ELEC 3500

Lab 9 – Egg Timer Project

Tawseef Patel 101145333

Individual Report

(Group Member: Saad Babur)

Dec 9, 2022

Background and Design Goals (GA 4.1, 4.7)

When cooking/boiling an egg there are many different levels of cooked that one may want as in the case of steaks. With eggs the consistencies may range from runny to hard and the differences between these two far ends of the spectrum occurs within a matter of minutes depending on the cooking temperature. To ensure the accurate timing for a consistent cook level, a timer can be used. One such timer which was designed in this lab for this exact reason. The egg timer which has a precision of one second allows the user to set a timer to cook an egg for up to an hour. The design specifications for what was required when designing this egg timer using Verilog are mentioned in section 2.

Design Specifications (GA 4.2, 4.7)

To provide a suitable solution for creating the perfect egg, the egg-timer was created and the requirements which were taken into consideration are listed below:

- The timer must indicate if the timer is enabled
- The timer must indicate if the timer is counting down
- The precision of the timer must be in the format mm:ss with a max time of 59:59
- To enhance user experience (innovation) the timer must be able to increment and decrement (0 to 59 if the user wants to press the buttons fewer times)
- The minutes and seconds must be able to increment individually
- The timer must have a reset button which allows for the timer to go back to 0
- The timer must have an enable switch and cooktime button to allow for user to begin inputting the desired time
- The timer must have a start timer button
- The timer must stop if the enable switch is disabled
- The timer must be accessible and be easy to use

These specifications allow for a seamless user experience when looking to use a timer to create the perfect egg.

Design Implementation (GA 4.4, 4.5)

The design was implemented using Vivado which uses Verilog on the NEXYS 4 DDR development FPGA board. The board consists of multiple toggle switches, buttons, LEDs, I/O ports and pins, 2 quad seven-segment displays etc. The schematic of the elaborated design in shown in Figure 1. The design is driven using an internal 100MHz system clock which is divided down to 5MHz using the in-built IP Catalogs Clocking Wizard. This signal is then further divided down into multiple different clock signals for different functionalities. The 5000Hz clock is used on the quad seven-segment displays to ensure a fast enough refresh rate that the human eye cannot detect. While the 1Hz signal is used to act as a 1s timer to decrement the time at the proper frequency. A third clock signal of 100Hz is used for the debouncer which detects changes for the incrementing buttons (minutes/seconds). The mux for the quad seven-segment displays selects between the count time, and cooktime inputs. This helps store the values in the count time or cook time output registers so that the correct time is displayed. The bcdto7seg decoders allow for output binary values from the mux to be displayed on the seven-segment displays.

ELEC3500 Egg Timer

Tawseef Patel

101145333

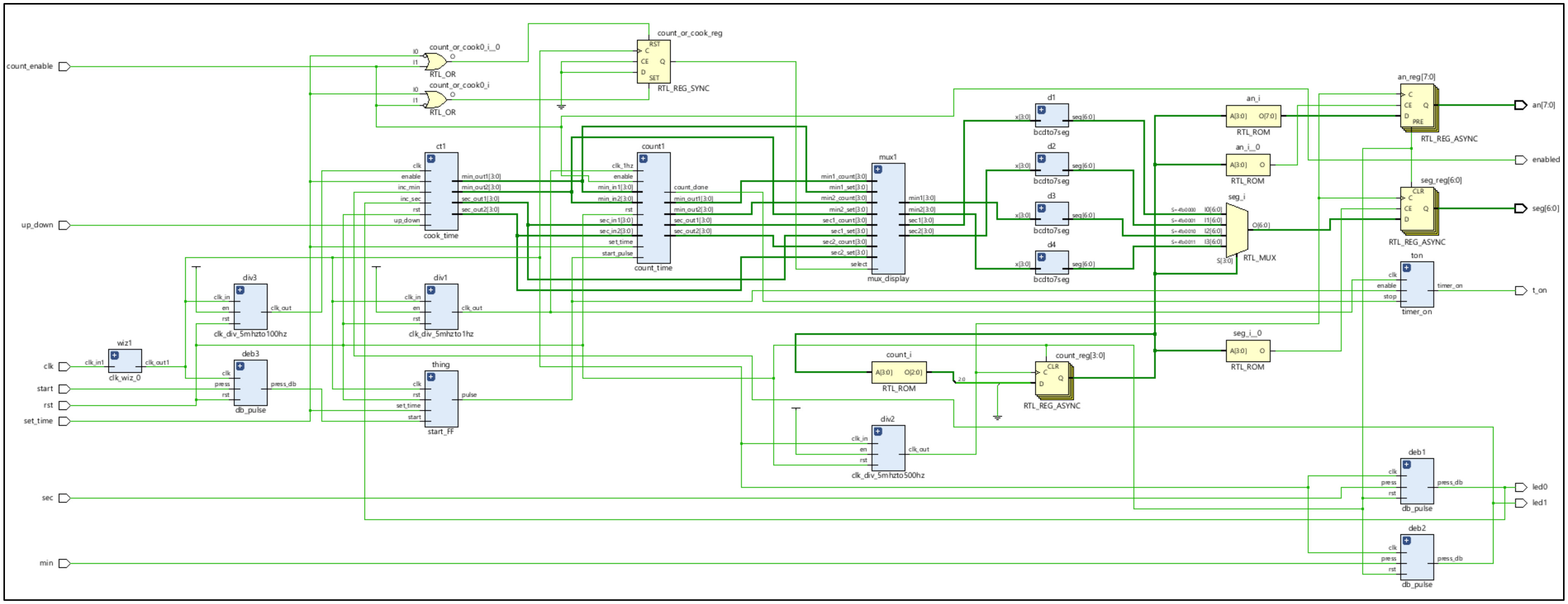


Figure 1 Elaborated Schematic of Master Control (top module)

Detailed Block Descriptions

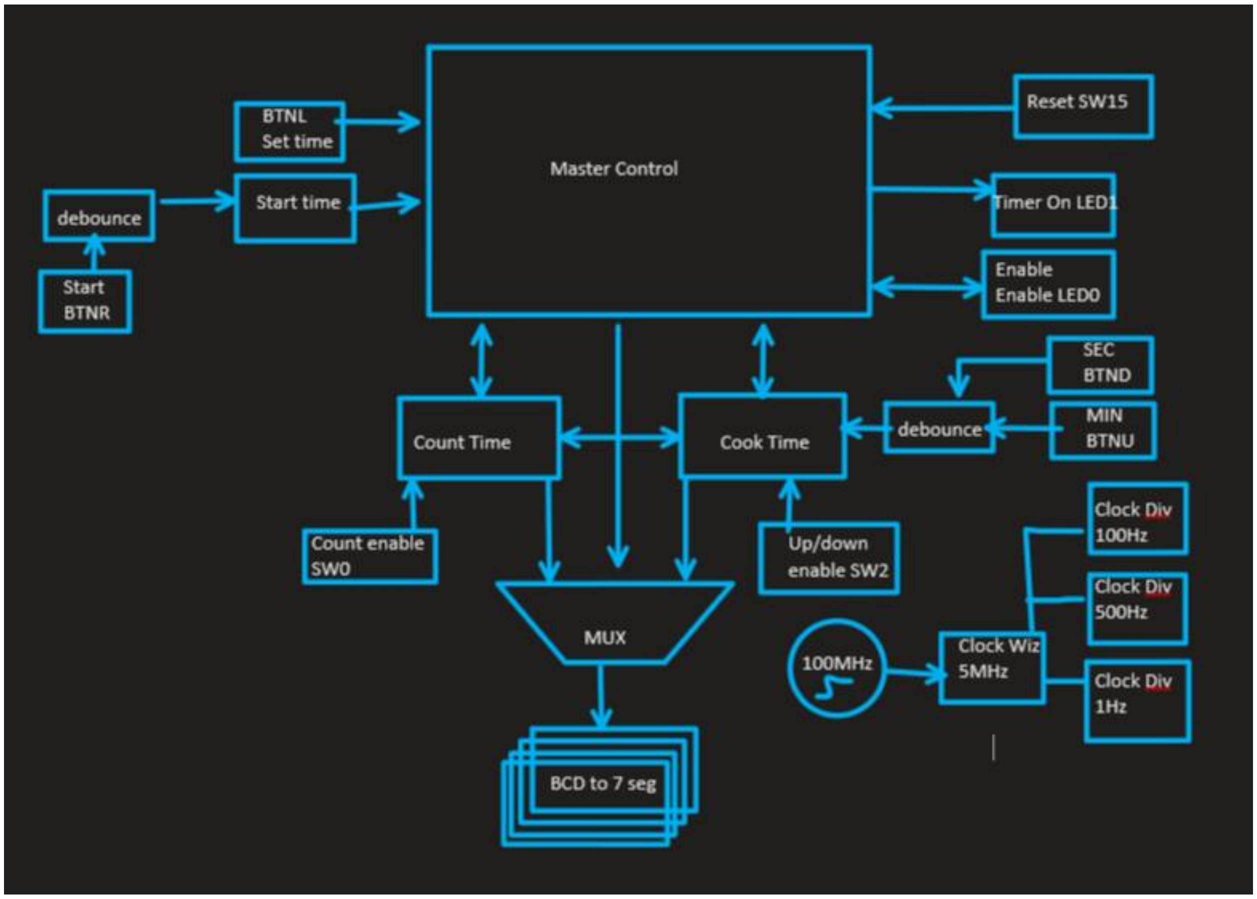


Figure 2 Block Diagram of the Egg Timer circuit

Master Control

The mater control module serves as the top-level module for the project. All the submodule instances are created in the master control and ties all the inputs and outputs together to feed into the submodules. There is some logic performed in the master control which ultimately could be in its own submodule (the quad seven-segment display decoder/initializer). Further, the master control has inputs: clk, enable (SW0), reset (SW15), set_time (BTNL), minutes (BTNU), seconds (BTND), start_time (BTNR), and the innovation up_down (SW2). The outputs of this module are: count_enabled (LED0), cathodes [7:0], anodes [6:0], eggtimer_enabled (LED1), min (LED15), sec (LED14). The clocking wizard was used to divide the 100MHz system clock to 5MHz to ensure use of only 1 clock domain. The master controller also contains the MUX logic which decides if the count or cooktime needs to be displayed on the seven-segment display.

