Stopwatch Lab
ELEC 3500-B5

Thursday April 11th, 2019
Ryan Frohar 101029053
Robert Irwin 101036941

I. Block Diagrams

To view the block diagrams please reference Appendix B. To obtain this PDF, we simply ran a schematic through vivado and right clicked on the figure to “export as PDF”.

II. Design and Operational Principles

For the most part, the lab 10 manual told us how to approach the design and operational principles. The main principle to our project was the master controller which ran our counter, seven segment display and the PMOD Rotary Encoder. The master controller received button presses as well as the mode select and microblaze interface signals. Based on these inputs it would: generate signals that were sent to the time block to start/stop the timer and also determine which data to display on the seven segment display. Refer to figure 1 below for a detailed image of how our principles related to each other.

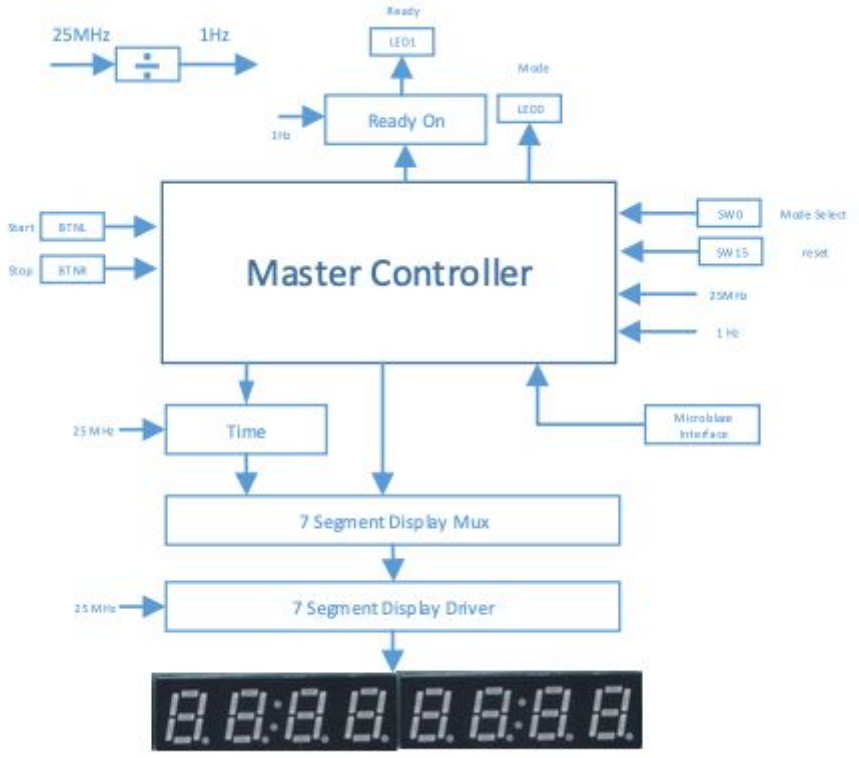

Figure 1: Master Controller [1]

III. Design Choices

To start the design process, we started with creating the up/down counter portion first with the master controller as well. To begin, the master controller was only used to have the basic framework for the stop watch. Once this portion worked we updated the top module of our program to include the other aspects of our

stopwatch. We made sure to include everything in our project first and ensured it worked before adding the PMOD in. To ensure that adding the PMOD wouldn't completely destroy the rest of the program, we designed the counter so that each segment would count independently as opposed to one large binary number.
To include the PMOD, we worked countless days but were not able to get it to work properly. We then came to the conclusion of focussing on the innovation aspects instead. These including modifying our timer to display as *HH:MM:SS.mm* instead of default configuration of *MM:SS.mmmm*, as well as creating a beep whenever the countdown timer reaches zero [2].

IV. Verilog/Software code

To see our code please reference appendix A. For the most part, some of our code for example: clock divider and up/down counter was taken from previous labs although the master controller and PMOD encoder were completely new features that needed to be implemented into our stopwatch.

V. Verification (e.g. Simulations and Test Benches)

An issue we ran into was the counter was not working. It wouldn't display any values on the seven segment display. We used a testbench to ensure the counter was working fine, and after a few small tweaks it was working as expected. It wasn't until we looked at the schematic that we saw there were some inputs not connected and that the segment and anode values were set at constants. Other less frequent included Vivado bug errors which were easily fixed by restarting the entire application. Evidently this became an annoyance although the teaching assistants explained that there was no other reasonable explanation for the problems that we saw.

VI. Hardware Testing

Hardware testing is a crucial part in the success of the project. Before starting, we evaluated the differences in testing types. After researching the lecture notes, we came to the conclusion of either using Constrained Random Testing or Directed Testing. Constrained Random Testing essentially automates/randomizes the stimulus in order to find bugs that you never anticipated. Directed Testing is an incremental approach to testing where the user tests new features as they are developed/implemented. We decided that directed testing was the approach for us because we liked the idea of getting small parts to work first before testing the entire project. This saved us tons of time compared to if we were to write/undergo a bunch of tests at the end of the project in order to test specific features.

VII. Final Hardware Configuration

Our final hardware configuration included: an up/down counting Stopwatch and a PMOD Rotary Encoder to manually modify individual segments (for example:

milliseconds, seconds, minutes) of the stopwatch. The PMOD Rotary Encoder also included a switch to activate the feature, essentially used as a safety feature so that the stopwatch could not be manipulated by accident.

