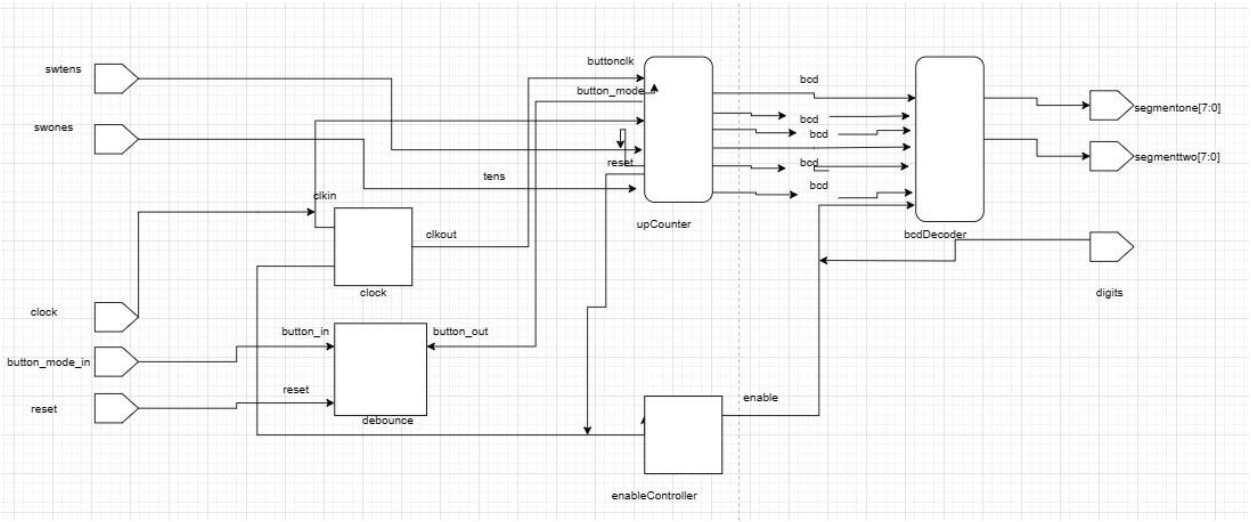
Digital Logic Design

Student ID: F11115114 **Name:** Alvaro Jara

1. **Topic**: Finite State Machine (FSM)