Clocks

The onboard 100MHz system clock which is routed using the .xdc constraint file is the main clock domain that is used and further divided down to 5MHz, 5000Hz, 100Hz, and 1Hz. This is because the system clock is too fast for the purposes of the eggtimer and cannot be used easily without lots of logical changes. This would reduce reliability which is why the clockdivider modules exist. The built-in clock divider uses Phase Locked Loop (PLL) to make it usable. The clockdivider modules have a threshold within it to serve as a counter-driven clock divider by having the outputs stay high for half of the desired period (50% duty cycle). The formula to get the divisor for clockdividers is $divisor = \frac{5MHz}{2 * f_{desired}}$.

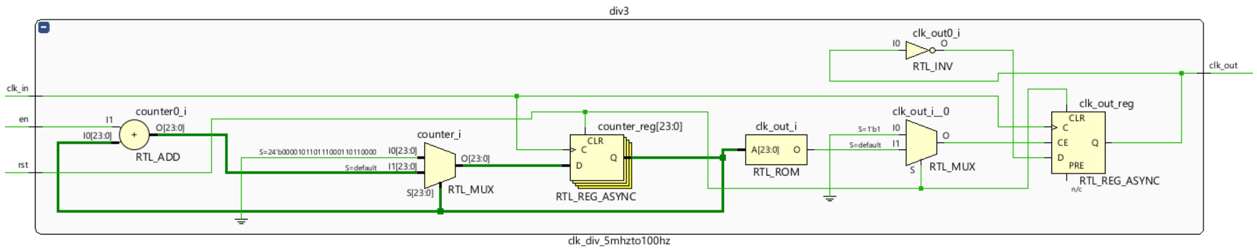


Figure 3 Clock Divider Elaborated Schematic

Debouncer

The debouncer as shown in the asynchronous lecture slides follow a simple standard. The input to the debouncer is the set_time button which is passed through 2 D-flipflops to synchronize the value to the 5MHz clock and then the outputs from both flipflops are passed through an XNOR gate to check for equality and then the output from the second D-flipflop and XNOR are passed into and AND gate to get the output. The debouncer makes use of a slower 100Hz clock because $t = \frac{1}{f} = 0.01s$ which is just fast enough to register user inputs but slow enough to not take each input from the user and multiple increments for counttimer. Running the debouncer at a higher frequency may create glitches in the circuit while the slower clocks would not let all inputs to be registered.

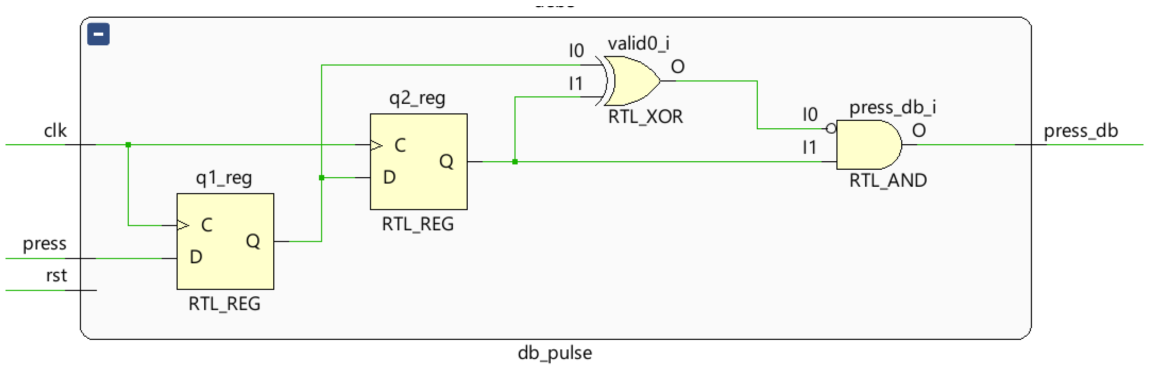


Figure 4 Debouncer Elaborated Schematic Design

Pulse Generator

The pulse generator acts as a synchronizer combined with some extra logic to generate a pulse of exactly one clock cycle. This is driven with the 5MHz clock to ensure the signals are properly synchronized. This is done by passing the signal through two D-flipflops and then each flipflop output is passed through an AND gate. This creates a pulse for every clock cycle, however, a millisecond pulse from the debouncer would last many pulses of the 5MHz signal which then can feed into count time and cook time modules to simplify the efficiency.

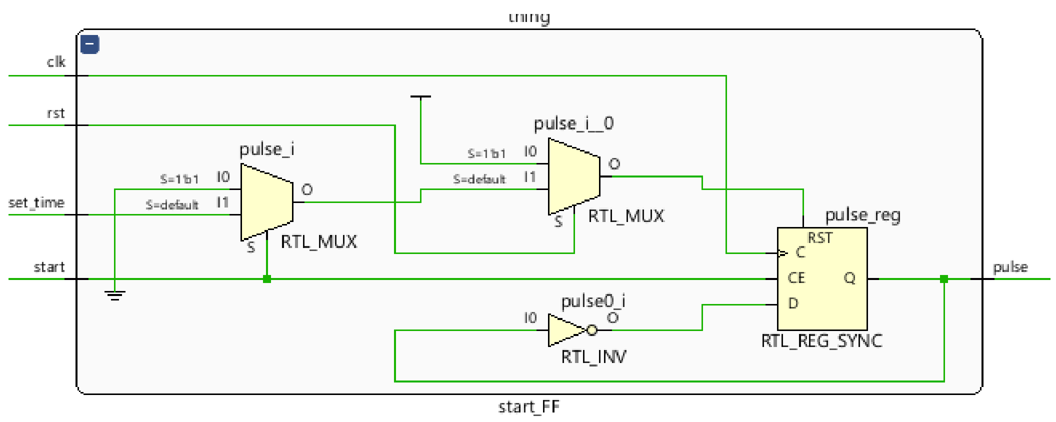


Figure 5 Pulse Generator Elaborated Schematic

Count Time

The count time module has 4 counters. From the LSB representing seconds to the MSB representing minutes. The LSB counter goes from 9 to 0 and loops back up to 9. When the loop around is happening, 1 is subtracted from the MSB which counts from 5 to 0 and back to 5. The seconds bits are tied to minutes, so after 60 seconds, the minutes decrement. The counter is complete when all values equal to 0 and the output is held at while the FPGA awaits for the new count values to be loaded in. When a new value is loaded in, the counter resets and starts counting and the outputs are once again selected from the mux and displayed on the segment display. The full schematic for count time can be found in the appendix.

Cook Time

The inputs to the cooktime module are clk, enable, reset, increment_second, increment_minute, and up_down. The outputs are seconds_out1, minutes_out1, seconds_out2, minutes_out2, which are 4-bit registers representing binary numbers from 0 to 9. The reset for the cook time allows the user to reset the time that has been set at any given time on the posedge of the clk. To decrement the time (innovation) allows the user to decrement the displayed time if the increment_minute/second are both enabled, along with the main enable signal and the up_down enable signal. The logic for the increment timer works the exact same way but without the up_down enable signal needing to be 1.

Timer On

The timer on module simply takes the start and count done signal from the rets of the circuit and displays a flashing LED to signify if the timer is counting down or stopped. The LED blinks at a rate of 1Hz only while the egg timer is counting down.

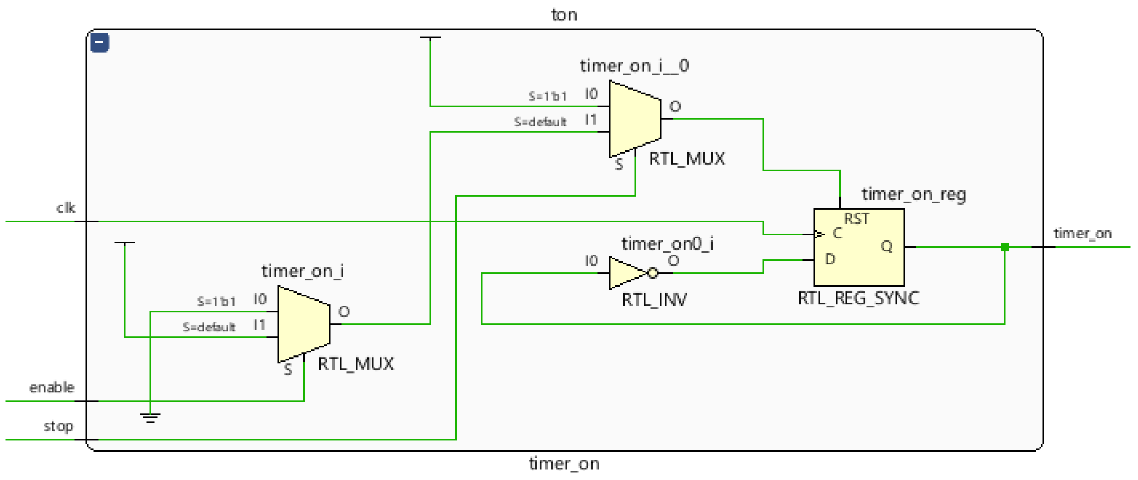


Figure 6 Timer on Elaborated Schematic

Display Logic and MUX

The display logic is made up of 3 portions, the MUX, BCD Decoder, and anode driver. The MUX selects between the outputs of count_time and cook_time with two 4-bit wire inputs and gives an output of one 4-bit bus (2 to 1 mux). The resultant signal from the MUX feeds into the BCD decoder and converts each 4-bit value from the bus into a Binary Coded Decimal of 7 bits corresponding with the cathodes of the seven-segment display. For example, 4'b0010 representing 1'd2 requires the cathodes to be 7'b0100100. The anodes can only be enabled once at a time which is why the 500Hz clock display is used to drive the anodes to make it seamless for the human experience. The anodes are enabled and take a new value with an active low signal.

