3), .y(y3));
add3 U_ADD3_4 (.a(a4), .y(y4));
add3 U_ADD3_5 (.a(a5), .y(y5));
add3 U_ADD3_6 (.a(a6), .y(y6));
add3 U_ADD3_7 (.a(a7), .y(y7));
assign a1 = {1'b0, b[7:5]};
assign a2 = {y1[2:0],b[4]};
assign a3 = {y2[2:0],b[3]};
assign a4 = {y3[2:0],b[2]};
assign a5 = {y4[2:0],b[1]};
assign a6 = {1'b0,y1[3],y2[3],y3[3]};
assign a7 = {y6[2:0],y4[3]};
assign ones = {y5[2:0],b[0]};
assign tens = {y7[2:0],y5[3]};
assign hundreds = {2'b0,y6[3],y7[3]};
endmodule
```

3.3.7 Count Registers

This module connects the three registers that represent the shift registers. The user pulse count is inputted every five seconds. The previously recorded pulse value is shifted to the next register. These values after being converted to beats per minute.

Inputs:
- clk: Clock signal
- delay: signal of delay counter, serves as an enable
- rst: Reset signal

Outputs:
- c1,c2,c3: outputs fed into the respective registers and used in the adders.

```
module count_registers( input logic clk, delay,rst,
                        input logic [3:0] q_in,
                        output logic[3:0] c1,c2,c3 );

//Instantiating Registers

p_register R3 (.clk,.delay,.rst(rst), .d(q_in),.q(c3)) ;
p_register R2 (.clk,.delay,.rst(rst), .d(c3),.q(c2)) ;
p_register R1 (.clk,.delay,.rst(rst), .d(c2),.q(c1)) ;

endmodule
```

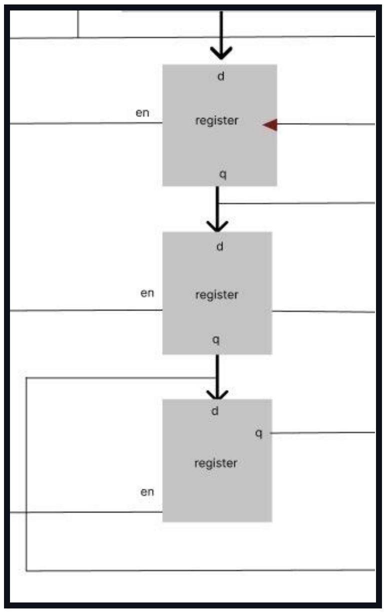

Figure 4: Count Registers

3.3.8 Pulse Adder

The pulse adder takes in the output values from the shift register, and calculates their sum. The first adder sums up the first two values, stored as a 5-bit input. The next set of adders sums up the using the sum of the first value and the third 3-bit input value.

Inputs:
- q1,q2,q3: The signals from the registers

Outputs:
- c1,c2,c3: The outputs in the respective bit size

```
module pulse_adder( input logic [3:0] q1, q2, q3,

                     output logic [5:0] sum);

 // Wires for adder

 logic [4:0] firstadd;

 logic [5:0] secondadd;
```

```
always_comb

       begin

       firstadd = 0;

       secondadd = 0;

         //Computing first add

         firstadd = q1 + q2 ;

         //Computing second add

         secondadd = firstadd + q3 ;

       end


  //assign sum = secondadd ;

  assign sum = secondadd;

endmodule
```

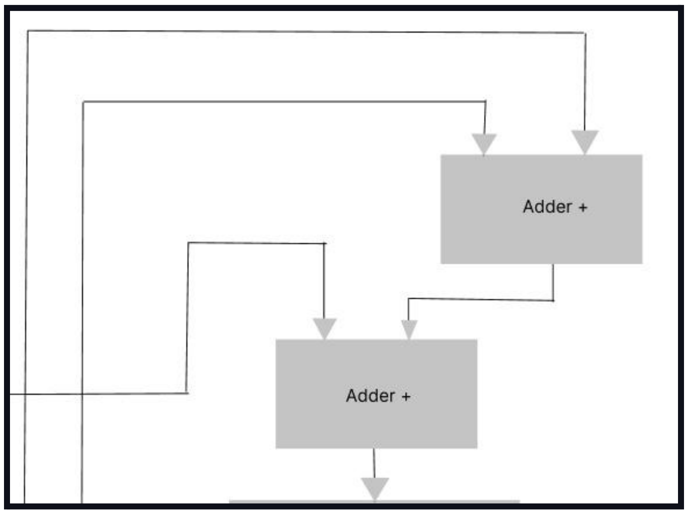

Figure 5: Pulse Adders

3.3.9 Convert to BPM

Since the pulse counter only samples three 5-second sets of pulses, we will only need to take the average of the value. The module can achieve the operation by shifting.

Inputs:
- sum: the output from the adders

Outputs:
- c1,c2,c3: The outputs converted in beats per minute