VIII. Lessons Learned

Throughout the lab, an immense amount of lessons were learned. The stopwatch lab was the last lab and it was evident that information from previous labs were combined into this one. At the beginning of this course, the thought of programming an FPGA devastated us, but now we can finally say that we created, through many modules, a well working stopwatch using VHDL.

On a separate note, the PMOD Rotary Encoder was an extremely hard feature to implement into our stopwatch. After countless days and help from teaching assistants we came to the conclusion that it was never going to work. In retrospect we should have spent some initial time researching online on how the PMOD feature actually works because the tutorial in lab 8 did not give us enough information to implement the feature into our stopwatch design.

IX. Conclusions

In conclusion, this lab and many others, helped us learn about the different industry leading tools that could help us in our future endeavours. Tools like Vivado and hands on experience with an FPGA are skills that not many students can say they have learned about so we are grateful.

X. References

[1] Department of Electronics. (2019 Winter). ELEC 3500 Lab 10 Manual. Ottawa, Ontario: Carleton University. Retrieved from class website at https://culearn.carleton.ca/moodle/course/view.php?id=117887

[2] (n.d.). Retrieved April 8, 2019 from https://www.fpga4fun.com/MusicBox2.html

Appendix A - Verilog Code

A.1 - Master Controller

```
module Master_Controller(
    input clk,
    input mode,
    input reset,
    input start,
    input stop,
    output reg ready,
    output reg [7:0] seg,
    output reg [7:0] an,
    output reg speaker,

    input enc_a,
    input enc_b,
    input enc_sw,
    input enc_btn
   );
    //States
    reg [2:0] state, nextstate;
    parameter upWait=0, downWait=1, up=2, down=3, resetState=4;

    // Counter Control
    reg load;
    reg enable;
    reg updn;
    initial begin load = 0; enable = 0; end

    // Display Control
    integer display;
    initial display = 0;

    // Master values
    reg [3:0] v_mil1, v_mil2, v_sec1, v_sec2, v_min1, v_min2, v_hr1, v_hr2;
    initial begin
        v_mil1 = 4'b0;
        v_mil2 = 4'b0;
        v_sec1 = 4'b0;
        v_sec2 = 4'b0;
        v_min1 = 4'b0;
        v_min2 = 4'b0;
        v_hr1 = 4'b0;
        v_hr2 = 4'b0;
    end
```

```
    // Encoder values
    reg [2:0] section;
    wire [3:0] vSet_mil1, vSet_mil2, vSet_sec1, vSet_sec2, vSet_min1, vSet_min2,
vSet_hr1, vSet_hr2;

    // Up/Down Counter values
    wire [3:0] vUpDown_mil1, vUpDown_mil2, vUpDown_sec1, vUpDown_sec2,
vUpDown_min1, vUpDown_min2, vUpDown_hr1, vUpDown_hr2;

    // Clock Divider
    wire hun_hz_clk;
    wire ten_khz_clk;
    wire refresh_clk;
    clk_divider
clk_div(.clk(clk),.hun_hz_clk(hun_hz_clk),.ten_khz_clk(ten_khz_clk),.refresh_clk(refresh_c
lk));

    // Speaker
    wire speaker_out;
    reg sound_en;

    always @(posedge refresh_clk or posedge reset)
        if(reset) state <= resetState;
        else state <= nextstate;

    always @(state or start or mode or stop)
        case(state)
            upWait: begin
                if(start & mode) nextstate = up;
                else if(!mode) nextstate = downWait;
                else nextstate = upWait;
            end
            downWait: begin
                if(start & !mode) nextstate = down;
                else if (mode) nextstate = upWait;
                else nextstate = downWait;
            end
            up: begin
                if(stop) nextstate = upWait;
                else nextstate = up;
            end
            down: begin
                if(stop) nextstate = upWait;
                else nextstate = down;
            end
```

```
        resetState: begin
            if(mode) nextstate = upWait;
            else if(!mode) nextstate = downWait;
            else nextstate = resetState;
        end
        default nextstate = upWait;
    endcase

always @(*)
    case(state)
        upWait: begin
            load = 1'b1;
            ready = 1'b1;
            enable = 'b0;
            updn = 1'b1;
            v_mil1 = (enc_sw) ? vSet_mil1 : vUpDown_mil1;
            v_mil2 = (enc_sw) ? vSet_mil2 : vUpDown_mil2;
            v_sec1 = (enc_sw) ? vSet_sec1 : vUpDown_sec1;
            v_sec2 = (enc_sw) ? vSet_sec2 : vUpDown_sec2;
            v_min1 = (enc_sw) ? vSet_min1 : vUpDown_min1;
            v_min2 = (enc_sw) ? vSet_min2 : vUpDown_min2;
            v_hr1 = (enc_sw) ? vSet_hr1 : vUpDown_hr1;
            v_hr2 = (enc_sw) ? vSet_hr2 : vUpDown_hr2;
            sound_en = 'b0;
            speaker = speaker_out;
        end
        downWait: begin
            load = 1'b1;
            ready = 1'b1;
            enable = 'b0;
            updn = 'b0;
            v_mil1 = (enc_sw) ? vSet_mil1 : vUpDown_mil1;
            v_mil2 = (enc_sw) ? vSet_mil2 : vUpDown_mil2;
            v_sec1 = (enc_sw) ? vSet_sec1 : vUpDown_sec1;
            v_sec2 = (enc_sw) ? vSet_sec2 : vUpDown_sec2;
            v_min1 = (enc_sw) ? vSet_min1 : vUpDown_min1;
            v_min2 = (enc_sw) ? vSet_min2 : vUpDown_min2;
            v_hr1 = (enc_sw) ? vSet_hr1 : vUpDown_hr1;
            v_hr2 = (enc_sw) ? vSet_hr2 : vUpDown_hr2;
            sound_en = 'b0;
            speaker = speaker_out;
        end