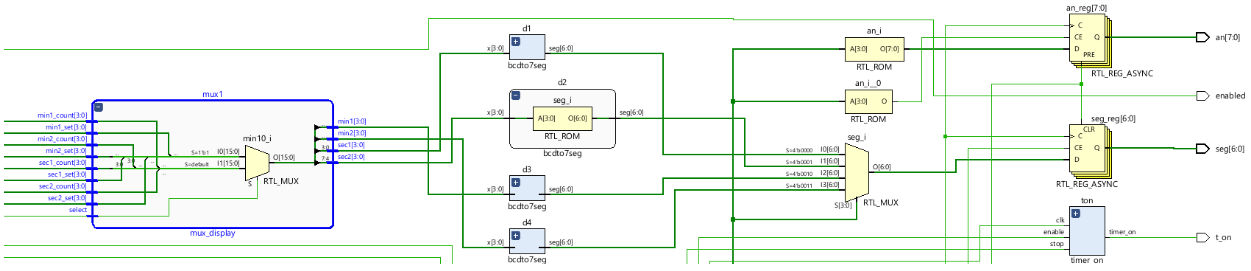


Figure 7 Display Logic and MUX display logic Elaborated Schematic

Evaluation (GA 4.7)

The design was evaluated at multiple points throughout the process due to changing requirements after building each block. Since each module was programmed separately, there was a need ensure compatibility between all of them and thus this caused us to re-evaluate how each module interacted with one another and what inputs would be fed in and what outputs are required. The final design which was used allowed for maximum flexibility amongst communicating modules.

Testbenches

To test the functionality of each module, a testbenches were individually developed for each module and tested to make troubleshooting easier. This was because, during the evaluation stage after making changes to the modules to make communication easier, there were errors within the code which caused improper functionality, however, with the use of the testbench simulations, allowed us to find common places where errors were occurring. While test cases for each possibility is not feasible, there were direct tests which were completed and then randomized to get the maximum coverage with minimal work. The modules which were proven to be successful in previous labs are not shown below.

Debouncer

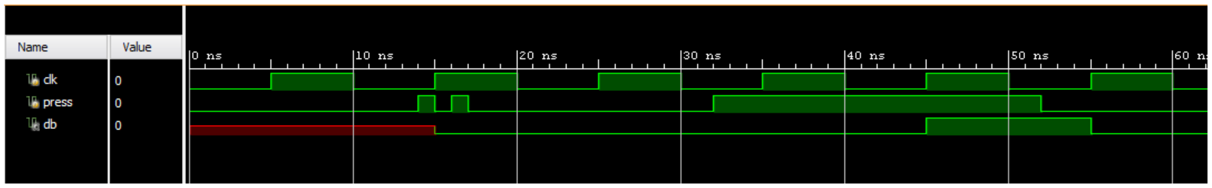


Figure 8 Debounce Module Test Bench

The debouncer is one of the most important functionalities of the eggtimer circuit, however, using software debouncers, reduces the need for a hardware module. The debouncer module was thoroughly tested to ensure the functionality would work on the board.

At 15ns, its noticeable that a few glitches occurred when it wasn't supposed to, this was because the pulse signals were not long enough and require a hold of 2 clock cycles to propagate to the output. This is why it is important to have a clock frequency which allows for multiple clocks within a second so that the user would not notice if it didn't work at the first clock cycle as it would be difficult to do things in under a second due to human limitations.

Pulse Generator

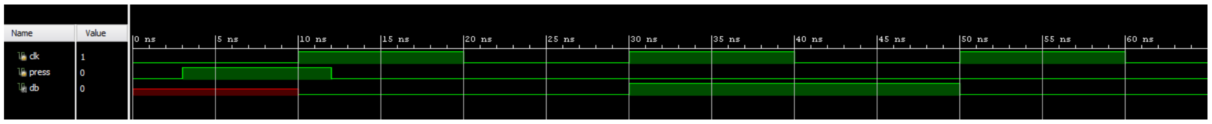


Figure 9 Pulse Generator Testbench

The pulse generator module synchronizes the inputs to generate a 1 clock cycle pulse. This is shown in the above figure.

At 10ns the input is latched and propagates to 1 clock cycle until 30ns. The resultant pulse is exactly one clock cycle in length.

Count Time

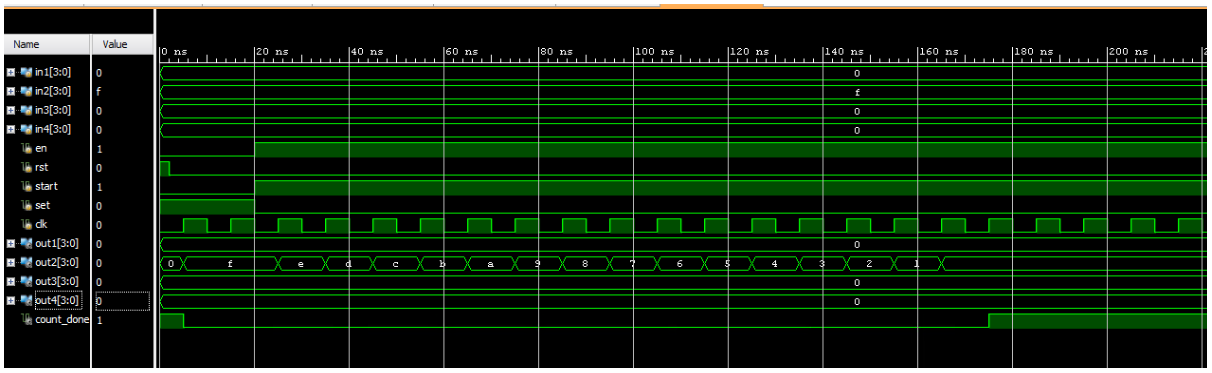


Figure 10 Count Time Testbench

The count time module is one of the core modules of this circuit and required extensive testing for all possible inputs to ensure functionality. Initially, the reset is checked to confirm functionality and makes all inputs go low. Following this, the count done signal is expected to go high on the following clock edge because all the inputs are 0. Once a number is loaded into the timer, it is expected that the output is decreasing if enable is high.

The reset starts off high and the count_done is 1 until the number is loaded into the register with set. The count does not start until the 20ns point when both enable and start are pulled high until the output reads 0 and then holds that value.

Cook Time

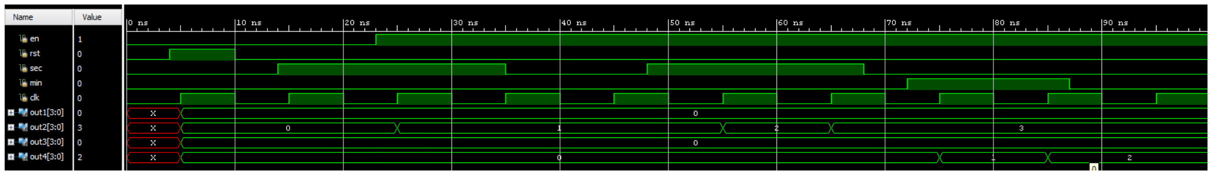


Figure 11 Cook Time Test Bench

The second key module for this program is to store and set the cooktime. When enable is low, the circuit should not activate. This ensures the user is only setting the time when they explicitly specify it with the press and hold of cooktime button. When the enable is high, the registers are set with the value by having pulses from the pulse generator module send a virtual enable increment signal. These values are stored and outputted.

The minutes and seconds both increment with each press of the button and reset sends everything back to 0. There is also no response when the enable signal is low.

Alternate Solutions (GA 4.6)

There are always ways to improve the design to enhance user experience. After considering how to improve, there were a few changes that were made: The first change that was made was regarding the start/stop logic for the timer. It is inconvenient for the user to use the "set" button to stop the timer because it also resets the time to the original set time. This allows the user to pause the time and restart from where it was stopped as most other real world timers work. The second change was implementing the innovation which was an up_down toggle switch for the user. This allows the user to decrement the time if they go past their desired time with the fewest amount of button clicks. These changes enhance user friendliness and functionality of the timer.

Start Toggle

The start toggle function is an improvement to the start pulse module which allows the start signal to toggle rather than hold.

Innovation

Up_down toggle

The up/down toggle is an extra signal that is fed into count_time that allows subtraction to the count registers. When this toggle switch is high, the register decrements the value.

Conclusion

Overall, the egg timer was successful implemented and tested against the outlined requirements that are mentioned in an earlier section. The results ended up meeting and surpassing the expectations as the functionality was improved upon to enhance the user experience by offering multiple solutions through accessibility and inclusivity. This solution was the final solution and it allowed me to learn how to properly implement multi module digital circuits while using test benches and actual hardware to compare results. With these lessons, the further improvements that can be made is to better understand the goals that were laid out in the lab manual to understand the project effectively.