```
module convert_to_bpm( input logic [5:0] sum ,
                       output logic [7:0] bpm );

// Shifting the sum (the avg of beats for 5 sec) to convert to
bpm

assign bpm = sum<<2;

end module
```

3.3.9 High-Level Module - Reaction Timer

The function of the reaction timer module is to record the reaction time of the health monitor, and it is initialized by the user pressing the start button. After that, the user has to wait randomly for a generated amount of time between 1-9 seconds which is implemented inside the random wait of the reaction timer module. After the random delay, the GO LED is turned on and records the amount of time, and users are required to respond by pressing the enter button. The time the user takes to press the enter button while the GO-LED remains on is displayed on the seven-segment display. Different error cases are handled based on the button pressed, which is represented by an LED. Based on other conditions, a Finite State Machine was employed to deal with these cases, whose module and explanation will be mentioned in a different section of the report.

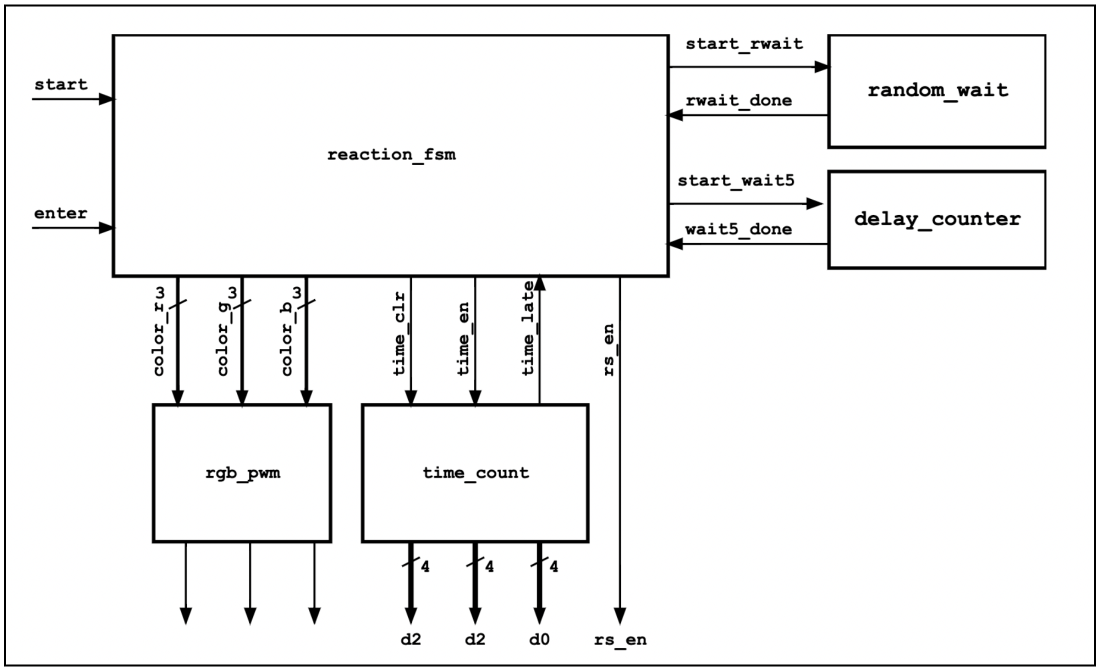

Fig 6: Top Level Organization of Reaction Timer

Inputs:

- `start`: Pressing a button to activate the start
- `enter`: Pressing a button to activate enter
- `rst`: Pressing a button to activate the reset

Outputs:

- `led_r,led_g, led_b`: red Led, green Led, blue Led: LED Colors to display the respective lights on the LED's
- `[3:0] d0,d1,d2,d3`: Displaying the time on the seven segment display

```
module reaction_timer( input logic clk ,start , enter, rst,
                       output logic led_r ,led_g , led_b, rs_en ,
                       output logic [3:0] d0,d1,d2,d3);
// Logic Instantiations
         logic start_rwait, rwait_done;
         logic start_wait5 , wait5_done ;
              logic [2:0] color_r, color_g, color_b ;
              logic time_clr, time_en, time_late ;

//Creating an instance of the reaction FSM

reaction_fsm U_INST1(.clk(clk),.rst(rst),.start(start),.enter(enter)
,.rwait_done(rwait_done), .wait5_done(wait5_done),.time_late
(time_late),.start_rwait(start_rwait),.start_wait5(start_wait5),.rs_e
n(rs_en),.time_clr(time_clr),.time_en(time_en),.color_r(color_r) ,
.color_g(color_g), .color_b(color_b));