2. **Design structure:**



3. **Code**:

```
module clock_24h_set(clk, B, rst, Y, Z, scan, dec, unit);
    input clk, rst, B;
    input [3:0] dec, unit;
    output [7:0] Y, Z;
    output reg [5:0] scan;
    wire clk_fast_out, clk_1hz_out;
    wire [3:0] bcd1, bcd2, bcd3, bcd4, bcd5, bcd6;
    integer x, a, b;
    reg [3:0] bcd_out;
    wire ctrl_button;

    d_bounce(clk, rst, B, ctrl_button);
    clk_fast(clk, rst, clk_fast_out);
    clk_divider_1hz(clk,rst,clk_1hz_out);

    integer state;
    integer idle = 0;
    integer sec = 1;
    integer min = 2;
    integer hr = 3;
```

```
    always@(posedge clk_1hz_out or posedge rst)begin
        if(rst) state<=idle;
        else begin
            case(state)
                idle:begin
                    if(ctrl_button)state<=sec;
                    else state<=idle;
                end
                sec:begin
                    if(ctrl_button)state<=min;
                    else state<=sec;
                end
                min:begin
                    if(ctrl_button)state<=hr;
                    else state<=min;
                end
                hr:begin
                    if(ctrl_button)state<=idle;
                    else state<=hr;
                end
            endcase
        end
    end

    counters(clk_1hz_out, rst, bcd1, bcd2, bcd3, bcd4, bcd5, bcd6, state,
dec, unit);
    always @(posedge clk_fast_out or posedge rst)begin
        if(rst)
            x<=0;
        else if (x==5) begin
            x<=0;
        end
        else
            x<=x+1;
    end
    always @(posedge clk_fast_out)begin
        case(x)
            0:begin
                bcd_out<=bcd1;
                scan<=6'b000001;
            end
            1:begin
                bcd_out<=bcd2;
```

```
                scan<=6'b000010;
            end
            2:begin
                bcd_out<=bcd3;
                scan<=6'b000100;
            end
            3:begin
                bcd_out<=bcd4;
                scan<=6'b001000;
            end
            4:begin
                bcd_out<=bcd5;
                scan<=6'b010000;
            end
            5:begin
                bcd_out<=bcd6;
                scan<=6'b100000;
            end
        endcase
    end
    bcd_to_7seg1(bcd_out, Y);
    bcd_to_7seg2(bcd_out, Z);
endmodule
```

```
module clk_divider_1hz(clk,rst, clk_out);
    input clk, rst;
    output reg clk_out;
    integer cnt;
    always @(posedge clk or posedge rst) begin
        if (rst)
            cnt<=0;
        else if (cnt==100000000)
            cnt<=0;
        else
            cnt<=cnt+1;
    end
    always @(posedge clk or posedge rst) begin
        if (rst)
            clk_out<=0;
        else if (cnt<=50000000)
            clk_out<=1'b1;
```

```
        else
            clk_out<=1'b0;
    end
endmodule

module clk_btn(clk, rst, clk_out);
    input clk, rst;
    output reg clk_out;
    integer cnt;
    always @(posedge clk or posedge rst) begin
        if (rst)
            cnt<=0;
        else if (cnt==1000000)
            cnt<=0;
        else
            cnt<=cnt+1;
    end
    always @(posedge clk or posedge rst) begin
        if (rst)
            clk_out<=0;
        else if (cnt<=500000)
            clk_out<=1'b1;
        else
            clk_out<=1'b0;
    end
endmodule

module clk_fast(clk, rst, clk_out);
    input clk, rst;
    output reg clk_out;
    integer cnt;
    always@(posedge clk or posedge rst)
    begin
        if(rst) cnt <=0;
        else if(cnt==100000)
        cnt<=0;
        else
        cnt<=cnt+1;
      end
    always @ (posedge(clk) or posedge rst)
    begin
        if (rst) clk_out <= 0;
        else if(cnt<=50000)clk_out<=1'b1;
```

```
        else clk_out<=1'b0;
    end
endmodule
```

```
module clk_slow(clk, rst, clk_out);
    input clk, rst;
    output reg clk_out;
    integer cnt;
    always@(posedge clk or posedge rst)
    begin
        if(rst) cnt <=0;
        else if(cnt==1000000)
        cnt<=0;
        else
        cnt<=cnt+1;
      end
    always @ (posedge(clk) or posedge rst)
    begin
        if (rst) clk_out <= 0;
        else if(cnt<=500000)clk_out<=1'b1;
        else clk_out<=1'b0;
    end
endmodule
module and_gate(x, y, f);
    input x, y;
    output f;
    assign f = x & y;
endmodule
module dff(clk, rst, D, Q);
    input clk, rst, D;
    output reg Q;
    always @(posedge (clk) or posedge rst) begin
        if (rst) Q <= 1'b0;
        else Q <= D;
    end
endmodule
module d_bounce(clk, rst, D, d_bnc);
    input clk, rst, D;
    output d_bnc;
    wire [2:0] Q;