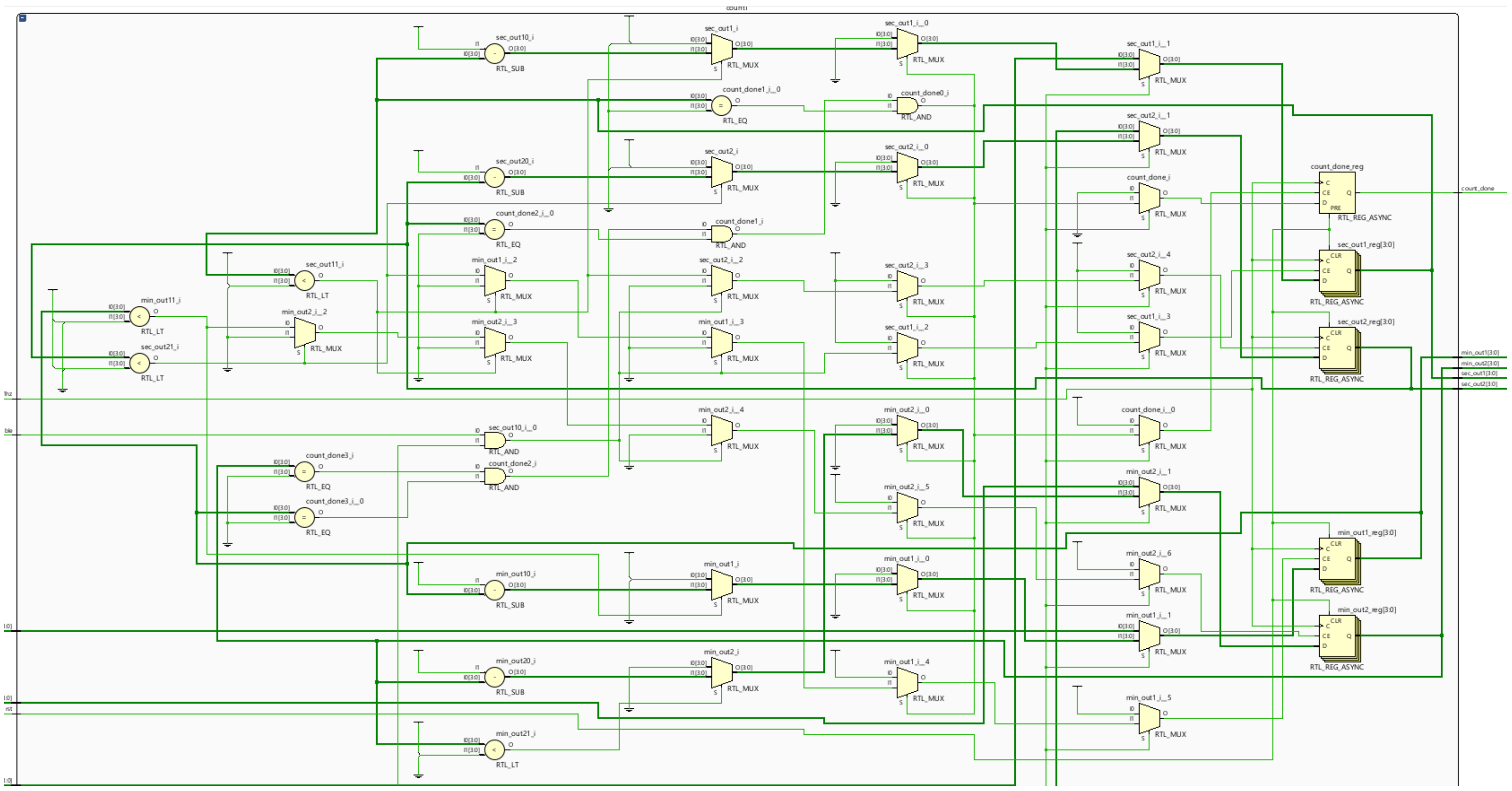
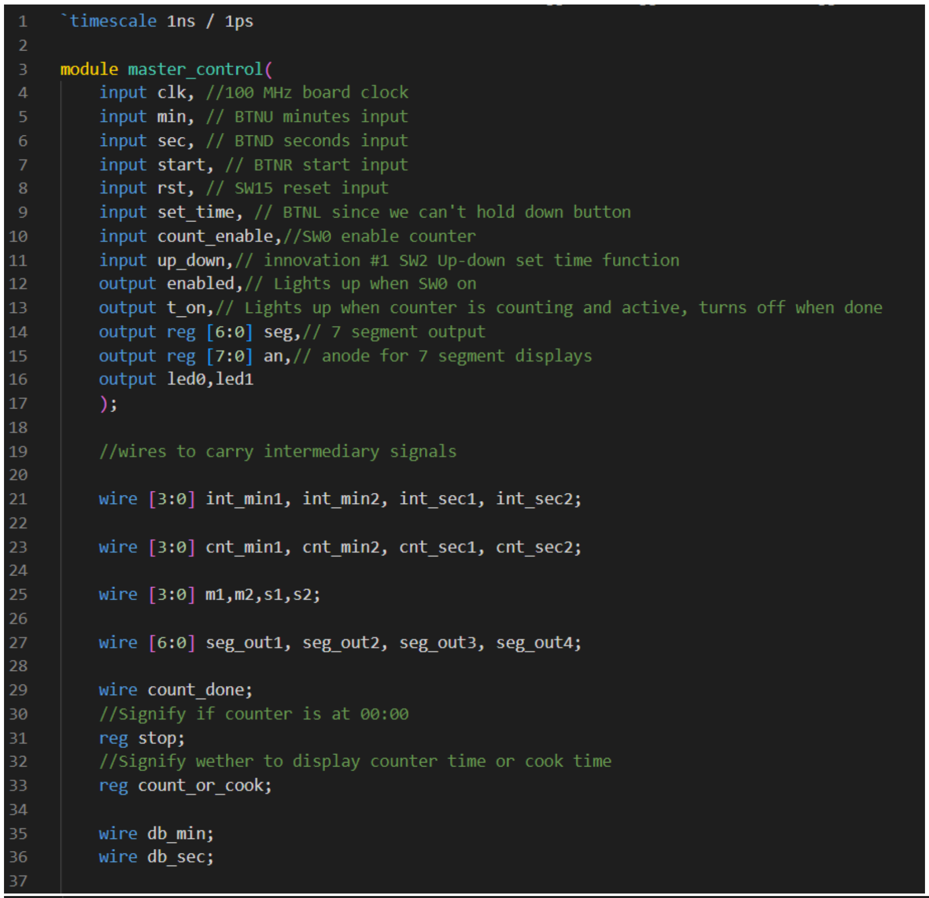
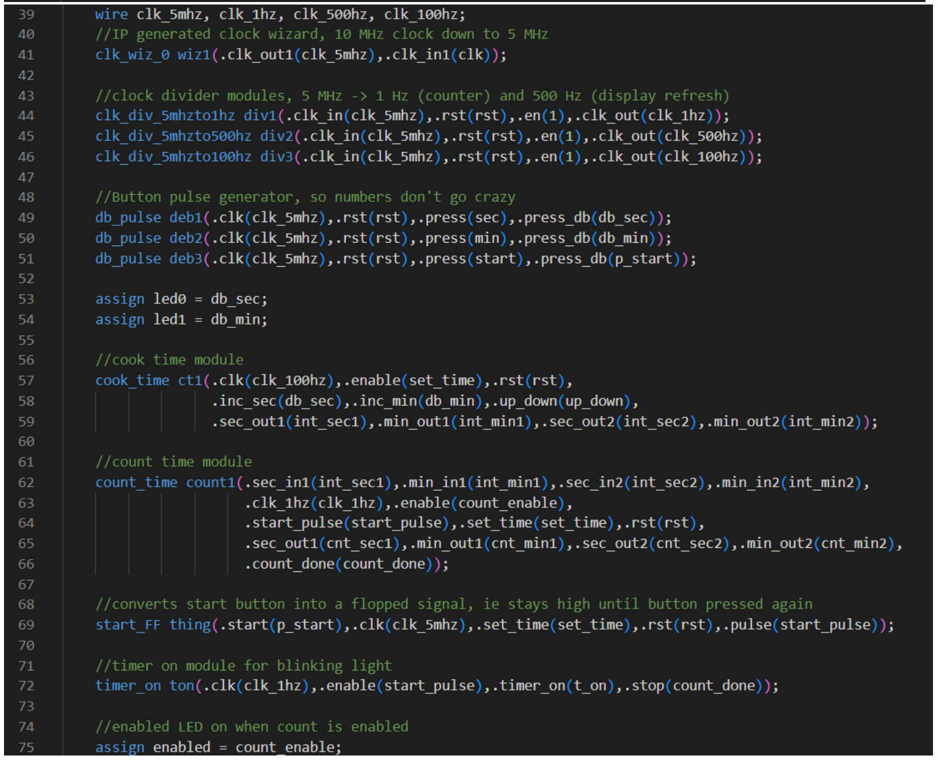