 // Creating an instance of the random wait module
random_wait U_INST2(.clk(clk), .rst(rst), .start_wait(start_rwait),
.rwait_done(rwait_done));

// Creating an instance of the delay counter

 delay_counter_ U_INST3(.clk(clk),.rst(rst),.start_wait5
(start_wait5), .wait5_done(wait5_done));

 //Creating an instance of Time Count

time_count U_INST4(.clk(clk), .time_clr(time_clr),.rst(rst),
.time_en(time_en),.time_late(time_late), .d0(d0), .d1(d1), .d2(d2),
.d3(d3));

// Creating an instance of rgb_pwm

     rgb_pwm U_INST5(.clk(clk),.rst(rst), .color_r(color_r),
     .color_g(color_g) , .color_b(color_b), .rgb_r(led_r) ,
     .rgb_g(led_g), .rgb_b(led_b));

 endmodule
```

3.3.10 Reaction FSM

This module is used to design a finite state machine for a reaction timer. The case statement is used to draw the FSM for the reaction timer. Its initial state is the IDLE state. When the start pushbutton is launched, the idle state will transition from that previous state into the next state i.e. r_wait. The system will stay for 5 seconds in this state before launching the GO LED. The reaction timer will wait until 10 seconds before changing to a late state. Likewise, the transition from one state occurs into another state based on the inputs as shown in the state transition diagram.

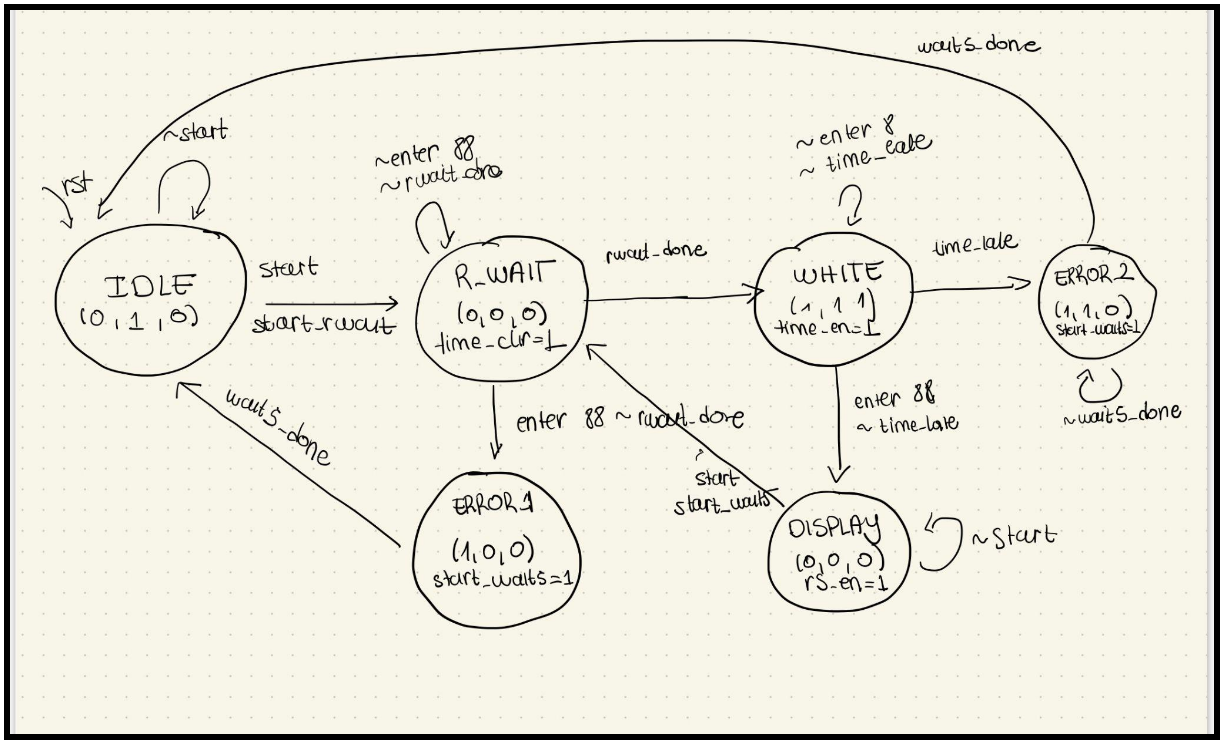

Fig 7: State transition diagram for reaction FSM

Inputs:

- `clk` : Clock signal
- `rst` : Pressing a button to activate the reset
- `start` : Pressing a button to activate the start
- `enter` : Pressing a button to activate enter
- `rwait_done` :output signal sent for reaction FSM to detect the change of state and allow the user to press the enter button to record time.
- `wait5_done` : output of the delay_counter to signal whether the five-second count has passed.
- `time_late` : output signal sent from the time count module that tells the user the 10-second time period in which they had to press the enter button.

Outputs:

- `start_rwait` : Input signal for random wait module to generate a random wait time for the user
- `start_wait5` : Input signal for the delay counter module to wait for 5 seconds
- `rs_en` : Reset signal to tell the seven segments to display the digits.
- `time_clr` : Input signal for the time count module which acts as a reset.
- `time_en` : Input signal for time count module

- `color_r , color_g , color_b` : Input pin for rgb_pwm module to display different LED for different error state of the reaction timer.