        up: begin
            load = 'b0;
            enable = 1'b1;
```

```
            ready = 'b0;
            updn = 1'b1;
            v_mil1 = (enc_sw) ? vSet_mil1 : vUpDown_mil1;
            v_mil2 = (enc_sw) ? vSet_mil2 : vUpDown_mil2;
            v_sec1 = (enc_sw) ? vSet_sec1 : vUpDown_sec1;
            v_sec2 = (enc_sw) ? vSet_sec2 : vUpDown_sec2;
            v_min1 = (enc_sw) ? vSet_min1 : vUpDown_min1;
            v_min2 = (enc_sw) ? vSet_min2 : vUpDown_min2;
            v_hr1 = (enc_sw) ? vSet_hr1 : vUpDown_hr1;
            v_hr2 = (enc_sw) ? vSet_hr2 : vUpDown_hr2;
            if (v_mil1 == 'b0 & v_mil2 == 'b0 & v_sec1 == 'b0 & v_sec2 == 'b0 & v_min1 ==
'b0 & v_min2 == 'b0 & v_hr1 == 'b0 & v_hr2 == 'b0) sound_en = 1'b1;
            else sound_en = 'b0;
            speaker = speaker_out;
          end
          down: begin
            load = 'b0;
            enable = 1'b1;
            ready = 'b0;
            updn = 'b0;
            v_mil1 = (enc_sw) ? vSet_mil1 : vUpDown_mil1;
            v_mil2 = (enc_sw) ? vSet_mil2 : vUpDown_mil2;
            v_sec1 = (enc_sw) ? vSet_sec1 : vUpDown_sec1;
            v_sec2 = (enc_sw) ? vSet_sec2 : vUpDown_sec2;
            v_min1 = (enc_sw) ? vSet_min1 : vUpDown_min1;
            v_min2 = (enc_sw) ? vSet_min2 : vUpDown_min2;
            v_hr1 = (enc_sw) ? vSet_hr1 : vUpDown_hr1;
            v_hr2 = (enc_sw) ? vSet_hr2 : vUpDown_hr2;
            if (v_mil1 == 'b0 & v_mil2 == 'b0 & v_sec1 == 'b0 & v_sec2 == 'b0 & v_min1 ==
'b0 & v_min2 == 'b0 & v_hr1 == 'b0 & v_hr2 == 'b0) sound_en = 1'b1;
            else sound_en = 'b0;
            speaker = speaker_out;
          end
          resetState: begin
            load = 'b0;
            ready = 'b0;
            enable = 'b0;
            updn = 'b0;
            v_mil1 = 4'b0;
            v_mil2 = 4'b0;
            v_sec1 = 4'b0;
            v_sec2 = 4'b0;
            v_min1 = 4'b0;
            v_min2 = 4'b0;
            v_hr1 = 4'b0;
```

```
            v_hr2 = 4'b0;
        end
        default: begin
        load = 'b0;
        ready = 'b0;
        enable = 'b0;
        updn = 'b0;
        v_mil1 = 4'b0;
        v_mil2 = 4'b0;
        v_sec1 = 4'b0;
        v_sec2 = 4'b0;
        v_min1 = 4'b0;
        v_min2 = 4'b0;
        v_hr1 = 4'b0;
        v_hr2 = 4'b0;
        end
    endcase

// Speaker
music sound(.clk(ten_khz_clk),.enable(sound_en),.speaker(speaker_out));

// Encoder
pmod_dial pmodDial(
    // Input Clock
    .clk(hun_hz_clk),
    //Control Inputs
    .enc_a(enc_a),
    .enc_b(enc_b),
    .enc_sw(enc_sw),
    .enc_btn(enc_btn),
    .enable(enable),
    .reset(reset),
    // Load Values
    .l_mil1(v_mil1),
    .l_mil2(v_mil2),
    .l_sec1(v_sec1),
    .l_sec2(v_sec2),
    .l_min1(v_min1),
    .l_min2(v_min2),
    .l_hr1(v_hr1),
    .l_hr2(v_hr2),
    // Output Values
    .v_mil1(vSet_mil1),
    .v_mil2(vSet_mil2),
    .v_sec1(vSet_sec1),
```

```
        .v_sec2(vSet_sec2),
        .v_min1(vSet_min1),
        .v_min2(vSet_min2),
        .v_hr1(vSet_hr1),
        .v_hr2(vSet_hr2)
        );

    // Counter
    up_down_counter upDownCount(
        // Input Clocks
        .clk (hun_hz_clk),
        // Control inputs
        .updn (updn),
        .load (load),
        .enable (enable),
        .reset (reset),
        // Load values
        .l_mil1 (v_mil1),
        .l_mil2 (v_mil2),
        .l_sec1 (v_sec1),
        .l_sec2 (v_sec2),
        .l_min1 (v_min1),
        .l_min2 (v_min2),
        .l_hr1 (v_hr1),
        .l_hr2 (v_hr2),
        // Output values
        .v_mil1 (vUpDown_mil1),
        .v_mil2 (vUpDown_mil2),
        .v_sec1 (vUpDown_sec1),
        .v_sec2 (vUpDown_sec2),
        .v_min1 (vUpDown_min1),
        .v_min2 (vUpDown_min2),
        .v_hr1 (vUpDown_hr1),
        .v_hr2 (vUpDown_hr2)
    );

    // Segment & Anode
    wire [7:0] seg_mil1, seg_mil2, seg_sec1, seg_sec2, seg_min1, seg_min2, seg_hr1,
seg_hr2;
    wire [7:0] an_mil1, an_mil2, an_sec1, an_sec2, an_min1, an_min2, an_hr1, an_hr2;

    BCD_Decoder mil1_bcd(.v(v_mil1),.anum(3'd0),.seg(seg_mil1),.an(an_mil1));
    BCD_Decoder mil2_bcd(.v(v_mil2),.anum(3'd1),.seg(seg_mil2),.an(an_mil2));
    BCD_Decoder sec1_bcd(.v(v_sec1),.anum(3'd2)..seg(seg_sec1),.an(an_sec1));
    BCD_Decoder sec2_bcd(.v(v_sec2),.anum(3'd3),.seg(seg_sec2),.an(an_sec2));
```

```
   BCD_Decoder min1_bcd(.v(v_min1),.anum(3'd4),.seg(seg_min1),.an(an_min1));
   BCD_Decoder min2_bcd(.v(v_min2),.anum(3'd5),.seg(seg_min2),.an(an_min2));
   BCD_Decoder hr1_bcd(.v(v_hr1), .anum(3'd6),.seg(seg_hr1),.an(an_hr1));
   BCD_Decoder hr2_bcd(.v(v_hr2),.anum(3'd7),.seg(seg_hr2),.an(an_hr2));

   always @(posedge refresh_clk) begin
     case(display)
       9: display = 0;
       8: begin an[7:0] <= an_hr2[7:0]; seg[7:0] <= seg_hr2[7:0]; end
       7: begin an[7:0] <= an_hr1[7:0]; seg[7:0] <= seg_hr1[7:0]; end
       6: begin an[7:0] <= an_min2[7:0]; seg[7:0] <= seg_min2[7:0]; end
       5: begin an[7:0] <= an_min1[7:0]; seg[7:0] <= seg_min1[7:0]; end
       4: begin an[7:0] <= an_sec2[7:0]; seg[7:0] <= seg_sec2[7:0]; end
       3: begin an[7:0] <= an_sec1[7:0]; seg[7:0] <= seg_sec1[7:0]; end
       2: begin an[7:0] <= an_mil2[7:0]; seg[7:0] <= seg_mil2[7:0]; end
       1: begin an[7:0] <= an_mil1[7:0]; seg[7:0] <= seg_mil1[7:0]; end
       endcase
       display = display + 1;
   end
endmodule
```