Figure 12 Count Time Elaborated Schematic

Master Control Code

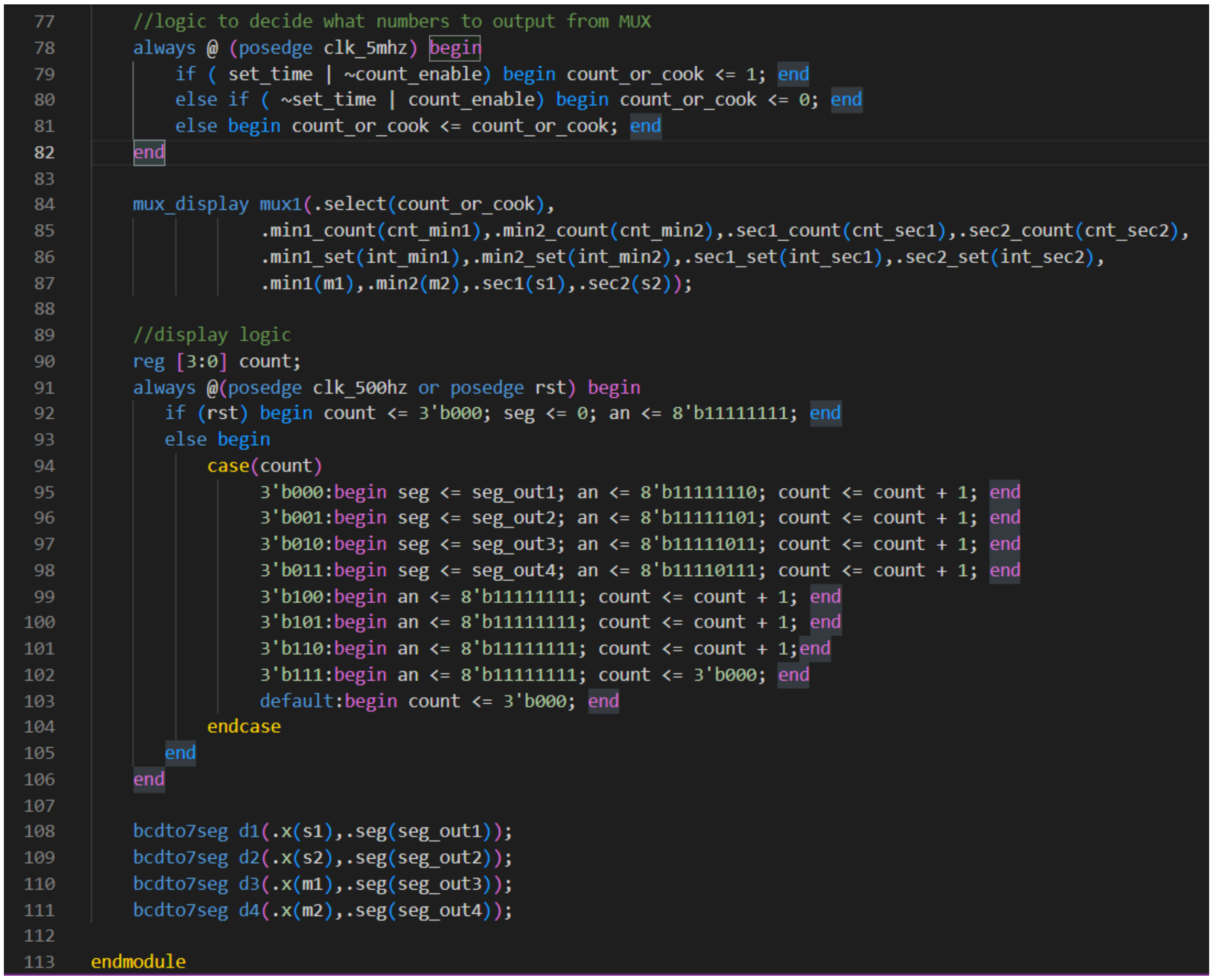


```verilog
`timescale 1ns / 1ps

module master_control(
    input clk, //100 MHz board clock
    input min, // BTNU minutes input
    input sec, // BTND seconds input
    input start, // BTNR start input
    input rst, // SW15 reset input
    input set_time, // BTNL since we can't hold down button
    input count_enable,//SW0 enable counter
    input up_down,// innovation #1 SW2 Up-down set time function
    output enabled,// Lights up when SW0 on
    output t_on,// Lights up when counter is counting and active, turns off when done
    output reg [6:0] seg,// 7 segment output
    output reg [7:0] an,// anode for 7 segment displays
    output led0,led1
    );

    //wires to carry intermediary signals

    wire [3:0] int_min1, int_min2, int_sec1, int_sec2;

    wire [3:0] cnt_min1, cnt_min2, cnt_sec1, cnt_sec2;

    wire [3:0] m1,m2,s1,s2;

    wire [6:0] seg_out1, seg_out2, seg_out3, seg_out4;

    wire count_done;
    //Signify if counter is at 00:00
    reg stop;
    //Signify wether to display counter time or cook time
    reg count_or_cook;

    wire db_min;
    wire db_sec;

```



```verilog
    wire clk_5mhz, clk_1hz, clk_500hz, clk_100hz;
    //IP generated clock wizard, 10 MHz clock down to 5 MHz
    clk_wiz_0 wiz1(.clk_out1(clk_5mhz),.clk_in1(clk));

    //clock divider modules, 5 MHz -> 1 Hz (counter) and 500 Hz (display refresh)
    clk_div_5mhzto1hz div1(.clk_in(clk_5mhz),.rst(rst),.en(1),.clk_out(clk_1hz));
    clk_div_5mhzto500hz div2(.clk_in(clk_5mhz),.rst(rst),.en(1),.clk_out(clk_500hz));
    clk_div_5mhzto100hz div3(.clk_in(clk_5mhz),.rst(rst),.en(1),.clk_out(clk_100hz));

    //Button pulse generator, so numbers don't go crazy
    db_pulse deb1(.clk(clk_5mhz),.rst(rst),.press(sec),.press_db(db_sec));
    db_pulse deb2(.clk(clk_5mhz),.rst(rst),.press(min),.press_db(db_min));
    db_pulse deb3(.clk(clk_5mhz),.rst(rst),.press(start),.press_db(p_start));

    assign led0 = db_sec;
    assign led1 = db_min;

    //cook time module
    cook_time ct1(.clk(clk_100hz),.enable(set_time),.rst(rst),
                  .inc_sec(db_sec),.inc_min(db_min),.up_down(up_down),
                  .sec_out1(int_sec1),.min_out1(int_min1),.sec_out2(int_sec2),.min_out2(int_min2));

    //count time module
    count_time count1(.sec_in1(int_sec1),.min_in1(int_min1),.sec_in2(int_sec2),.min_in2(int_min2),
                      .clk_1hz(clk_1hz),.enable(count_enable),
                      .start_pulse(start_pulse),.set_time(set_time),.rst(rst),
                      .sec_out1(cnt_sec1),.min_out1(cnt_min1),.sec_out2(cnt_sec2),.min_out2(cnt_min2),
                      .count_done(count_done));

    //converts start button into a flopped signal, ie stays high until button pressed again
    start_FF thing(.start(p_start),.clk(clk_5mhz),.set_time(set_time),.rst(rst),.pulse(start_pulse));

    //timer on module for blinking light
    timer_on ton(.clk(clk_1hz),.enable(start_pulse),.timer_on(t_on),.stop(count_done));

    //enabled LED on when count is enabled
    assign enabled = count_enable;
```

Master Control Code CNTD

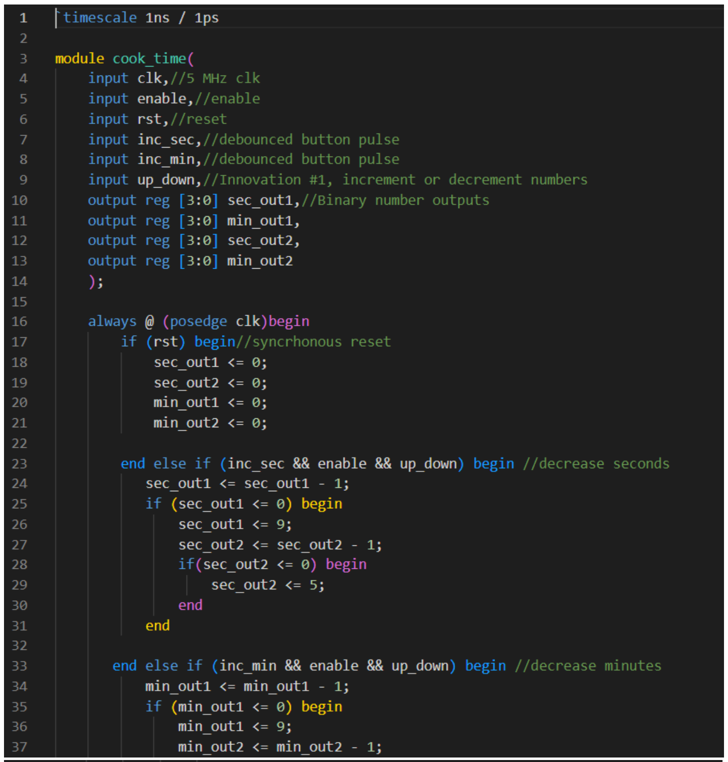


```
    //logic to decide what numbers to output from MUX
    always @ (posedge clk_5mhz) begin
        if ( set_time | ~count_enable) begin count_or_cook <= 1; end
        else if ( ~set_time | count_enable) begin count_or_cook <= 0; end
        else begin count_or_cook <= count_or_cook; end
    end

    mux_display mux1(.select(count_or_cook),
                .min1_count(cnt_min1),.min2_count(cnt_min2),.sec1_count(cnt_sec1),.sec2_count(cnt_sec2),
                .min1_set(int_min1),.min2_set(int_min2),.sec1_set(int_sec1),.sec2_set(int_sec2),
                .min1(m1),.min2(m2),.sec1(s1),.sec2(s2));

    //display logic
    reg [3:0] count;
    always @(posedge clk_500hz or posedge rst) begin
       if (rst) begin count <= 3'b000; seg <= 0; an <= 8'b11111111; end
       else begin
           case(count)
               3'b000:begin seg <= seg_out1; an <= 8'b11111110; count <= count + 1; end
               3'b001:begin seg <= seg_out2; an <= 8'b11111101; count <= count + 1; end
               3'b010:begin seg <= seg_out3; an <= 8'b11111011; count <= count + 1; end
               3'b011:begin seg <= seg_out4; an <= 8'b11110111; count <= count + 1; end
               3'b100:begin an <= 8'b11111111; count <= count + 1; end
               3'b101:begin an <= 8'b11111111; count <= count + 1; end
               3'b110:begin an <= 8'b11111111; count <= count + 1;end
               3'b111:begin an <= 8'b11111111; count <= 3'b000; end
               default:begin count <= 3'b000; end
           endcase
       end
    end

    bcdto7seg d1(.x(s1),.seg(seg_out1));
    bcdto7seg d2(.x(s2),.seg(seg_out2));
    bcdto7seg d3(.x(m1),.seg(seg_out3));
    bcdto7seg d4(.x(m2),.seg(seg_out4));

endmodule
```

