ELC 5396 02 Class Report 2

Elizabeth Kooiman

September 14, 2023

Summary

The goal of this project was to design and implement a reaction timer on the Nexys4 DDR board. The user controlled inputs were specified to be a clear button, a reset button, and a stop button. In this design, the left button (BTNL) was used as clear, the upper button was used as start (BTNU), and the right button (BTNR) was used as stop. The goal of this module was to control the action of the reaction timer and the output to the seven segment (sseg) display depending on the current state of the implementation. A straightforward way to implement this design was to use a state machine, which was the approach taken in this lab. The overall design layout is shown in Figure 1. The top level module is the reaction time module. The code for this is shown below in Listing 7. The top module contains the state machine and uses the sseg decoder module to determine what to output to the sseg display depending on the current state. The other modules instantiated within the top module are the count ms module in Listing 4, the counter module in Listing 3, and the randnum module in Listing 5. The count ms module slows the clock speed to increment once every millisecond. It does this by counting to 99,999 and then incrementing once it does so. The reason this number was selected can be seen below:

$$1ms \times \frac{1000000ns}{1ms} \times \frac{1clockcycle}{10ns} = 1000000clockcycles$$

The count module is used to increment a counter until it reaches a determined number that indicated the desired number of seconds ranging between 2 and 15 has passed. The randnum module shown in Listing 5 contains a linear feedback shift register setup that is used to generate a pseudo-random number. The output of this module is a N bit random number output. that output is then taken and the modulus operation is performed and then an offset is used to generate a pseudo-random number between 2 and 15 for this specific application.

The ASMD derived for this problem is shown in Figure 2. It shows the various states of the state machine as well as the datapath to transfer between them. This design was successfully implemented and ran on the board correctly. A video of the operation as well as the code and constraint file can be found in the github repo at https://github.com/elizabethkooiman/Kooiman_SoC.

Results

Figure 3 shows the simulation results of the bcdconverter module shown in Listing 2. The testbench is shown in Listing 10. The module operated as expected and successfully converted 11 bits of binary into binary coded decimal.