    wire clk_div, 1;
```

```
    clk_fast u1(clk, rst, clk_div);
    dff d1(clk_div, rst, D, Q[0]);
    dff d2(clk_div, rst, Q[0], Q[1]);
    dff d3(clk_div, rst, Q[1], Q[2]);
    and_gate and1(Q[0], Q[1], l);
    and_gate and2(Q[2], l, d_bnc);
endmodule

module counters(clk, rst, bcd1, bcd2, bcd3, bcd4, bcd5, bcd6, state, dec,
unit);
    input clk, rst;
    input [3:0]dec, unit;
    input [31:0] state;
    output reg [3:0] bcd1, bcd2, bcd3, bcd4, bcd5, bcd6;
    integer cnt1, cnt2, cnt3, cnt4, cnt5, cnt6;
    integer a, b;

    always@(posedge clk or posedge rst)begin
        if(rst)begin
            a<=0;
            b<=0;
        end
        else begin
            case(unit)
                4'b0001: a = 1;
                4'b0010: a = 2;
                4'b0011: a = 3;
                4'b0100: a = 4;
                4'b0101: a = 5;
                4'b0110: a = 6;
                4'b0111: a = 7;
                4'b1000: a = 8;
                4'b1001: a = 9;
                4'b1010: a = 9;
                4'b1011: a = 9;
                4'b1100: a = 9;
                4'b1101: a = 9;
                4'b1110: a = 9;
                4'b1111: a = 9;
                default: a = 0;
            endcase
            case(dec)
                4'b0001: b = 1;
```

```
            4'b0010: b = 2;
            4'b0011: b = 3;
            4'b0100: b = 4;
            4'b0101: b = 5;
            4'b0110: b = 6;
            4'b0111: b = 7;
            4'b1000: b = 8;
            4'b1001: b = 9;
            4'b1010: b = 9;
            4'b1011: b = 9;
            4'b1100: b = 9;
            4'b1101: b = 9;
            4'b1110: b = 9;
            4'b1111: b = 9;
            default: b = 0;
        endcase
    end
end
always@(posedge clk or posedge rst) begin //cnt1
   if(rst) begin
          cnt1<=0;
          cnt2<=0;
          cnt3<=0;
          cnt4<=0;
          cnt5<=0;
          cnt6<=0;
       end
   else
   case(state)
        0:begin
           cnt1<=cnt1+1;
           if(cnt1>=9)begin
                cnt1<=0;
                cnt2<=cnt2+1;
                if(cnt2>=5)begin
                    cnt2<=0;
                    cnt3<=cnt3+1;
                    if(cnt3>=9)begin
                        cnt3<=0;
                        cnt4<=cnt4+1;
                        if(cnt4>=5)begin
                            cnt4<=0;
                            cnt5<=cnt5+1;
```

```
                        if(cnt5>=9)begin
                            cnt5<=0;
                            cnt6<=cnt6+1;
                        end
                        else if(cnt5>=3 && cnt6>=2)begin
                            cnt5<=0;
                            cnt6<=0;
                        end
                    end
                end
            end
        end
    end
    1:begin
         cnt1<=a;
         cnt2<=b;
    if(a > 9)begin
          cnt1<=9;
    end
    if(b > 5)begin
        cnt1<=9;
         cnt2<=5;
    end
    end
    2:begin
         cnt3<=a;
         cnt4<=b;
    if(a > 9)begin
          cnt3<=9;
    end
    if(b > 5)begin
        cnt3 <=9;
         cnt4<=5;
    end
    end
    3:begin
         cnt5<=a;
         cnt6<=b;
    if(b < 2 & a > 9)begin
          cnt5<=9;
    end
    else if(b >= 2 & a >= 3)begin
         cnt5<=3;
```

```
            end
            if(b > 2)begin
                cnt5<=3;
                 cnt6<=2;
            end
            end
        endcase
    end
    always @(cnt1) begin
         case(cnt1)
          1      : bcd1 = {4'b0001};
          2      : bcd1 = {4'b0010};
          3      : bcd1 = {4'b0011};
          4      : bcd1 = {4'b0100};
          5      : bcd1 = {4'b0101};
          6      : bcd1 = {4'b0110};
          7      : bcd1 = {4'b0111};
          8      : bcd1 = {4'b1000};
          9      : bcd1 = {4'b1001};
          default: bcd1 = {4'b0000};
       endcase
    end
    always @(cnt2) begin
         case(cnt2)
          1      : bcd2 = {32'b0001};
          2      : bcd2 = {32'b0010};
          3      : bcd2 = {32'b0011};
          4      : bcd2 = {32'b0100};
          5      : bcd2 = {32'b0101};
          default: bcd2 = {32'b0000};
       endcase
    end
    always @(cnt3) begin
         case(cnt3)
          1      : bcd3 = {4'b0001};
          2      : bcd3 = {4'b0010};
          3      : bcd3 = {4'b0011};
          4      : bcd3 = {4'b0100};
          5      : bcd3 = {4'b0101};
          6      : bcd3 = {4'b0110};
          7      : bcd3 = {4'b0111};
          8      : bcd3 = {4'b1000};
          9      : bcd3 = {4'b1001};
```