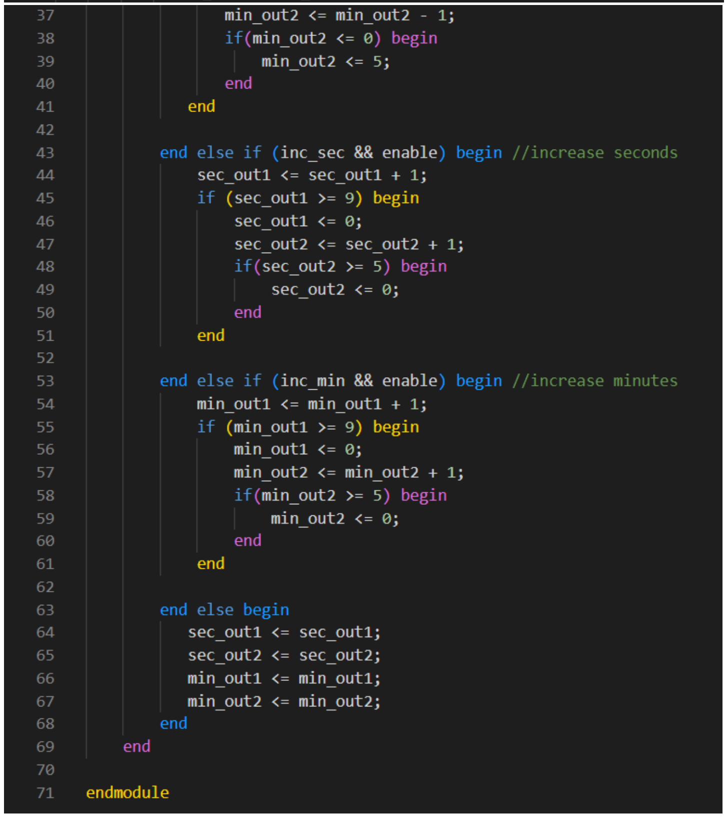
ELEC3500 Egg Timer

Tawseef Patel
101145333

Cook Time Code



```verilog
`timescale 1ns / 1ps

module cook_time(
    input clk,//5 MHz clk
    input enable,//enable
    input rst,//reset
    input inc_sec,//debounced button pulse
    input inc_min,//debounced button pulse
    input up_down,//Innovation #1, increment or decrement numbers
    output reg [3:0] sec_out1,//Binary number outputs
    output reg [3:0] min_out1,
    output reg [3:0] sec_out2,
    output reg [3:0] min_out2
    );

    always @ (posedge clk)begin
        if (rst) begin//syncrhonous reset
            sec_out1 <= 0;
            sec_out2 <= 0;
            min_out1 <= 0;
            min_out2 <= 0;

        end else if (inc_sec && enable && up_down) begin //decrease seconds
           sec_out1 <= sec_out1 - 1;
           if (sec_out1 <= 0) begin
               sec_out1 <= 9;
               sec_out2 <= sec_out2 - 1;
               if(sec_out2 <= 0) begin
                   sec_out2 <= 5;
               end
           end

       end else if (inc_min && enable && up_down) begin //decrease minutes
           min_out1 <= min_out1 - 1;
           if (min_out1 <= 0) begin
               min_out1 <= 9;
               min_out2 <= min_out2 - 1;
```

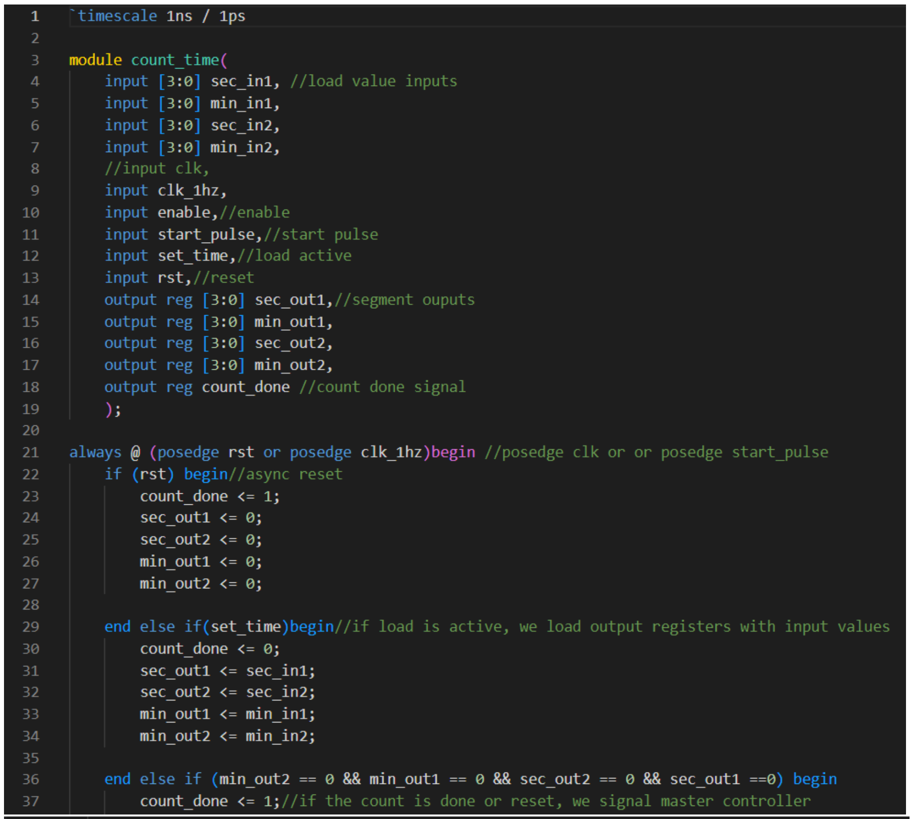


min_out2 <= min_out2 - 1;

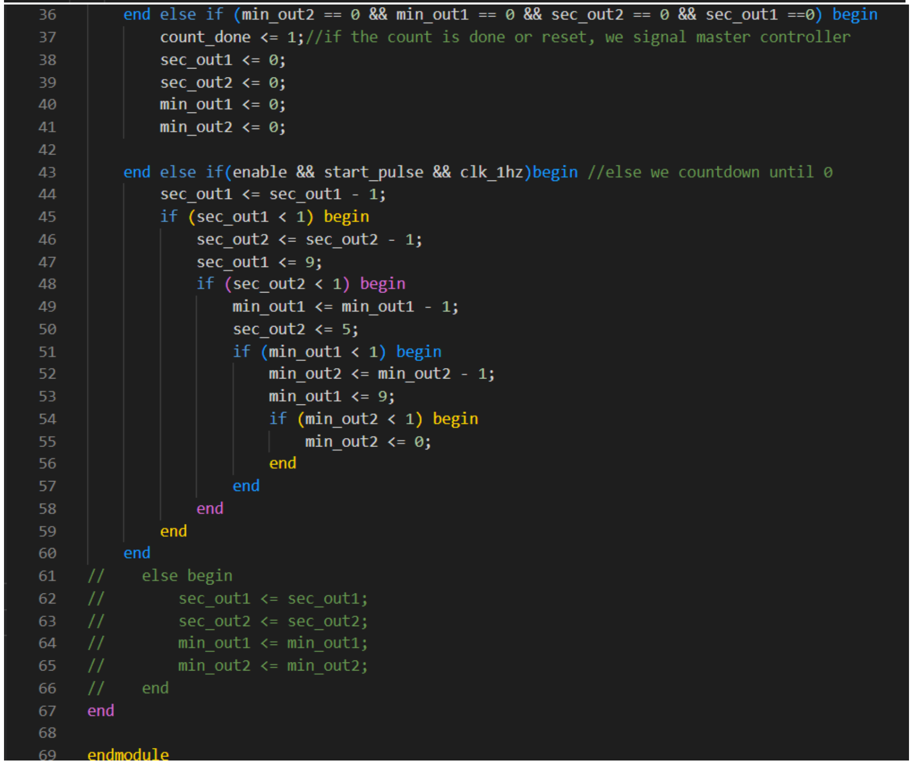
```verilog
               if(min_out2 <= 0) begin
                   min_out2 <= 5;
               end
           end

        end else if (inc_sec && enable) begin //increase seconds
            sec_out1 <= sec_out1 + 1;
            if (sec_out1 >= 9) begin
                sec_out1 <= 0;
                sec_out2 <= sec_out2 + 1;
                if(sec_out2 >= 5) begin
                    sec_out2 <= 0;
                end
            end

        end else if (inc_min && enable) begin //increase minutes
            min_out1 <= min_out1 + 1;
            if (min_out1 >= 9) begin
                min_out1 <= 0;
                min_out2 <= min_out2 + 1;
                if(min_out2 >= 5) begin
                    min_out2 <= 0;
                end
            end

        end else begin
           sec_out1 <= sec_out1;
           sec_out2 <= sec_out2;
           min_out1 <= min_out1;
           min_out2 <= min_out2;
        end
    end

endmodule
```

Tawseef Patel
101145333

Count Time Code



```verilog
`timescale 1ns / 1ps

module count_time(
    input [3:0] sec_in1, //load value inputs
    input [3:0] min_in1,
    input [3:0] sec_in2,
    input [3:0] min_in2,
    //input clk,
    input clk_1hz,
    input enable,//enable
    input start_pulse,//start pulse
    input set_time,//load active
    input rst,//reset
    output reg [3:0] sec_out1,//segment ouputs
    output reg [3:0] min_out1,
    output reg [3:0] sec_out2,
    output reg [3:0] min_out2,
    output reg count_done //count done signal
    );

always @ (posedge rst or posedge clk_1hz)begin //posedge clk or or posedge start_pulse
    if (rst) begin//async reset
        count_done <= 1;
        sec_out1 <= 0;
        sec_out2 <= 0;
        min_out1 <= 0;
        min_out2 <= 0;

    end else if(set_time)begin//if load is active, we load output registers with input values
        count_done <= 0;
        sec_out1 <= sec_in1;
        sec_out2 <= sec_in2;
        min_out1 <= min_in1;
        min_out2 <= min_in2;

    end else if (min_out2 == 0 && min_out1 == 0 && sec_out2 == 0 && sec_out1 ==0) begin
        count_done <= 1;//if the count is done or reset, we signal master controller
```