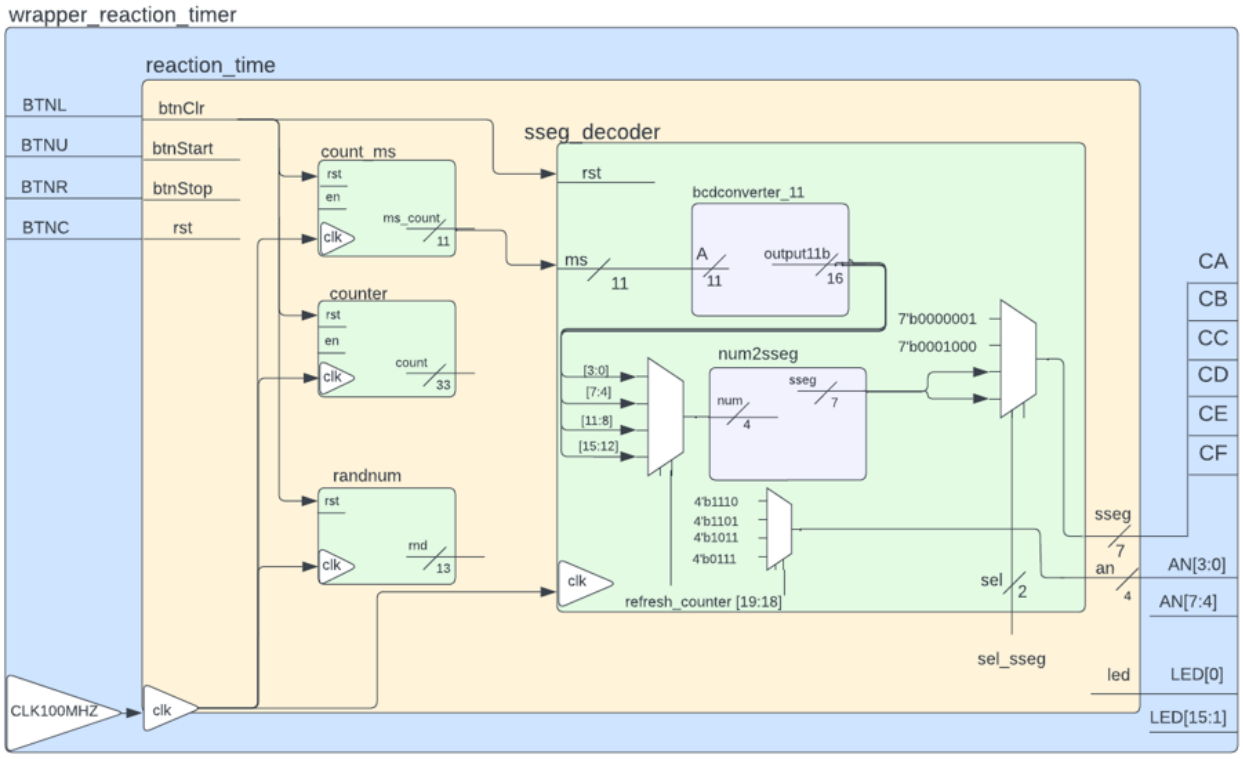

Figure 1: Overall Design Diagram

Figure 4 shows the simulation results of the count module shown in Listing 3. The testbench is shown in Listing 11. The module operated as expected and incremented the count once every clock cycle when the enable was high.

Figure 5 shows the simulation results of the num2sseg module shown in Listing 6. The testbench is shown in Listing 12. As seen in the figure, the num2sseg module successfully muxes the input number to output the corresponding 7 bit sseg value.

Figure 6 shows the simulation results of the sseg decoder module shown in Listing 8. The testbench is shown in Listing 13. The sseg decoder module successfully muxes between the 4 different input options depending on the specified state.