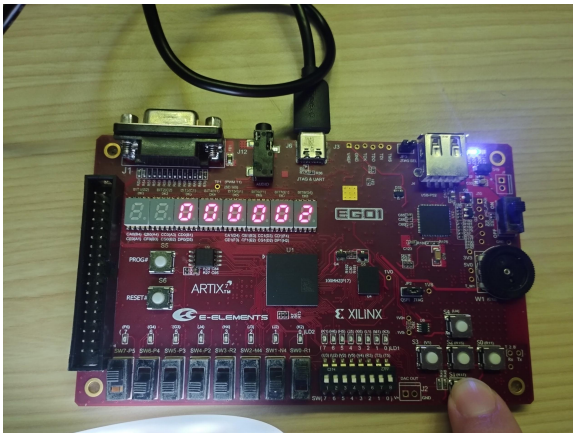
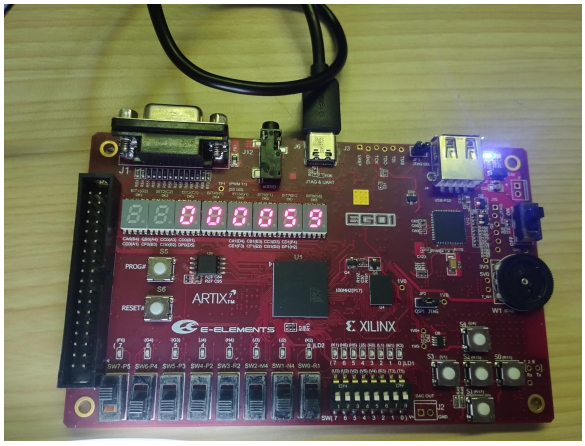
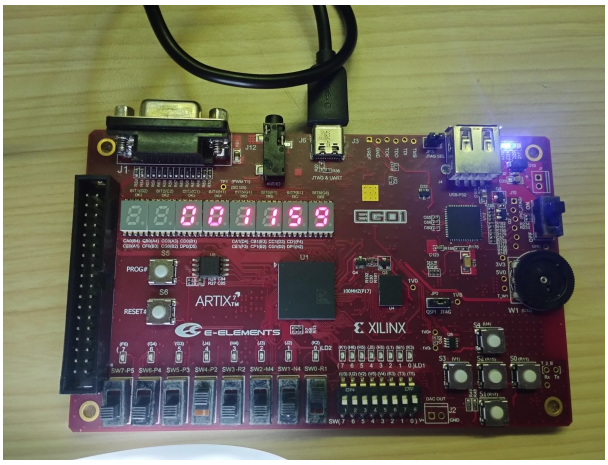
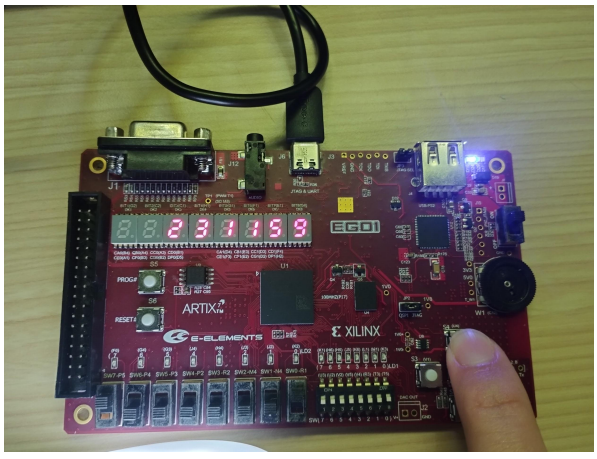
```
          default: bcd3 = {4'b0000};
       endcase
    end
    always @(cnt4) begin
         case(cnt4)
          1      : bcd4 = {4'b0001};
          2      : bcd4 = {4'b0010};
          3      : bcd4 = {4'b0011};
          4      : bcd4 = {4'b0100};
          5      : bcd4 = {4'b0101};
          default: bcd4 = {4'b0000};
       endcase
    end
    always @(cnt5) begin
         case(cnt5)
          1      : bcd5 = {4'b0001};
          2      : bcd5 = {4'b0010};
          3      : bcd5 = {4'b0011};
          4      : bcd5 = {4'b0100};
          5      : bcd5 = {4'b0101};
          6      : bcd5 = {4'b0110};
          7      : bcd5 = {4'b0111};
          8      : bcd5 = {4'b1000};
          9      : bcd5 = {4'b1001};
          default: bcd5 = {4'b0000};
       endcase
    end
    always @(cnt6) begin
         case(cnt6)
          1      : bcd6 = {4'b0001};
          2      : bcd6 = {4'b0010};
          default: bcd6 = {4'b0000};
       endcase
    end
endmodule
module bcd_to_7seg1(bcd_in, Y);
    input [3:0] bcd_in;
    output reg [7:0] Y;
    always @(bcd_in) begin
        case({bcd_in})
           4'b0001 : Y = {8'b01100000};
           4'b0010 : Y = {8'b11011010};
           4'b0011 : Y = {8'b11110010};
```