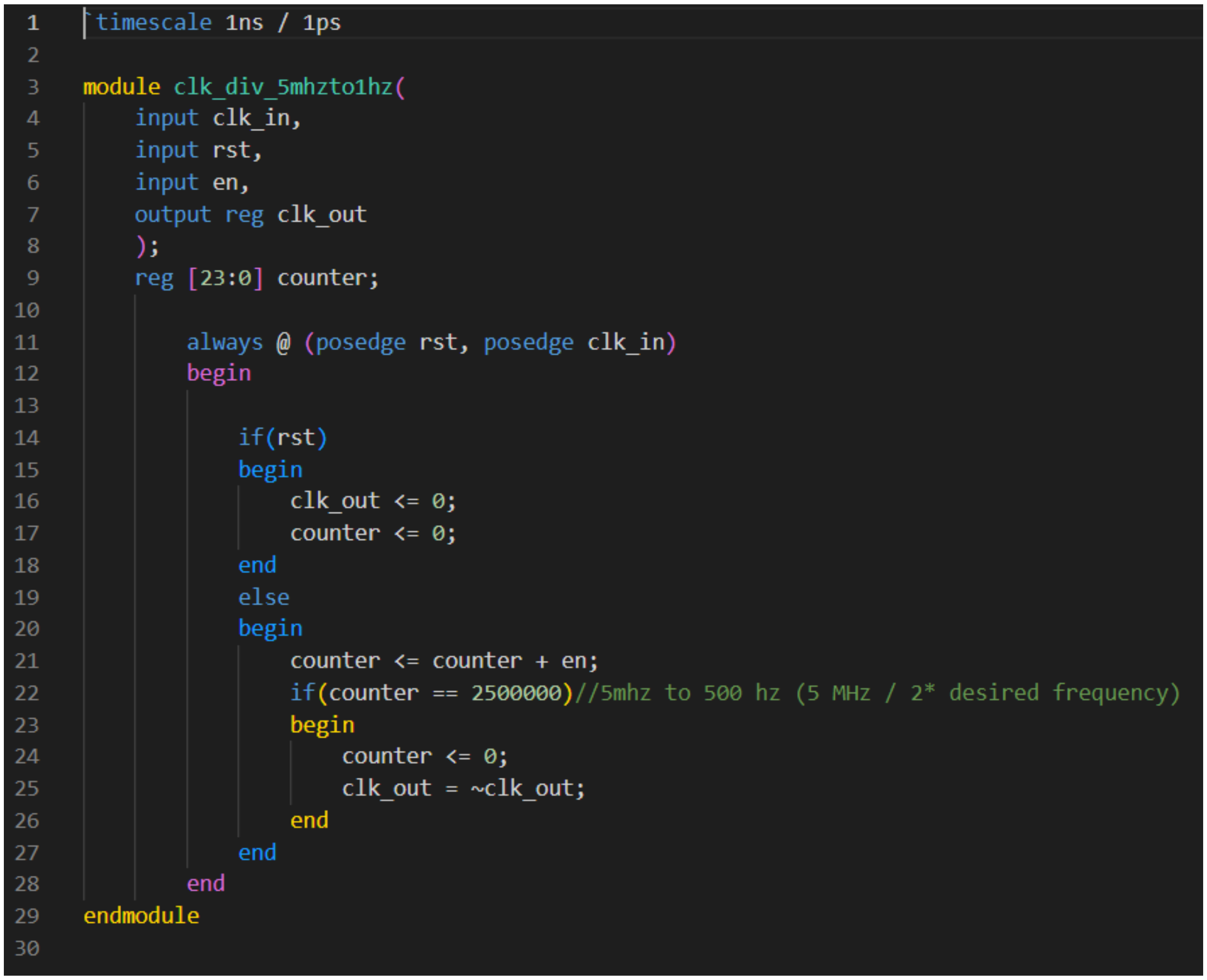


```verilog
    end else if (min_out2 == 0 && min_out1 == 0 && sec_out2 == 0 && sec_out1 ==0) begin
        count_done <= 1;//if the count is done or reset, we signal master controller
```

```verilog
        sec_out1 <= 0;
        sec_out2 <= 0;
        min_out1 <= 0;
        min_out2 <= 0;

    end else if(enable && start_pulse && clk_1hz)begin //else we countdown until 0
        sec_out1 <= sec_out1 - 1;
        if (sec_out1 < 1) begin
            sec_out2 <= sec_out2 - 1;
            sec_out1 <= 9;
            if (sec_out2 < 1) begin
                min_out1 <= min_out1 - 1;
                sec_out2 <= 5;
                if (min_out1 < 1) begin
                    min_out2 <= min_out2 - 1;
                    min_out1 <= 9;
                    if (min_out2 < 1) begin
                        min_out2 <= 0;
                    end
                end
            end
        end
    end
//    else begin
//        sec_out1 <= sec_out1;
//        sec_out2 <= sec_out2;
//        min_out1 <= min_out1;
//        min_out2 <= min_out2;
//    end
end

endmodule
```

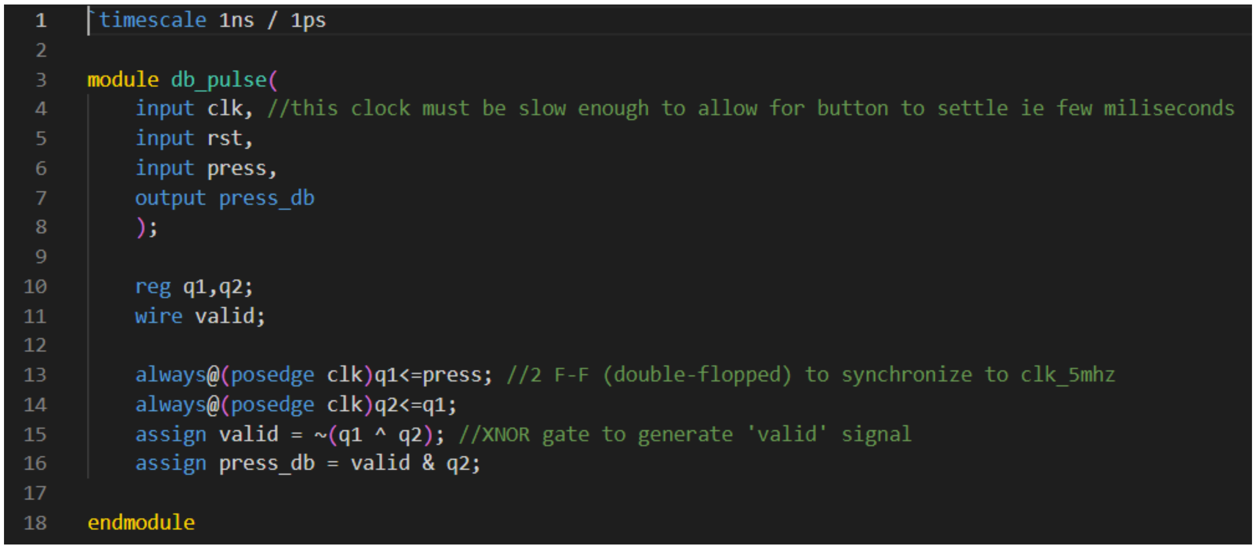
Tawseef Patel
101145333

Clock Divider Code



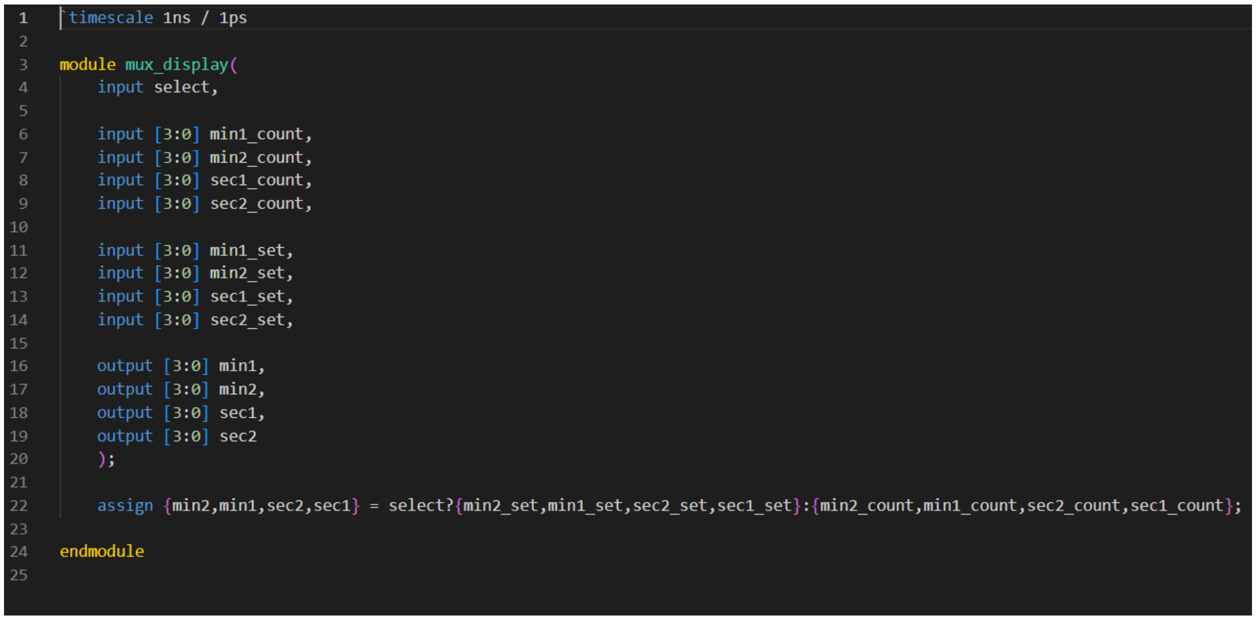
```
`timescale 1ns / 1ps

module clk_div_5mhzto1hz(
    input clk_in,
    input rst,
    input en,
    output reg clk_out
    );
    reg [23:0] counter;

        always @ (posedge rst, posedge clk_in)
        begin

            if(rst)
            begin
                clk_out <= 0;
                counter <= 0;
            end
            else
            begin
                counter <= counter + en;
                if(counter == 2500000)//5mhz to 500 hz (5 MHz / 2* desired frequency)
                begin
                    counter <= 0;
                    clk_out = ~clk_out;
                end
            end
        end
endmodule
```