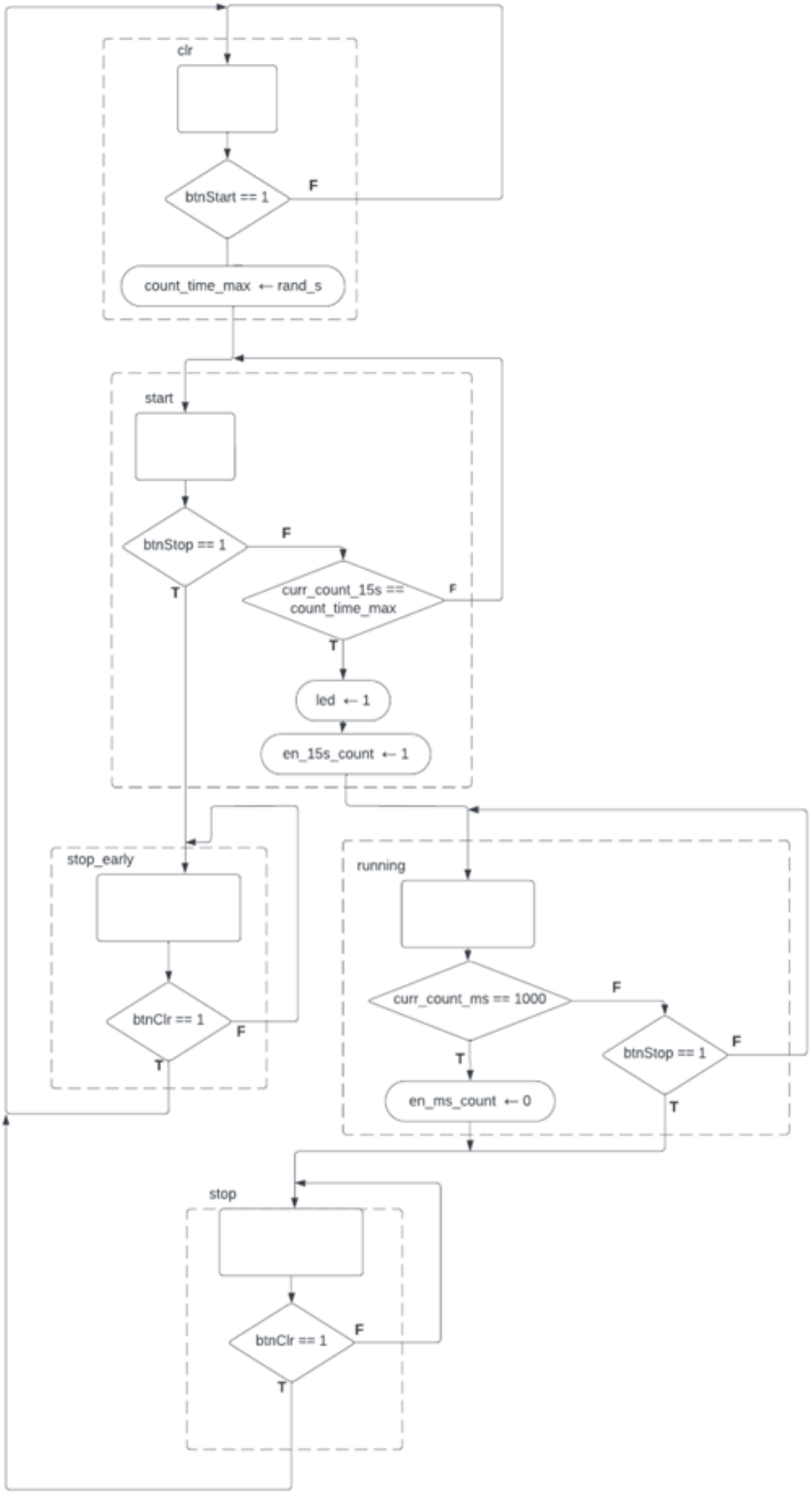

Figure 2: Overall Design Diagram

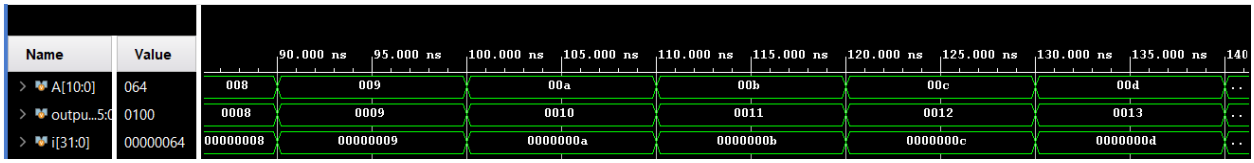

Figure 3: bcdconverter Test Results

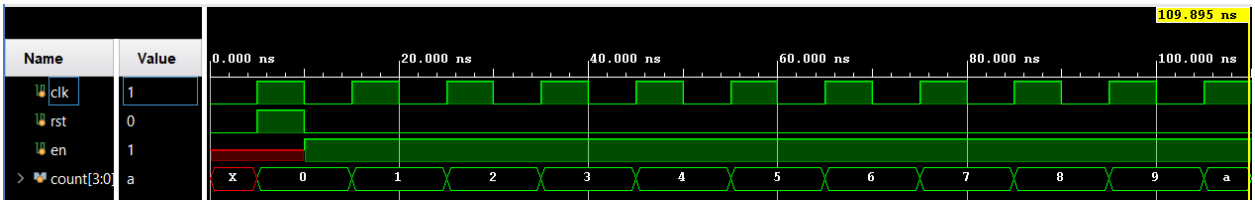

Figure 4: Counter Test Results

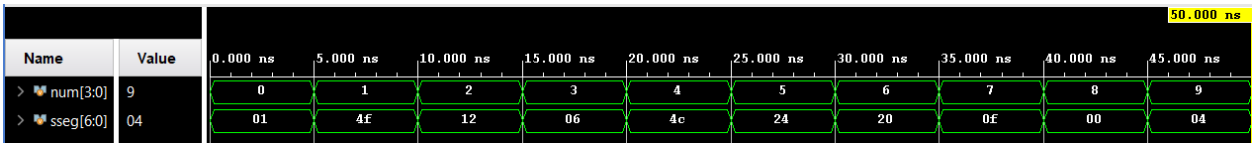

Figure 5: num2sseg Test Results

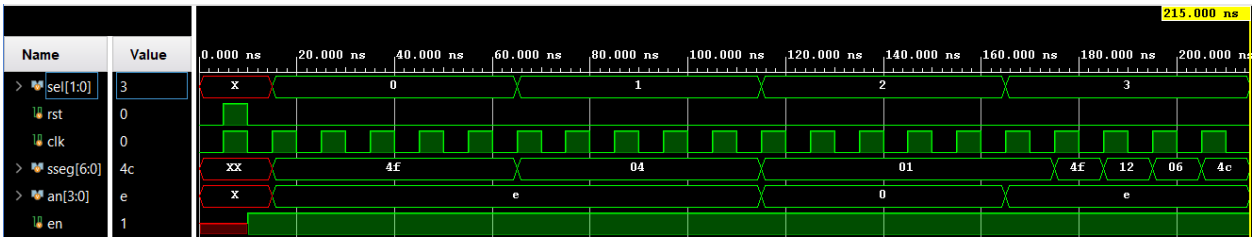

Figure 6: sseg decoder Test Results

Code

Listing 1: add3 module

```
`timescale 1ns / 1ps
//////////////////////////////////////
// Engineer: Elizabeth Kooiman
//
// Create Date: 09/07/2023 08:02:58 PM
// Module Name: add3
// Project Name: Class Report 2
// Description:
//
//////////////////////////////////


module add3(
    input logic [3:0] num,
    output logic[3:0] modnum
    );

    always_comb
        if(num>=5)
            modnum = num+3;
        else
            modnum = num;
endmodule
```

Listing 2: bcdconverter 11 bit

```
`timescale 1ns / 1ps
//////////////////////////////////////
// Engineer: Elizabeth Kooiman
// Create Date: 09/07/2023 08:01:03 PM
// Module Name: bcdconverter_11
// Project Name: Class Report 2
// Description:
//////////////////////////////////////

module bcdconverter_11(
    input logic [10:0] A,
    output logic [15:0] output11b
    );
    wire tens;
    wire ones;
    wire c4msb;
    wire c5msb;
    wire c6msb;
    wire c7msb;
    wire c9msb;
    wire c10msb;
    wire c11msb;
    wire c12msb;

    wire [2:0] temp1;
```

```
    wire [2:0] temp2;
    wire [2:0] temp3;
    wire [2:0] temp4;
    wire [2:0] temp5;
    wire [2:0] temp6;
    wire [2:0] temp7;
   // wire [2:0] temp8;