```
module reaction_fsm( input logic clk,rst, start, enter , rwait_done,
                     wait5_done, time_late,
                     output logic start_rwait, start_wait5, rs_en,
                     time_clr, time_en,
                     output logic [2:0] color_r,color_g,color_b);


 typedef enum logic [2:0]{
    IDLE= 3'b000, R_WAIT = 3'b001, ERROR_1 = 3'b010, WHITE = 3'b011,
    DISPLAY = 3'b100, ERROR_2 = 3'b101
    }state_t ;

//ERROR_1 is showing the red light

 state_t state, next ;

 always_ff @(posedge clk)

     if(rst) state <= IDLE;
     else state <= n_s;

     always_comb
          begin

               color_r = 3'b000;
               color_g = 3'b000;
               color_b = 3'b000;

               start_rwait = 0 ;
               start_wait5 = 0 ;
               time_clr = 0;
               rs_en = 0 ;
               time_en = 0;

               n_s = IDLE;

 // Creating case statements

               case(state)
```

```
                    IDLE:
                         begin
                              color_r = 3'b000;
                              color_g = 3'b001;
                              color_b = 3'b000;
                              rs_en = 0 ;
                              time_clr = 0;
                              time_en =0;
                              start_rwait =0;
                              start_wait5 =0;

//Initiating the random wait
                         if(start)
                                   begin
                                        start_rwait = 1 ;
                                        next = R_WAIT ;
                                   end

                         else if(~start)
                              n_s = IDLE;

//when the start button is not activated, we are in the IDLE state
                         end

                    R_WAIT:
                         begin

                              time_clr = 1 ;

                              // LEDS are off in Random Wait
                              color_r = 3'b000;
                              color_g = 3'b000;
                              color_b = 3'b000;
                              rs_en = 0 ;
start_rwait = 0;
start_wait5 = 0;
time_en = 0;


                               if (enter && ~rwait_done)
                              // Go into the first error state
                                   n_s = ERROR_1 ;

 // If enter is not pressed and random wait is not finished
```

```
                              else if (~enter && ~rwait_done)
                             // we remain in the R_wait state
                                  n_s = R_WAIT ;

// If the random counter is finished, enter into the white state

                              else if (rwait_done)
                              begin
                                    time_en = 1 ;
                                     n_s = WHITE;
                              end
                        end

                  ERROR_1 :
                        begin
                             // Turn on the RED light
                                  color_r = 3'b001;
                                  color_g = 3'b000;
                                  color_b = 3'b000;

                             // Start the delay counter
                                  start_wait5 = 1 ;

                             if (wait5_done)
                                  n_s = IDLE;
// If the 5 second timer is not done we want to remain in idle
                            else
                                  n_s = ERROR_1;

                        end

                  WHITE:
                        begin
                             time_en = 1 ;
                             time_clr =0;

                                  color_r = 3'b001;
                                  color_g = 3'b001;
                                  color_b = 3'b001;

                             if (enter && ~time_late)
                                  begin
                                       time_en= 0;
   // Stop  the timer
                                       n_s = DISPLAY ;
```

```
                                        end

                                    else if(~enter && ~time_late)
                                        n_s = WHITE;

                                    else if (time_late)
                                    n_s = ERROR_2;
                            end


                        DISPLAY:
                            begin
    //Green LED remains on
                                        color_r = 3'b000;
                                        color_g = 3'b000;
                                        color_b = 3'b000;

  // Tells the seven seg to display the digits
                                     rs_en = 1 ;
                                        if (start)
                                            begin
                                               start_wait5 = 1 ;
                                               next = R_WAIT;
                                            end
                                        else
     // If start button is not pressed
                                            n_s = DISPLAY ;
                            end

                        ERROR_2:
                            begin

                                    start_wait5 = 1 ;

                                    //Yellow light is on
                                        color_r = 3'b001;
                                        color_g = 3'b001;
                                        color_b = 3'b000;
// Once the five second time period is up return to the idle state

                                        if(~wait5_done)
                                            n_s = ERROR_2;
                                        else if (wait5_done)
                                            n_s = IDLE ;
                                    end
                            endcase
                        end
                    endmodule // reaction fsm
```

3.3.11 Random Wait

Random Wait is the module that generates a random wait time before the start signal will be given to the user. This module has an input of clk, start_wait, and outputs rwait_done which is passed to the reaction_fsm.It uses 3 bit counter that is passed into the flip flop when the start button is pressed and adds one to that random number generated using the random_num module.

Inputs:
- `start_rwait`: Signal sent by the reaction FSM to start the random wait
- `clk`: Clock signal
- `rst`: Reset signal

Outputs:
- `rwait_done`: output sent back to the reaction FSM. It allows the FSM to record the reaction time.