Debounce Code



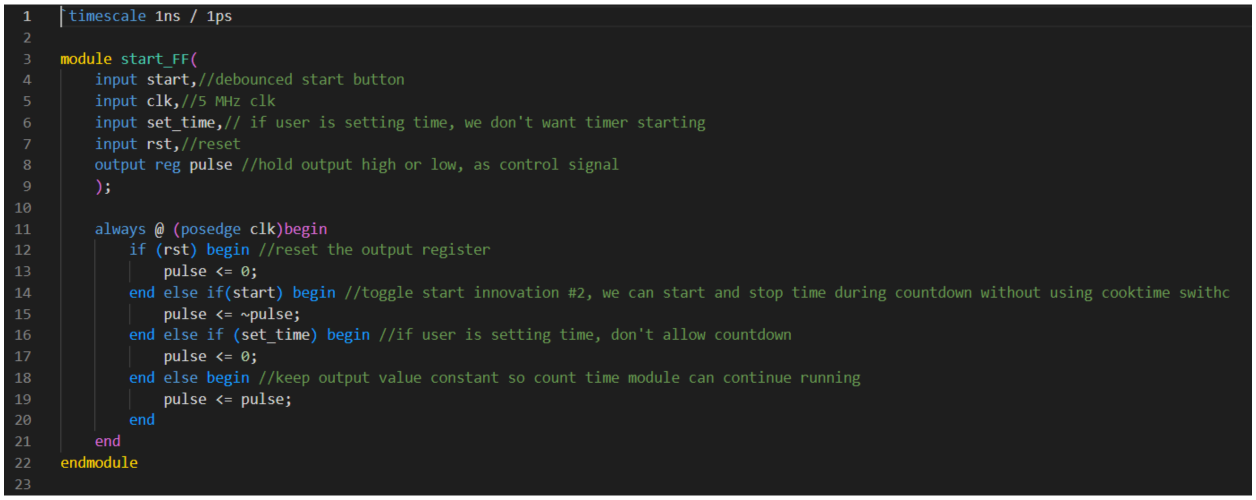
```
`timescale 1ns / 1ps

module db_pulse(
    input clk, //this clock must be slow enough to allow for button to settle ie few miliseconds
    input rst,
    input press,
    output press_db
    );

    reg q1,q2;
    wire valid;

    always@(posedge clk)q1<=press; //2 F-F (double-flopped) to synchronize to clk_5mhz
    always@(posedge clk)q2<=q1;
    assign valid = ~(q1 ^ q2); //XNOR gate to generate 'valid' signal
    assign press_db = valid & q2;

endmodule
```

Mux Control



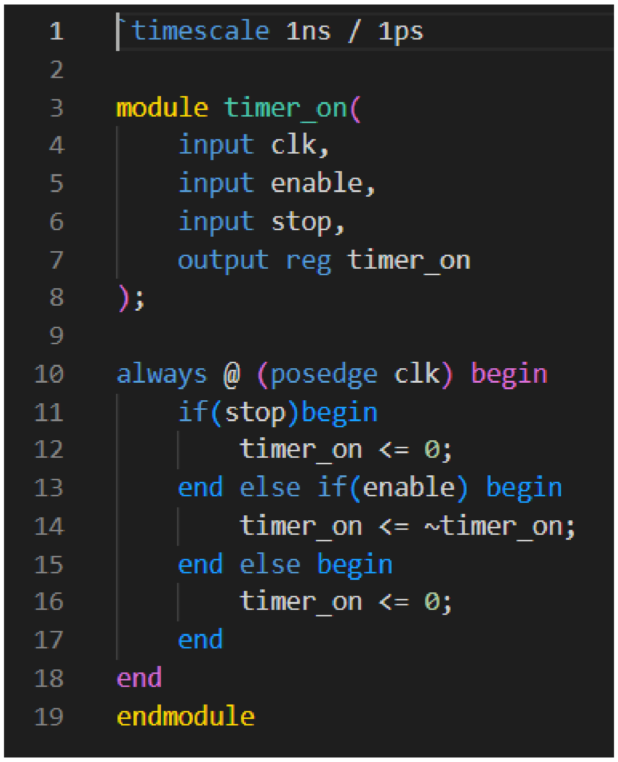
```
`timescale 1ns / 1ps

module mux_display(
    input select,

    input [3:0] min1_count,
    input [3:0] min2_count,
    input [3:0] sec1_count,
    input [3:0] sec2_count,

    input [3:0] min1_set,
    input [3:0] min2_set,
    input [3:0] sec1_set,
    input [3:0] sec2_set,

    output [3:0] min1,
    output [3:0] min2,
    output [3:0] sec1,
    output [3:0] sec2
    );

    assign {min2,min1,sec2,sec1} = select?{min2_set,min1_set,sec2_set,sec1_set}:{min2_count,min1_count,sec2_count,sec1_count};

endmodule

```

Start FF / Pulse Gen Code Code



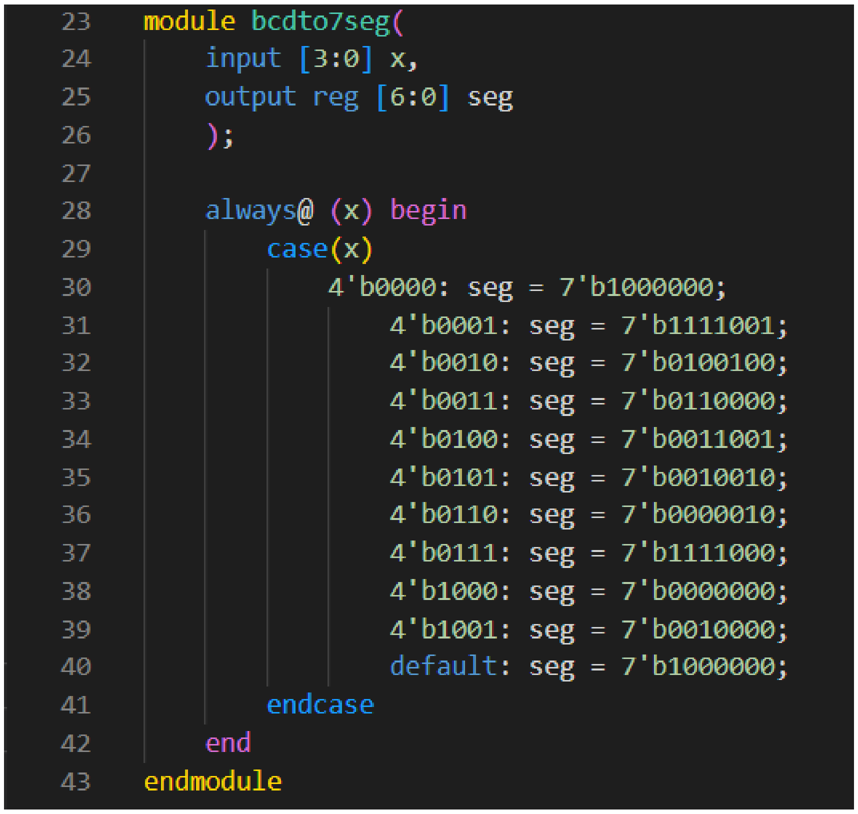
```
`timescale 1ns / 1ps

module start_FF(
    input start,//debounced start button
    input clk,//5 MHz clk
    input set_time,// if user is setting time, we don't want timer starting
    input rst,//reset
    output reg pulse //hold output high or low, as control signal
    );

    always @ (posedge clk)begin
        if (rst) begin //reset the output register
            pulse <= 0;
        end else if(start) begin //toggle start innovation #2, we can start and stop time during countdown without using cooktime swithc
            pulse <= ~pulse;
        end else if (set_time) begin //if user is setting time, don't allow countdown
            pulse <= 0;
        end else begin //keep output value constant so count time module can continue running
            pulse <= pulse;
        end
    end
endmodule

```

Timer On Code



```
`timescale 1ns / 1ps

module timer_on(
    input clk,
    input enable,
    input stop,
    output reg timer_on
);

always @ (posedge clk) begin
    if(stop)begin
        timer_on <= 0;
    end else if(enable) begin
        timer_on <= ~timer_on;
    end else begin
        timer_on <= 0;
    end
end
endmodule
```

BCD to 7 Segment Display



```
module bcdto7seg(
    input [3:0] x,
    output reg [6:0] seg
    );

    always@ (x) begin
        case(x)
            4'b0000: seg = 7'b1000000;
                4'b0001: seg = 7'b1111001;
                4'b0010: seg = 7'b0100100;
                4'b0011: seg = 7'b0110000;
                4'b0100: seg = 7'b0011001;
                4'b0101: seg = 7'b0010010;
                4'b0110: seg = 7'b0000010;
                4'b0111: seg = 7'b1111000;
                4'b1000: seg = 7'b0000000;
                4'b1001: seg = 7'b0010000;
                default: seg = 7'b1000000;
        endcase
    end
endmodule
```