    wire [2:0] temp9;
    wire [2:0] temp10;
    wire [2:0] temp11;
    wire [2:0] temp12;
    wire [2:0] temp14;

    wire [3:0] ones11;
    wire [3:0] tens11;
    wire [3:0] hundreds11;
    wire [3:0] thousands11;

    add3 c1(
    .num({1'b0,A[10:8]}),
    .modnum({c1msb,temp1})
    );
    add3 c2(
    .num({temp1,A[7]}),
    .modnum({c2msb,temp2})
    );
    add3 c3(
    .num({temp2,A[6]}),
    .modnum({c3msb,temp3})
    );
    add3 c4(
    .num({temp3,A[5]}),
    .modnum({c4msb,temp4})
    );
     add3 c5(
    .num({temp4,A[4]}),
    .modnum({c5msb,temp5})
    );
     add3 c6(
    .num({temp5,A[3]}),
    .modnum({c6msb,temp6})
    );
     add3 c7(
    .num({temp6,A[2]}),
    .modnum({c7msb,temp7})
    );
     add3 c8(
    .num({temp7,A[1]}),
    .modnum({tens11[0], ones11[3:1]})
    );
     add3 c9(
    .num({1'b0,c1msb,c2msb,c3msb}),
    .modnum({c9msb,temp9})
    );
```

```
     add3 c10(
    .num({temp9,c4msb}),
    .modnum({c10msb,temp10})
    );
     add3 c11(
    .num({temp10,c5msb}),
    .modnum({c11msb,temp11})
    );
     add3 c12(
    .num({temp11,c6msb}),
    .modnum({c12msb,temp12})
    );
     add3 c13(
    .num({temp12,c7msb}),
    .modnum({hundreds11[0],tens11[3:1]})
    );
     add3 c14(
    .num({1'b0,c9msb,c10msb,c11msb}),
    .modnum({thousands11[1],temp14})
    );
     add3 c15(
    .num({temp14,c12msb}),
    .modnum({thousands11[0],hundreds11[3:1]})
    );

    assign ones11[0] = A[0];
    assign thousands11[3] = 1'b0;
    assign thousands11[2] = 1'b0;

    assign output11b = {thousands11,hundreds11,tens11,ones11};

endmodule
```