```
module random_wait( input logic clk, rst, start_rwait,
                    output logic rwait_done );

  //instantiations
       logic [2:0] random_num;
       logic [2:0] add;
       logic [13:0] wait;
       logic [13:0] d_count;
       logic [3:0] res1;

      count_3bit U_INST1 (.clk(clk), .rst(rst), .q(random_num));

        random_num U_INST2 (.clk(clk) , .en(start_rwait),
        .d(random_num), .q(add));

//Adding a delay counter
       delay_counter_1 U_INST3 (.clk(clk), .rst(start_rwait),
       .delay(d_count));
```

```
     // We compute the addition

        assign res1 = add + 1 ;

    //We shift the time for wait for 10 to the left
        assign wait =  res1 << 10 ;

 // We check if the values of the wait equals the output of the delay
counter to assert wait_done true

         assign rwait_done = (wait == d_count) ;


endmodule //random wait
```

3.3.12 Count 3 Bit

Inputs:

- `clk`: Clock signal
- `rst`: Reset signal

Outputs:

- `[2:0] q`: The value generated as an output of the counter

It is a simple counter module that counts 3-bit inputs at the positive clock edge of the circuit. It has an input of clock, reset and output q that is used to count 3 bits.

```
module count_3bit ( input logic clk, rst,
                         output logic [2:0] q );
   always_ff @(posedge clk)
              if (rst) q<= 3'd0;
         else q<= q + 3'd1;
endmodule // count_3bit
```

3.3.11 Delay Counter

Inputs:

- `clk`: Clock signal
- `rst`: Reset signal

Outputs:

- `[13:0] delay`: The value generated as a delay

Delay Counter for the additional wait is used in the random wait to add an additional delay in the random wait module. It has an input of clk, rst, and output q_delay for the reaction timer. In this module when rst is pressed q_delay is set to 0. If rst is not pressed delay is added for the reaction timer.

```
          module delay_counter_1( input logic clk, rst,
                                 output logic [13:0] delay);

        always_ff @(posedge clk)
            if (rst) delay<= 0;

           else delay<= delay + 1;

          endmodule // delay counter
```

3.3.12 Random Number

Inputs:

- `input logic [2:0] d`: We use the output of the counter as an input
- `clk`: Clock signal
- `en`: Enable signal

Outputs:

- `logic [2:0] q`: Output which will be sent back to the random wait module

The random number module helps us to generate the random number for our random_wait in the reaction fsm. It has an input of clk, en, and d with an output of q. At the positive edge of the clk the value of d is assigned to q which will help us to generate a random number.

```
module random_num( input logic clk ,en, input logic [2:0] d,
                 output logic [2:0] q);

               always_ff @(posedge clk)
// If en is asserted q gets d
               if (en) q <= d ;
          endmodule // random_num
```

3.3.13 Delay Counter - Delay Wait

Inputs :

- `clk`: Clock signal
- `rst`: Reset signal
- `start_wait5:` signal to initiate the five-second counter

Outputs :

- `wait5_done` : delay counter, input signal that says the five-second counter is complete.

This module is used to reset the counter every 5 seconds as we need three samples at 5-second intervals. This module has an input of clk, en, and an output q which is passed to d when en is true. The signal for the delay counter will rise every 5 seconds and will be used as the reset for the counter and the clock edge for the registers so they can take the data every 5 seconds from the counter.

```
module delay_counter( input logic clk, rst, start_wait5,
                      output logic wait5_done);

// Logic Instantiation
logic [12:0] q ;

always_ff @ (posedge clk)
begin

    if(rst)
        q <= 0 ;

     else if  (start_wait5 && q == 13'd5000)

          begin
// set wait5_done equal to one
          wait5_done <= 1 ;
          q <= 0 ;
          end
     else if (start_wait5 && q != 13'd5000)
        begin
            q <= q+1 ;
            wait5_done <= 0 ;
        end
  end
endmodule
```

3.3.14 RGB

Inputs:

- `logic [2:0] color_r , color_g, color_b`: Inputs the intensities of the red, green and blue LEDs

- `clk`: Clock signal
- `rst`: Reset signal

Outputs:

- `rgb_r, rgb_g, rgb_b`: Turn on the LEDs

This module uses the input about the color RGB into the actual color configuration color in the LED. It is used to indicate various error states and has a color_r, color_g, color_b, and output led_r, led_g, led_b to represent the different states of the reaction timer.