A.2 - Up/Down Counter

```
module up_down_counter(
    // Input Clocks
    input clk,
    // Control inputs
    input updn,
    input load,
    input enable,
    input reset,
    // Load values
    input [3:0] l_mil1,
    input [3:0] l_mil2,
    input [3:0] l_sec1,
    input [3:0] l_sec2,
    input [3:0] l_min1,
    input [3:0] l_min2,
    input [3:0] l_hr1,
    input [3:0] l_hr2,
    // Output values
    output reg [3:0] v_mil1,
    output reg [3:0] v_mil2,
    output reg [3:0] v_sec1,
    output reg [3:0] v_sec2,
    output reg [3:0] v_min1,
    output reg [3:0] v_min2,
    output reg [3:0] v_hr1,
    output reg [3:0] v_hr2
    );

    always @ (posedge clk) begin
        if (reset) begin
              v_mil1 <= 4'b0;
              v_mil2 <= 4'b0;
              v_sec1 <= 4'b0;
              v_sec2 <= 4'b0;
              v_min1 <= 4'b0;
              v_min2 <= 4'b0;
              v_hr1 <= 4'b0;
              v_hr2 <= 4'b0;
        end
        else begin
            if (load) begin
                v_mil1 <= l_mil1;
                v_mil2 <= l_mil2;
                v_sec1 <= l_sec1;
```

```
            v_sec2 <= l_sec2;
            v_min1 <= l_min1;
            v_min2 <= l_min2;
            v_hr1 <= l_hr1;
            v_hr2 <= l_hr2;
        end
        else begin
            if(updn) begin
                if (enable) begin
                    if(v_mil1 < 4'd9) begin
                        v_mil1 = v_mil1 + 4'b0001;
                    end
                    else if(v_mil1 == 4'd9 & v_mil2 < 4'd9) begin
                        v_mil1 <= 4'd0;
                        v_mil2 = v_mil2 + 4'b0001;
                    end
                    else if(v_mil1 == 4'd9 & v_mil2 == 4'd9 & v_sec1 < 4'd9) begin
                        v_mil1 <= 4'd0;
                        v_mil2 <= 4'd0;
                        v_sec1 = v_sec1 + 4'b0001;
                    end
                    else if(v_mil1 == 4'd9 & v_mil2 == 4'd9 & v_sec1 == 4'd9 & v_sec2 < 4'd5)
begin
                        v_mil1 <= 4'd0;
                        v_mil2 <= 4'd0;
                        v_sec1 <= 4'd0;
                        v_sec2 <= v_sec2 + 4'b0001;
                    end
                    else if(v_mil1 == 4'd9 & v_mil2 == 4'd9 & v_sec1 == 4'd9 & v_sec2 == 4'd5
& v_min1 < 4'd9) begin
                        v_mil1 <= 4'd0;
                        v_mil2 <= 4'd0;
                        v_sec1 <= 4'd0;
                        v_sec2 <= 4'd0;
                        v_min1 <= v_min1 + 4'b0001;
                    end
                    else if(v_mil1 == 4'd9 & v_mil2 == 4'd9 & v_sec1 == 4'd9 & v_sec2 == 4'd5
& v_min1 == 4'd9 & v_min2 < 4'd5) begin
                        v_mil1 <= 4'd0;
                        v_mil2 <= 4'd0;
                        v_sec1 <= 4'd0;
                        v_sec2 <= 4'd0;
                        v_min1 <= 4'd0;
                        v_min2 <= v_min2 + 4'b0001;
                    end
```

```
                else if(v_mil1 == 4'd9 & v_mil2 == 4'd9 & v_sec1 == 4'd9 & v_sec2 == 4'd5
& v_min1 == 4'd9 & v_min2 == 4'd5 & v_hr1 < 4'd9) begin
                    v_mil1 <= 4'd0;
                    v_mil2 <= 4'd0;
                    v_sec1 <= 4'd0;
                    v_sec2 <= 4'd0;
                    v_min1 <= 4'd0;
                    v_min2 <= 4'd0;
                    v_hr1 <= v_hr1 + 4'b0001;
                end
                else if(v_mil1 == 4'd9 & v_mil2 == 4'd9 & v_sec1 == 4'd9 & v_sec2 == 4'd5
& v_min1 == 4'd9 & v_min2 == 4'd5 & v_hr1 == 4'd9 & v_hr2 < 4'd5) begin
                    v_mil1 <= 4'd0;
                    v_mil2 <= 4'd0;
                    v_sec1 <= 4'd0;
                    v_sec2 <= 4'd0;
                    v_min1 <= 4'd0;
                    v_min2 <= 4'd0;
                    v_hr1 <= 4'd0;
                    v_hr2 <= v_hr2 + 4'b0001;
                end
            end
        end
        else begin
            if (enable) begin
                if(v_mil1 > 4'b0) begin
                    v_mil1 = v_mil1 - 4'b0001;
                end
                else if(v_mil1 == 4'd0 & v_mil2 > 4'd0) begin
                    v_mil1 <= 4'd9;
                    v_mil2 = v_mil2 - 4'b0001;
                end
                else if(v_mil1 == 4'd0 & v_mil2 == 4'd0 & v_sec1 > 4'd0) begin
                    v_mil1 <= 4'd9;
                    v_mil2 <= 4'd9;
                    v_sec1 = v_sec1 - 4'b0001;
                end
                else if(v_mil1 == 4'd0 & v_mil2 == 4'd0 & v_sec1 == 4'd0 & v_sec2 > 4'd0)
begin
                    v_mil1 <= 4'd9;
                    v_mil2 <= 4'd9;
                    v_sec1 <= 4'd9;
                    v_sec2 <= v_sec2 - 4'b0001;
                end
                else if(v_mil1 == 4'd0 & v_mil2 == 4'd0 & v_sec1 == 4'd0 & v_sec2 == 4'd0
```

```
& v_min1 > 4'd0) begin
                    v_mil1 <= 4'd9;
                    v_mil2 <= 4'd9;
                    v_sec1 <= 4'd9;
                    v_sec2 <= 4'd5;
                    v_min1 <= v_min1 - 4'b0001;
                end
                else if(v_mil1 == 4'd0 & v_mil2 == 4'd0 & v_sec1 == 4'd0 & v_sec2 == 4'd0
& v_min1 == 4'd0 & v_min2 > 4'd0) begin
                    v_mil1 <= 4'd9;
                    v_mil2 <= 4'd9;
                    v_sec1 <= 4'd9;
                    v_sec2 <= 4'd5;
                    v_min1 <= 4'd9;
                    v_min2 <= v_min2 - 4'b0001;
                end
                else if(v_mil1 == 4'd0 & v_mil2 == 4'd0 & v_sec1 == 4'd0 & v_sec2 == 4'd0
& v_min1 == 4'd0 & v_min2 == 4'd0 & v_hr1 > 4'd0) begin
                    v_mil1 <= 4'd9;
                    v_mil2 <= 4'd9;
                    v_sec1 <= 4'd9;
                    v_sec2 <= 4'd5;
                    v_min1 <= 4'd9;
                    v_min2 <= 4'd5;
                    v_hr1 <= v_hr1 - 4'b0001;
                end
                else if(v_mil1 == 4'd0 & v_mil2 == 4'd0 & v_sec1 == 4'd0 & v_sec2 == 4'd0
& v_min1 == 4'd0 & v_min2 == 4'd0 & v_hr1 == 4'd0 & v_hr2 > 4'd0) begin
                    v_mil1 <= 4'd9;
                    v_mil2 <= 4'd9;
                    v_sec1 <= 4'd9;
                    v_sec2 <= 4'd5;
                    v_min1 <= 4'd9;
                    v_min2 <= 4'd5;
                    v_hr1 <= 4'd9;
                    v_hr2 <= v_hr2 - 4'b0001;
                end
            end
        end
    end
  end
endmodule
```