Listing 3: count module

```
`timescale 1ns / 1ps
//
   //////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer:
//
// Create Date: 09/08/2023 11:41:08 AM
// Design Name:
// Module Name: counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//
//
    //////////////////////////////////////////////////////////////////////////////////



module counter#(parameter N = 20)(
    input logic clk,
    input logic rst,
    input logic en,
    output logic [N-1:0] count
    );

    parameter ZERO={N{1'b0}};
    logic [N-1:0] count_curr;
    logic [N-1:0] count_next;

    always_ff @(posedge(clk),posedge(rst))
    begin
        if (rst) begin
        count_curr <= ZERO;
        end
        else begin
        count_curr <= count_next;
        end
    end

    always_comb
        if(en) begin
            count_next = count_curr +1;

        end
        else begin
            count_next = count_curr;
        end

    assign count = count_curr;

endmodule
```

Listing 4: count ms module

```
`timescale 1ns / 1ps
////////////////////////////////////
// Engineer: Elizabeth Kooiman
// Create Date: 09/13/2023 09:55:52 AM
// Module Name: counter_ms
// Project Name: Class Report 2
// Description:
////////////////////////////////////

module counter_ms(
    input logic clk,
```

```
    input logic rst,
    input logic en,
    output logic [10:0] ms_count
    );

    logic [31:0] count, curr_ms;
    always_ff @(posedge(clk),posedge(rst))
    if(rst) begin
        curr_ms = {32{1'b0}};
        count = {32{1'b0}};
        end
    else begin
        if(count == 99999)
        begin
            count <=0;
            curr_ms = curr_ms +1;
        end
        else if(en)
            count = count +1;
        end

    assign ms_count = curr_ms[10:0];

endmodule
```

Listing 5: LFSR random number module

```
`timescale 1ns / 1ps
//////////////////////////////////////
// Engineer: Elizabeth Kooiman
// Create Date: 09/13/2023 09:16:28 PM
// Module Name: lfsr_randnum
// Project Name: Class Report 2
// Description:
//////////////////////////////


module lfsr_randnum#(parameter N = 13)(
        input logic clk,
        input logic rst,
        output logic [N-1:0] rnd
    );

always_ff@(posedge(clk),posedge(rst))
begin
if(rst)
    rnd = {N{1'b1}};
else
    rnd = {rnd[N-2:0],(rnd[N-1]^rnd[N-2])};
end
endmodule
```

Listing 6: num2sseg module

```
`timescale 1ns / 1ps
//////////////////////////////////////
// Engineer: Elizabeth Kooiman
// Create Date: 09/08/2023 10:07:38 AM
// Module Name: num2sseg
// Project Name: Class Report 2
// Description:
//////////////////////////////////////


module num2sseg(
    input logic [3:0] num,
    output logic [6:0] sseg
    );

    always_comb
    case(num)
        4'b0000: sseg = 7'b0000001;
        4'b0001: sseg = 7'b1001111;
        4'b0010: sseg = 7'b0010010;
        4'b0011: sseg = 7'b0000110;
        4'b0100: sseg = 7'b1001100;
        4'b0101:  sseg = 7'b0100100;
        4'b0110: sseg = 7'b0100000;
        4'b0111: sseg = 7'b0001111;
        4'b1000: sseg = 7'b0000000;
        4'b1001: sseg = 7'b0000100;
        default: sseg = 7'b0000001;

    endcase
endmodule
```