```
module rgb_pwm (input logic clk, rst,
                input logic [2:0] color_r , color_g, color_b,
                output logic rgb_r, rgb_g, rgb_b);
// Logic Instantiations
                logic [3:0] q;

always_ff @(posedge clk)
            if(rst) q <= 0;
            else q <= q + 1 ;

always_comb
 begin

        if (q < color_r) rgb_r <= 1;
        else rgb_r <= 0 ;
        if (q < color_g) rgb_g <= 1;
        else rgb_g <= 0 ;
        if (q < color_b) rgb_b <= 1;
        else rgb_b <= 0 ;
 end
Endmodule
```

3.3.15 Time Count

Inputs :

- `clk`: Clock signal
- `rst`: Reset signal
- `time_clr:` Input signal that acts as a rst for the counter.

Outputs :

- `time_late:` output signal sent from the time count module that tells the user the 10-second time in which they have to press the enter button.

- `[3:0] d0,d1,d2,d3` : output which displays the value of the counter in a seven segment display.

The use of time_count module is used to count the time while waiting for the user's input.This module has an input of clk,time_clr,time_en,rst as an input and an output of time_late,d0,d1,d2,d3 to count the time.This module has four register that shifts the carry out every time the register reaches more than 9 incrementation. The output obtained from the time count is used to display the output in the seven-segment display.

```
module time_count( input logic clk, time_clr, time_en,rst,
                          output logic time_late,
                          output logic [3:0] d0,d1,d2,d3);


// Logic Instantiations
logic c0,c1,c2,c3 ;
logic [3:0] d;
logic q ;

register U_INST1(.clk(clk),.rst(time_clr),
.enb(time_en),.q(d0), .carry(c0)) ;
register U_INST2(.clk(clk), .rst(time_clr),
.enb(c0) ,.q(d1), .carry(c1)) ;
register U_INST3(.clk(clk), .rst(time_clr),
.enb(c1),.q(d2), .carry(c2)) ;
register U_INST4(.clk(clk), .rst(time_clr),
.enb(c2),.q(d3), .carry(c3)) ;

assign time_late = (c3==1) ;

ten_count U_INST5(.clk(clk) , .enb(time_en),.rst(rst),
.time_late(time_late));

endmodule  // time_count
```

3.3.16 Register

Inputs:

- `enb`: Enable signal
- `clk`: Clock signal
- `rst`: Reset signal

Outputs:

- `[3:0] q`: The outputs of the registers
- `Carry:` The carry from the registers

This register module is used to count the time of the reaction timer. This module is instantiated in the time count module to perform the count and display the time elapsed in the seven-segment decoder.

```
module register( input logic clk, rst, enb,
               output logic [3:0] q,
               output logic carry);

  assign carry =(q==20) && enb;

always_ff @(posedge clk )
    begin
        if (rst || carry) q <= 0;
        else if (enb) q <= q + 1;
    end
endmodule // register
```

3.3.17 Ten Count

Inputs:

- `enb`: Enable signal
- `clk`: Clock signal
- `rst`: Reset signal

Outputs:

- `time_late`: Goes back to the reaction fsm to count if 10 seconds have passed (and control states accordingly)

The ten count module is used to count the time and return the time_late.This module runs on the same clock signal input signal that initiates the 10-sec timer.This module has an input of clk,enb and rst with an output of time_late to count the time.

```
module ten_count( input clk, enb,rst,
                             output logic time_late);

     // Logic Instantiation
       logic [13:0] q ;
```

```
          always_ff @ (posedge clk)
               begin

                if (~enb || rst) q <= 0;  // Serves reset for the
 counter
                else if (enb && q != 14'd9999)
                  begin
                      q<= q + 1 ;

                     time_late <= 0;
                  end
                else if (enb && q == 14'd9999)
                    begin
                      time_late <= 1 ;
                      q <= 0 ;
                    end
                 end
           endmodule
```

3.3.18 Clock Divider:

Inputs:

- `clk`: Clock signal
- `rst`: Reset signal

Outputs:

- `sclk:` synchronized signal on a particular clock. The clock divider divides the clock into 1000 Hz.