A.3 - Clock Divider

```
module clk_divider(
    input clk,
    input reset,
    output reg hun_hz_clk,
    output reg ten_khz_clk,
    output reg refresh_clk
    );
    wire ten_mhz_clk;
    integer hun_hz_count, ten_khz_count, refresh_count;

    initial begin
        hun_hz_count = 0;
        ten_khz_count = 0;
        refresh_count = 0;
    end


    ten_mhz_clk clkwiz(
          // Clock out ports
          .clk_out1 (ten_mhz_clk),
          // Status and control signals
          //.reset,
          //.locked,
         // Clock in ports
          .clk_in1  (clk)
     );
    always @ (posedge ten_mhz_clk) begin
        if(reset) begin
            hun_hz_count = 0;
            ten_khz_count = 0;
            refresh_count = 0;
        end
        else begin
            if(hun_hz_count == 50000) begin
                hun_hz_count = 0;
                hun_hz_clk = !hun_hz_clk;
            end
            if(ten_khz_count == 500) begin
                ten_khz_count = 0;
                ten_khz_clk = !ten_khz_clk;
            end
            if(refresh_count == 12000) begin
                refresh_count = 0;
                refresh_clk = !refresh_clk;
```

```
        end
        refresh_count = refresh_count + 1;
        ten_khz_count = ten_khz_count + 1;
        hun_hz_count = hun_hz_count + 1;
      end
    end

endmodule
```