Listing 7: reaction timer top module

```
`timescale 1ns / 1ps
//////////////////////////////////////
// Engineer: Elizabeth Kooiman
// Create Date: 09/06/2023 08:37:52 PM
// Design Name:
// Module Name: reaction_time
// Project Name: Class Report 2
// Description:
//////////////////////////////////////


module reaction_time(
    input logic btnClr,
    input logic btnStart,
    input logic btnStop,
    input logic clk,
    input logic rst,
    output logic [6:0] sseg,
    output logic [3:0] an,
    output logic led
```

```
    );

    //define local parameters for state machine
    localparam [2:0] clear = 3'b000;
    localparam [2:0] start = 3'b001;
    localparam [2:0] running = 3'b010;
    localparam [2:0] stop = 3'b011;
    localparam[2:0] stop_early = 3'b100;

    //Parameters for determining count
    localparam N_15s = 32;

    //define logic variables
    logic [N_15s-1:0] curr_count_15s;
    logic [10:0] curr_count_ms;
    logic [2:0] curr_state, next_state;
    logic [1:0] sel_sseg;
    logic [12:0] rand_initial;
    logic [12:0] rand_s;
    logic [N_15s-1:0] count_time_max;

    logic en_ms_count;
    logic en_15s_count;
    logic rst_ms;
    logic rst_15s;

    counter_ms countms(
        .clk(clk),
        .rst(rst_ms),
        .en(en_ms_count),
        .ms_count(curr_count_ms)
        );

    counter#(.N(N_15s)) count15s(
        .clk(clk),
        .rst(rst_15s),
        .en(en_15s_count),
        .count(curr_count_15s)
        );

    lfsr_randnum#(.N(13)) rand_num(
        .clk(clk),
        .rst(btnClr),
        .rnd(rand_initial)
        );

    assign rand_s = (rand_initial % 13) + 2;

    always_ff @(posedge(clk),posedge(btnClr)) begin
        if(btnClr)begin
            curr_state <= clear;
            rst_ms  <= 1'b1;
            rst_15s <= 1'b1;
        end
```

```
        else begin
            curr_state <= next_state;
            rst_ms  <= 1'b0;
            rst_15s <= 1'b0;
        end

    end

    sseg_decoder decode0(
        .sel(sel_sseg),
        .ms(curr_count_ms),
        .rst(btnClr),
        .clk(clk),
        .sseg(sseg),
        .an(an)
        );

    always_comb begin
        next_state = curr_state;
        case(curr_state)
            clear: begin
                sel_sseg = 2'b00;
                en_ms_count = 1'b0;
                en_15s_count = 1'b0;
                led = 1'b0;
                if(btnStart) begin
                    next_state = start;
                    count_time_max = 27'd100000000 * rand_s;
                end
                else
                    next_state = clear;
                end

            start: begin
                sel_sseg = 2'b10;
                led = 1'b0;
                en_15s_count = 1'b1;
                if(btnStop) begin
                    next_state = stop_early;
                    end
                else if(curr_count_15s == count_time_max) begin
                    led = 1'b1;
                    en_15s_count = 1'b0;
                    next_state = running;
                end
                else begin
                    next_state = start;
                end
                end

            running: begin
                en_ms_count = 1'b1;
                sel_sseg = 2'b11;
                led = 1'b1;
```

```
                if(curr_count_ms == 32'd1000) begin
                    next_state = stop;
                    en_ms_count = 1'b0;
                    end
                else if (btnStop) begin
                    next_state = stop;
                end
                else
                    next_state = running;
                end

            stop: begin
                sel_sseg = 2'b11;
                led = 1'b0;
                en_ms_count = 1'b0;
                if(btnClr) begin
                    next_state = clear;
                end
                else begin
                    next_state = stop;
                end
                end
            stop_early:
                begin
                    sel_sseg = 2'b01;
                    led = 1'b0;
                    en_ms_count = 1'b0;
                    en_15s_count = 1'b0;
                if(btnClr) begin
                    next_state = clear;
                    end
                else begin
                    next_state = stop_early;
                    end
                end
        endcase
        end
endmodule
```