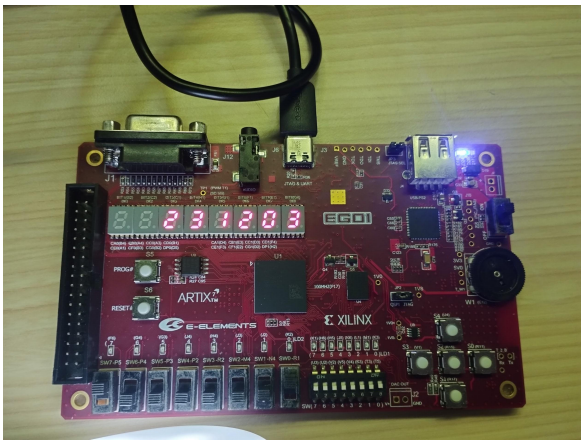
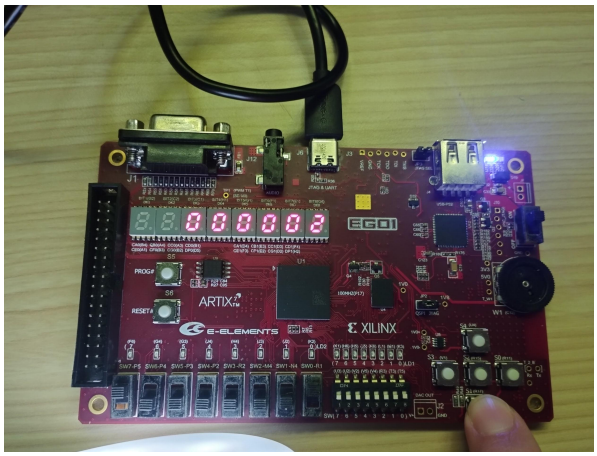
```
            4'b0100 : Y = {8'b01100110};
            4'b0101 : Y = {8'b10110110};
            4'b0110 : Y = {8'b10111110};
            4'b0111 : Y = {8'b11100000};
            4'b1000 : Y = {8'b11111110};
            4'b1001 : Y = {8'b11110110};
            4'b1010 : Y = {8'b11101110};
            4'b1011 : Y = {8'b00111110};
            4'b1100 : Y = {8'b10011100};
            4'b1101 : Y = {8'b01111010};
            4'b1110 : Y = {8'b10011110};
            4'b1111 : Y = {8'b10001110};
            default: Y = {8'b11111100};
        endcase
    end
endmodule

module bcd_to_7seg2(bcd_in, Z);
    input [3:0] bcd_in;
    output reg [7:0] Z;
    always @(bcd_in) begin
        case({bcd_in})
            4'b0001 : Z = {8'b01100000};
            4'b0010 : Z = {8'b11011010};
            4'b0011 : Z = {8'b11110010};
            4'b0100 : Z = {8'b01100110};
            4'b0101 : Z = {8'b10110110};
            4'b0110 : Z = {8'b10111110};
            4'b0111 : Z = {8'b11100000};
            4'b1000 : Z = {8'b11111110};
            4'b1001 : Z = {8'b11110110};
            4'b1010 : Z = {8'b11101110};
            4'b1011 : Z = {8'b00111110};
            4'b1100 : Z = {8'b10011100};
            4'b1101 : Z = {8'b01111010};
            4'b1110 : Z = {8'b10011110};
            4'b1111 : Z = {8'b10001110};
            default: Z = {8'b11111100};
        endcase
    end
endmodule
```

4. **Result**:

Digital Logic Design

Normal Counter	When the 8 on the left side is high. The display is set to 59. Then I pressed set and saved the state.
	
Set next state to 11 and save state	The maximum value of the last display is 23
	
When the set button is pressed start to count	Reset is pressed so start to count again

from desired position	
	

5. **Reflection**:

It was funny to work with the constraints because we needed to refine the code and follow the guidelines accurately.