A.4 - BCD Converter

```
module BCD_Decoder(
    input [3:0]v,
          input [2:0] anum,
    output [7:0]seg,
    output [7:0]an
    );
    wire [2:0]circAin;
    wire [3:0]m;
    wire z;
    Comparator comparator(
        .A      (v[0]),
        .B      (v[1]),
        .C      (v[2]),
        .D      (v[3]),
        .z      (z)
    );

    Circuit_A circuitA(
        .A      (v[0]),
        .B      (v[1]),
        .C      (v[2]),
        .m      (circAin)
    );


    twoToOneMUX mux0(
        .y      (v[0]),
        .x      (circAin[0]),
        .s      (z),
        .m      (m[0])
    );

    twoToOneMUX mux1(
        .y      (v[1]),
        .x      (circAin[1]),
        .s      (z),
        .m      (m[1])
    );
    twoToOneMUX mux2(
        .y      (v[2]),
        .x      (circAin[2]),
        .s      (z),
        .m      (m[2])
```

```
    );
    twoToOneMUX mux3(
        .y      (v[3]),
        .x      (1'b0),
        .s      (z),
        .m      (m[3])
    );
    segment_anode_set seg_an(
                    .anum  (anum),
        .bcd    (m),
        .seg    (seg),
        .an     (an)
    );
endmodule

module Comparator(
    input A,
    input B,
    input C,
    input D,
    output z
    );
    assign z = D & ((!A & B & !C) | (A & B & !C) | (!A & !B &C) | (A & !B & C) | (!A & B & C) |
(A & B & C));
endmodule

module Circuit_A(
    input A,
    input B,
    input C,
    output [2:0]m
    );
    assign m[0] = (!C & B & A) | (C & !B & A) | (C & B & A);
    assign m[1] = (C & !B & !A) | (C & !B & A);
    assign m[2] = (C & B & !A) | (C & B & A);
endmodule

module twoToOneMUX(
    input x, y,
    input s,
    output reg m
    );
    always @(x or y or s)
        if(s == 1)
            m = x;
```

```
      else
        m = y;
endmodule

module segment_anode_set(
   input [3:0]bcd,
         input [2:0] anum,
   output reg [7:0]seg,
   output reg [7:0]an
   );
   always @(bcd or anum) begin
                  case(anum)
                        0: an = 8'b11111110;
                        1: an = 8'b11111101;
                        2: an = 8'b11111011;
                        3: an = 8'b11110111;
                        4: an = 8'b11101111;
                        5: an = 8'b11011111;
                        6: an = 8'b10111111;
                        7: an = 8'b01111111;
                        default: an = 8'b11111111;
                  endcase
                  case (bcd)
                        0 : seg = 8'b11000000;
                        1 : seg = 8'b11111001;
                        2 : seg = 8'b10100100;
                        3 : seg = 8'b10110000;
                        4 : seg = 8'b10011001;
                        5 : seg = 8'b10010010;
                        6 : seg = 8'b10000010;
                        7 : seg = 8'b11111000;
                        8 : seg = 8'b10000000;
                        9 : seg = 8'b10010000;
                        default : seg = 8'b10000000;
                  endcase
         end
endmodule
```

A.5 - Sound

```
module music(
    input clk,
    input enable,
    output reg speaker
    );
    reg [2:0] count;
    always @(posedge clk)
        if (enable) begin
            if (count == 3'd5) begin
                speaker <= ~speaker;
                count <= 0;
            end
            else count <= count+1;
        end
endmodule
```