Listing 8: sseg decoder module

```
`timescale 1ns / 1ps
//////////////////////////////////////
// Engineer: Elizabeth Kooiman
// Create Date: 09/07/2023 12:38:20 PM
// Module Name: sseg_decoder
// Project Name: Class Report 2
// Description:
//////////////////////////////////////


module sseg_decoder(
    input logic [1:0] sel,
    input logic [10:0] ms,
```

```
input logic rst,
input logic clk,
output logic [6:0] sseg,
output logic[3:0] an
);

localparam clear = 2'b00;
localparam stop_early = 2'b01;
localparam start = 2'b10;
localparam count = 2'b11;
localparam refresh_N = 20;

//declare variables
logic [15:0] bcd_ms;
logic [3:0] num2sseg_in;
logic [6:0] num2sseg_out;

//setup refresh to display on the 4 sseg
logic [refresh_N-1:0] refresh_counter;
always_ff @(posedge(clk),posedge(rst)) begin
    if(rst)
        refresh_counter <= 0;
    else
        refresh_counter <= refresh_counter + 1;
end

bcdconverter_11 bin2bcd(
.A(ms),
.output11b(bcd_ms)
);

num2sseg n2s0(
.num(num2sseg_in),
.sseg(num2sseg_out)
);

always_comb begin
case(sel)
    clear:
        if(refresh_counter[refresh_N-1:refresh_N-2] == 2'b00)begin
            an = 4'b1110;
            sseg = 7'b1001111;
            end
        else if(refresh_counter[refresh_N-1:refresh_N-2] == 2'b01)
           begin
            an = 4'b1101;
            sseg = 7'b1001000;
            end
        else
            an = 4'b1111;


    stop_early:
        begin
```

```
            if(refresh_counter[refresh_N-1:refresh_N-2] == 2'b00)begin
                an = 4'b1110;
                sseg = 7'b0000100;
                end
            else if(refresh_counter[refresh_N-1:refresh_N-2] == 2'b01)
               begin
                an = 4'b1101;
                sseg = 7'b0000100;
                end
            else if(refresh_counter[refresh_N-1:refresh_N-2] == 2'b10)
               begin
                an = 4'b1011;
                sseg = 7'b0000100;
                end
            else begin
                an = 4'b0111;
                sseg = 7'b0000100;
                end
            end

        start:
            begin
            an = 4'b0000;
            sseg = 7'b0000001;
            end
        count: begin
            if(refresh_counter[refresh_N-1:refresh_N-2] == 2'b00)begin
                an = 4'b1110;
                num2sseg_in = bcd_ms[3:0];
                end
            else if(refresh_counter[refresh_N-1:refresh_N-2] == 2'b01)
               begin

                an = 4'b1101;
                num2sseg_in = bcd_ms[7:4];
                end
            else if(refresh_counter[refresh_N-1:refresh_N-2] == 2'b10)
               begin
                an = 4'b1011;
                num2sseg_in = bcd_ms[11:8];
                end
            else begin
                an = 4'b0111;
                num2sseg_in = bcd_ms[15:12];
            end
             sseg = num2sseg_out;
            end
         endcase
    end
endmodule
```

Listing 9: wrapper module for reaction timer

```
`timescale 1ns / 1ps
```

```
//////////////////////////////////////
// Engineer: Elizabeth Kooiman
// Create Date: 09/09/2023 12:37:23 PM
// Module Name: wrapper_reaction_timer
// Project Name: Class Report 2
// Description:
//////////////////////////////////////


module wrapper_reaction_timer(
        input CLK100MHZ,
        input BTNC,//rst
        input BTNL, //btnClr
        input BTNU, //btnStart
        input BTNR, //btnStop
        output [15:0] LED,
        output [7:0] AN,
        output CA,
        output CB,
        output CC,
        output CD,
        output CE,
        output CF,
        output CG
        //output DP
    );


    assign LED[15:1] = 0;
    assign AN[7:4] = 4'b1111;

    reaction_time(
    .btnClr(BTNL),
    .btnStart(BTNU),
    .btnStop(BTNR),
    .clk(CLK100MHZ),
    .rst(BTNC),
    .sseg({CA,CB,CC,CD,CE,CF,CG}),
    .an(AN[3:0]),
    .led(LED[0])
    );
endmodule
```