```
module clkdiv(input logic clk, input logic rst, output logic
   sclk);
     parameter DIVFREQ = 100;  // desired frequency in Hz (change as
   needed)
     parameter DIVBITS = 26;   // enough bits to divide 100MHz down
to 1
     Hz
     parameter CLKFREQ = 100_000_000;
     parameter DIVAMT = (CLKFREQ / DIVFREQ) / 2;

      logic [DIVBITS-1:0] q;

      always_ff @(posedge clk)
      if (rst) begin      // Check reset and set q and sclk to 0
    q <= 0;
    sclk <= 0;
      end
      else if (q == DIVAMT-1) begin
    q <= 0;
    sclk <= ~sclk;
     end
      else q <= q + 1;
endmodule // clkdiv
```

4. System Validation and Performance:

Reaction Timer:

Test	Action	Result	Pass/Fail	Initial
1	SW0 is off	All seven-segment are off	**PASS**	
2	Pressed start button	Random waiting time before the LED is turned on	**PASS**	A.B & L.P
3	Pressed Start button and pressed the Enter Button before the GO LED is turned on	Red LED is turned on for 5 seconds and then we return back to the IDLE Stage	**PASS**	A.B & L.P
4	Pressed Start after the LED has been turned on for 10s	Yellow LED is turned on for 10 seconds and then we return back to the IDLE Stage	**PASS**	A.B & L.P
5	Pressed the Enter Push Button after the white light has been turned on	Device is in the IDLE Stage. The display is turned off.	**PASS**	A.B & L.P
6	Pressed the Reset Button after the Display state	IDLE Mode; The seven-segment display is off.	**PASS**	A.B & L.P
7	Pressed Enter button before Start button	Seven-segment display is turned off; LEDs are off	**PASS**	A.B & L.P
8	Pressed Enter after the reaction time is displayed	Reaction Time remained on the seven-segment display	**PASS**	A.B & L.P
9	Random_Wait Times Verify	FirstTrail : 8.038 sec Second Trail : 5.072 sec	**PASS**	A.B & L.P

		Trail 3: 7.190 sec		

Pulse Monitor

Test	Action	Result	Remarks	Initials
1	SW0 is turned on	Display XXXXX000 on a seven segment display. *The X's represent segments are off	PASS	A.B & L.P
2	Placed finger on the sensor to measure pulse	Pulse Monitor 1: 84 Pulse Monitor 2: 90 Pulse Monitor 3: 96	PASS	A.B & L.P
3	Pressed reset button	Display is reset to displaying	PASS	A.B & L.P
4	No finger is placed on the sensor	Display shows XXXXX000	PASS	A.B & L.P

5. Appendix A:

This section consists of the specification and functionality of the health monitor system.

Inputs

- Mode select switch (slide switch SW0)
- Reaction time START button start (push button BUTNC)
- Reaction time ENTER button (push button BUTNL)
- System RESET button (pushbutton BTNL)
- Pulse Sensor (PMOD JA connector input pin 1)

Outputs

- 8-digit seven-segment display (anode_l, segs_l)
- Reaction Timer “Go” Lamp (RGB LED LD17)

Operation

- The health monitor provides two different functions: (a) when the mode select switch SW0 is on, it measures the user’s pulse, and (b) When the mode select switch SW0 is off, it tests the user’s reaction time.

Pulse Monitor

- Receives a pulse signal from an analog pulse sensor on an attached daughterboard plugged into the PMOD connector.
- Counts the number of heartbeats over five second intervals while maintaining the last three samples to calculate the user's pulse as a moving average.
- Displays the user's pulse in beats per minute (BPM) up to a maximum of 255 BPM.
- Unused digits on the 7-segment display should be blank.

Reaction Timer

- When the START button is pressed, the seven-segment display should be turned off (if it isn't already). The circuit should then wait for a random amount of time between roughly 1 and 9 seconds. The wait time should be randomly selected from at least eight different delay values in this range.
- After the random wait, turn on the GO LED and record the amount of time which passes before the user presses the ENTER button. The LED should be off except when waiting for the user to press ENTER.
- Depending on when (and if) the user presses the ENTER button, the seven- segment display and LED will display the result of the reaction time test, as follows:

    - If the ENTER button is pressed up to 9.999 seconds *after* the GO LED turns on, the seven-segment display should be turned on and display the reaction time in the format x.xxx (in seconds). The circuit will continue to display this time until the START button is pressed again.
    - If the ENTER button is pressed *before* the GO LED turns on, the seven-segment display should remain off and the LED color should change to *red* for five seconds to indicate an error. It should remain lit for five seconds after which it should be turned off and the system should return to waiting for the START button to be pressed.
    - If the ENTER button has *not* been pressed 10 seconds after the GO LED turns on, the seven-segment display should remain off and the LED color should change to *yellow* for five seconds to indicate an error. It should remain lit for five seconds after which it should be turned off and the system should return to waiting for the START button to be pressed.

Additional requirements and constraints

- The circuit must be implemented as a fully synchronous circuit using a 1 kHz clock generated by a clock divider.
- All sequential logic (except the clock divider and single pulser circuits) should include a synchronous reset and be connected to a single master RESET input.
- All storage in the circuit must be implemented using flip-flops - the circuit must contain no latches. To check whether your circuit contains latches, use the Vivado Synthesis Report (or watch for warnings about latch inferences in the “messages” pane).
- The RGB LED should display outputs at a comfortable intensity and all colors should be displayed at approximately equal intensity.
- Unused digits in the 7-segment display should be blank in both modes of operation.

6. Appendix B :

This section consists of a set of rules which describe the process of testing the FPGA board.

Reaction Timer

- To verify that SW0 changed the two modes, during the first test we set the SW0 to a logic low (turning the switch off). This caused the seven segments to turn off and remain off until the reaction time was to be displayed.
- To test the “ERROR_1” state of the reaction timer, the enter button was pressed before the LED turned white, which then caused the red light to turn on for five seconds.
- To test the “ERROR_2” state of the reaction timer, the enter button was pressed in the ten-second interval after the white LED was turned on. A yellow LED is thus displayed for five seconds
- We tested the reaction time, where we turned on the enter button within the ten-second interval following the “Go”LED turning on.
- Pressed the reset button after the Display State where the seven segment display is turned off
- Pressed the Enter button while the health monitor was in the IDLE state, before the Start State has been activated
- Pressed Enter button after the reaction time is displayed; The reaction timer has been displayed on the display.
- Verified the random wait times after the start button is pressed.

Pulse Monitor

- Verified that when the SW0 is turned on, the functionality of the board has been switched from reaction timer mode to a pulse monitor mode.

- Tested the functionality and accuracy of the pulse counter by conducting three trials of pulse readings.
- Pressed Reset Button; No finger is placed on the sensor the value displayed on the sensor is XXXXX000.

7. Appendix C:

In order for our code to be implemented on an FPGA Board, it was essential to use the constraints file to connect the code functions to relevant ports on the motherboard.