A.6 - PMOD

```
module pmod_dial(
    // Input Clock
    input clk,
    // Control values
    input enc_a,
    input enc_b,
    input enc_sw,
    input enc_btn,
    input enable,
    input reset,
    // Load Values
    input [3:0] l_mil1,
    input [3:0] l_mil2,
    input [3:0] l_sec1,
    input [3:0] l_sec2,
    input [3:0] l_min1,
    input [2:0] l_min2,
    input [3:0] l_hr1,
    input [3:0] l_hr2,
    // Output Values
    output reg [3:0] v_mil1,
    output reg [3:0] v_mil2,
    output reg [3:0] v_sec1,
    output reg [3:0] v_sec2,
    output reg [3:0] v_min1,
    output reg [2:0] v_min2,
    output reg [3:0] v_hr1,
    output reg [3:0] v_hr2
    );

    wire [7:0] switch_db;
    wire [4:0] btn_db;
    wire enc_a_db;
    wire enc_b_db;
    wire enc_sw_db;
    wire enc_btn_db;

    wire [7:0] switch_rise;
    wire [4:0] btn_rise;
    wire enc_a_rise;
    wire enc_b_rise;
    wire enc_sw_rise;
    wire enc_btn_rise;
```

```
    wire [7:0] switch_fall;
    wire [4:0] btn_fall;
    wire enc_a_fall;
    wire enc_b_fall;
    wire enc_sw_fall;
    wire enc_btn_fall;

    debounce
    #(
        .width(4),
        .bounce_limit(50000)
    )
    debounce
    (
        .clk(clk),
        .switch_in({enc_a,enc_b,enc_sw,enc_btn,btn,switch}),
        .switch_out({enc_a_db,enc_b_db,enc_sw_db,enc_btn_db,btn_db,switch_db}),
        .switch_rise({enc_a_rise,enc_b_rise,enc_sw_rise,enc_btn_rise,btn_rise,switch_rise}),
        .switch_fall({enc_a_fall,enc_b_fall,enc_sw_fall,enc_btn_fall,btn_fall,switch_fall})
    );

    reg [2:0] select;
    reg [2:0] state, nextstate;
    parameter break=0, seg1=1, seg2=2, seg3=3, seg4=4, seg5=5, seg6=6, seg7=7,
seg8=8;

    always @(posedge enc_btn) state <= nextstate;

    always @(*)
        case(state)
            break: nextstate = seg1;
            seg1: nextstate = seg2;
            seg2: nextstate = seg3;
            seg3: nextstate = seg4;
            seg4: nextstate = seg5;
            seg5: nextstate = seg6;
            seg6: nextstate = seg7;
            seg7: nextstate = seg8;
            seg8: nextstate = break;
        endcase

    always @(state)
        case(state)
            break: select = 0;
```

```
        seg1: select = 1;
        seg2: select = 2;
        seg3: select = 3;
        seg4: select = 4;
        seg5: select = 5;
        seg6: select = 6;
        seg7: select = 7;
        seg8: select = 8;
    endcase

always @(posedge clk or posedge reset) begin
    if (reset) begin
            v_mil1 <= 4'b0;
            v_mil2 <= 4'b0;
            v_sec1 <= 4'b0;
            v_sec2 <= 4'b0;
            v_min1 <= 4'b0;
            v_min2 <= 4'b0;
            v_hr1 <= 4'b0;
            v_hr2 <= 4'b0;
    end
    else if (!enable) begin
        if (enc_a_rise) begin
            if (!enc_b_db) begin
                if (select == 1) begin
                    if (l_mil1 > 0) v_mil1 <= l_mil1-1;
                    else v_mil1 <= l_mil1;
                    v_mil2 <= l_mil2;
                    v_sec1 <= l_sec1;
                    v_sec2 <= l_sec2;
                    v_min1 <= l_min1;
                    v_min2 <= l_min2;
                    v_hr1 <= l_hr1;
                    v_hr2 <= l_hr2;
                end
                else if (select == 2) begin
                    v_mil1 <= l_mil1;
                    if (l_mil2 > 0) v_mil2 <= l_mil2-1;
                    else v_mil2 <= l_mil2;
                    v_sec1 <= l_sec1;
                    v_sec2 <= l_sec2;
                    v_min1 <= l_min1;
                    v_min2 <= l_min2;
                    v_hr1 <= l_hr1;
                    v_hr2 <= l_hr2;
```

```
        end
        else if (select == 3) begin
            v_mil1 <= l_mil1;
            v_mil2 <= l_mil2;
            if (l_sec1 > 0) v_sec1 <= l_sec1-1;
            else v_sec1 <= l_sec1;
            v_sec2 <= l_sec2;
            v_min1 <= l_min1;
            v_min2 <= l_min2;
            v_hr1 <= l_hr1;
            v_hr2 <= l_hr2;
        end
        else if (select == 4) begin
            v_mil1 <= l_mil1;
            v_mil2 <= l_mil2;
            v_sec1 <= l_sec1;
            if (l_sec2 > 0) v_sec2 <= l_sec2-1;
            else v_sec2 <= l_sec2;
            v_min1 <= l_min1;
            v_min2 <= l_min2;
            v_hr1 <= l_hr1;
            v_hr2 <= l_hr2;
        end
        else if (select == 5) begin
            v_mil1 <= l_mil1;
            v_mil2 <= l_mil2;
            v_sec1 <= l_sec1;
            v_sec2 <= l_sec2;
            if (l_min1 > 0) v_min1 <= l_min1-1;
            else v_min1 <= l_min1;
            v_min2 <= l_min2;
            v_hr1 <= l_hr1;
            v_hr2 <= l_hr2;
        end
        else if (select == 6) begin
            v_mil1 <= l_mil1;
            v_mil2 <= l_mil2;
            v_sec1 <= l_sec1;
            v_sec2 <= l_sec2;
            v_min1 <= l_min1;
            if (l_min2 > 0) v_min2 <= l_min2-1;
            else v_min2 <= l_min2;
            v_hr1 <= l_hr1;
            v_hr2 <= l_hr2;
        end
```

```
            else if (select == 7) begin