Listing 10: bcdconverter 11 test

```
`timescale 1ns / 1ps
//
   //////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer:
//
// Design Name:
// Module Name: bcdconverter_11_test
```

```
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
   //////////////////////////////////////////////////////////////////////////////////



module bcdconverter_11_t();

    logic [10:0] A;
    logic [15:0] output11b;
    integer i;


bcdconverter_11 uut(
    .A(A),
    .output11b(output11b)

    );

 initial begin

    for (i=11'd0;i<=11'd2047;i=i+1) begin
        A[10:0] = i; #10;
    end




    $finish;
 end

endmodule
```

Listing 11: count module test

```
`timescale 1ns / 1ps
//
   //////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer:
//
// Create Date: 09/08/2023 12:50:25 PM
```

```
// Design Name:
// Module Name: counter_t
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
   //////////////////////////////////////////////////////////////////////////////////



module counter_t();

parameter N = 4;
     logic clk;
     logic rst;
     logic en;
     logic [N-1:0] count;



counter#(.N(N)) uut(
    .clk(clk),
    .rst(rst),
    .en(en),
    .count(count)
    );
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end


    initial begin
    rst = 0;
    #5;
    rst = 1;
    #5;
    rst = 0;
    en = 1;
    #100

    $finish;
    end
endmodule
```

Listing 12: num2sseg module test

```
`timescale 1ns / 1ps
//
   //////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer:
//
// Create Date: 09/08/2023 02:49:39 PM
// Design Name:
// Module Name: num2sseg_t
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
   //////////////////////////////////////////////////////////////////////////////////



module num2sseg_t();

     logic [3:0] num;
     logic [6:0] sseg;


num2sseg uut(
    .num(num),
    .sseg(sseg)
    );

    initial begin
    num = 4'b0000;
    #5;
    num = 4'b0001;
    #5;
    num = 4'b0010;
    #5;
    num = 4'b0011;
    #5;
    num = 4'b0100;
    #5;
    num = 4'b0101;
    #5;
    num = 4'b0110;
    #5;
    num = 4'b0111;
```

```
    #5;
    num = 4'b1000;
    #5;
    num = 4'b1001;
    #5;
    $finish;
    end

endmodule
```

Listing 13: sseg decoder module test

```
`timescale 1ns / 1ps
//
   //////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer:
//
// Create Date: 09/13/2023 09:25:48 AM
// Design Name:
// Module Name: sseg_decoder_t
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
   //////////////////////////////////////////////////////////////////////////////////



module sseg_decoder_t();

     logic [1:0] sel;
     logic [27:0] ms;
     logic rst;
     logic clk;
     logic [6:0] sseg;
     logic[3:0] an;
     logic en;
     parameter N = 4;
     counter#(.N(N)) count1(
    .clk(clk),
    .rst(rst),
    .en(en),
    .count(ms)
    );
```

```
    sseg_decoder uut(
    .sel(sel),
    .ms(ms),
    .rst(rst),
    .clk(clk),
    .sseg(sseg),
    .an(an)
    );


    initial begin
    clk = 0;
    forever #5 clk = ~clk;
    end
    initial begin

        rst = 0;
        #5;
        rst = 1;
        #5;
        rst = 0;
        en = 1;
        ms[27:N] = {23{1'b0}};
        #5;
        sel = 2'b00;
        #50;
        sel = 2'b01;
        #50;
        sel = 2'b10;
        #50;
        sel = 2'b11;
        #50;
        $finish;
    end


endmodule
```