```
# Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk100Mhz }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports {clk100Mhz}];


#Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { mode }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]

# LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { led_r }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { led_g }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { led_b }]; #IO_L17N_T2_A25_15 Sch=led[2]
#set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]


##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 }
[get_ports { segs_l[6] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 }
[get_ports { segs_l[5] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 }
[get_ports { segs_l[4] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 }
[get_ports { segs_l[3] }]; #IO_L17P_T2_A26_15 Sch=cd
```

```
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 }
[get_ports { segs_l[2] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 }
[get_ports { segs_l[1] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 }
[get_ports { segs_l[0] }]; #IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 }
[get_ports { dp_l }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 }
[get_ports { an_l[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 }
[get_ports { an_l[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 }
[get_ports { an_l[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 }
[get_ports { an_l[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 }
[get_ports { an_l[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 }
[get_ports { an_l[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 }
[get_ports { an_l[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 }
[get_ports { an_l[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]

#Buttons
set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 }
[get_ports { rst }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { start }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 }
[get_ports { enter }]; #IO_L4N_T0_D05_14 Sch=btnu
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 }
[get_ports { BTNL }]; #IO_L12P_T1_MRCC_14 Sch=btnl
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 }
[get_ports { BTNR }]; #IO_L10N_T1_D15_14 Sch=btnr
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 }
[get_ports { BTND }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd


#Pmod Headers
#Pmod Header JA
set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 }
[get_ports { pulse_in }]; #IO_L20N_T3_A19_15 Sch=ja[1]
```

```
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 }
[get_ports { JA[2] }]; #IO_L21N_T3_DQS_A18_15 Sch=ja[2]
#set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 }
[get_ports { JA[3] }]; #IO_L21P_T3_DQS_15 Sch=ja[3]
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 }
[get_ports { JA[4] }]; #IO_L18N_T2_A23_15 Sch=ja[4]
#set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 }
[get_ports { JA[7] }]; #IO_L16N_T2_A27_15 Sch=ja[7]
#set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 }
[get_ports { JA[8] }]; #IO_L16P_T2_A28_15 Sch=ja[8]
#set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 }
[get_ports { JA[9] }]; #IO_L22N_T3_A16_15 Sch=ja[9]
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 }
[get_ports { JA[10] }]; #IO_L22P_T3_A17_15 Sch=ja[10]
```

8. Summary

This report describes the process of implementation and testing of the health monitor with the help of FPGA board. It consists of two major components , the pulse monitor and the reaction timer. These two different modules were designed separately to implement the functionality of health monitor system. Combining these modules we were able to design a top module for our project able to operate in both modes. We switched between both modes using the mode.