               v_mil1 <= l_mil1;
               v_mil2 <= l_mil2;
               v_sec1 <= l_sec1;
               v_sec2 <= l_sec2;
               v_min1 <= l_min1;
               v_min2 <= l_min2;
               if (l_hr1 > 0) v_hr1 <= l_hr1-1;
               else v_hr1 <= l_hr1;
               v_hr2 <= l_hr2;
            end
            else if (select == 8) begin
               v_mil1 <= l_mil1;
               v_mil2 <= l_mil2;
               v_sec1 <= l_sec1;
               v_sec2 <= l_sec2;
               v_min1 <= l_min1;
               v_min2 <= l_min2;
               v_hr1 <= l_hr1;
               if (l_hr2 > 0) v_hr2 <= l_hr2-1;
               else v_hr2 <= l_hr2;
            end
         end
      end
      else if (enc_b_rise) begin
         if (!enc_a_db) begin
            if (select == 1) begin
               if (l_mil1 < 9) v_mil1 <= l_mil1+1;
               else v_mil1 <= l_mil1;
               v_mil2 <= l_mil2;
               v_sec1 <= l_sec1;
               v_sec2 <= l_sec2;
               v_min1 <= l_min1;
               v_min2 <= l_min2;
               v_hr1 <= l_hr1;
               v_hr2 <= l_hr2;
            end
            else if (select == 2) begin
               v_mil1 <= l_mil1;
               if (l_mil2 < 9) v_mil2 <= l_mil2+1;
               else v_mil2 <= l_mil2;
               v_sec1 <= l_sec1;
               v_sec2 <= l_sec2;
               v_min1 <= l_min1;
               v_min2 <= l_min2;
```

```
        v_hr1 <= l_hr1;
        v_hr2 <= l_hr2;
    end
    else if (select == 3) begin
        v_mil1 <= l_mil1;
        v_mil2 <= l_mil2;
        if (l_sec1 < 9) v_sec1 <= l_sec1+1;
        else v_sec1 <= l_sec1;
        v_sec2 <= l_sec2;
        v_min1 <= l_min1;
        v_min2 <= l_min2;
        v_hr1 <= l_hr1;
        v_hr2 <= l_hr2;
    end
    else if (select == 4) begin
        v_mil1 <= l_mil1;
        v_mil2 <= l_mil2;
        v_sec1 <= l_sec1;
        if (l_sec2 < 9) v_sec2 <= l_sec2+1;
        else v_sec2 <= l_sec2;
        v_min1 <= l_min1;
        v_min2 <= l_min2;
        v_hr1 <= l_hr1;
        v_hr2 <= l_hr2;
    end
    else if (select == 5) begin
        v_mil1 <= l_mil1;
        v_mil2 <= l_mil2;
        v_sec1 <= l_sec1;
        v_sec2 <= l_sec2;
        if (l_min1 < 9) v_min1 <= l_min1+1;
        else v_min1 <= l_min1;
        v_min2 <= l_min2;
        v_hr1 <= l_hr1;
        v_hr2 <= l_hr2;
    end
    else if (select == 6) begin
        v_mil1 <= l_mil1;
        v_mil2 <= l_mil2;
        v_sec1 <= l_sec1;
        v_sec2 <= l_sec2;
        v_min1 <= l_min1;
        if (l_min2 < 5) v_min2 <= l_min2+1;
        else v_min2 <= l_min2;
        v_hr1 <= l_hr1;
```

```
                v_hr2 <= l_hr2;
            end
            else if (select == 7) begin
                v_mil1 <= l_mil1;
                v_mil2 <= l_mil2;
                v_sec1 <= l_sec1;
                v_sec2 <= l_sec2;
                v_min1 <= l_min1;
                v_min2 <= l_min2;
                if (l_hr1 < 9) v_hr1 <= l_hr1+1;
                else v_hr1 <= l_hr1;
                v_hr2 <= l_hr2;
            end
            else if (select == 8) begin
                v_mil1 <= l_mil1;
                v_mil2 <= l_mil2;
                v_sec1 <= l_sec1;
                v_sec2 <= l_sec2;
                v_min1 <= l_min1;
                v_min2 <= l_min2;
                v_hr1 <= l_hr1;
                if (l_hr2 < 9) v_hr2 <= l_hr2+1;
                else v_hr2 <= l_hr2;
            end
          end
        end
      end
      else begin
        v_mil1 <= l_mil1;
        v_mil2 <= l_mil2;
        v_sec1 <= l_sec1;
        v_sec2 <= l_sec2;
        v_min1 <= l_min1;
        v_min2 <= l_min2;
        v_hr1 <= l_hr1;
        v_hr2 <= l_hr2;
      end
    end
endmodule
```

A.7 - Debounce for PMOD

```
module debounce
 #(
  parameter width = 1,
  parameter bounce_limit = 1024
  )
 (
  input clk,
  input [width-1:0] switch_in,
  output reg [width-1:0] switch_out,
  output reg [width-1:0] switch_rise,
  output reg [width-1:0] switch_fall
  );
 genvar  i;
 generate
  for (i=0; i<width;i=i+1)
      begin
         reg [$clog2(bounce_limit)-1:0] bounce_count = 0;
         reg switch_latched = 0;
         reg switch_latched_state = 0;

               reg [1:0] switch_shift = 0;
               always @(posedge clk)
                 switch_shift <= {switch_shift,switch_in[i]};

         always @(posedge clk)
                 if (bounce_count == 0)
                    begin
                  switch_rise[i] <= switch_shift == 2'b01;
                  switch_fall[i] <= switch_shift == 2'b10;
                  switch_out[i] <= switch_shift[0];
                  if (switch_shift[1] != switch_shift[0])
                     bounce_count <= bounce_limit-1;
                    end
            else
            begin
               switch_rise[i] <= 0;
               switch_fall[i] <= 0;
               bounce_count <= bounce_count-1;
            end
         end
   endgenerate
endmodule
```

Appendix B - Schematic