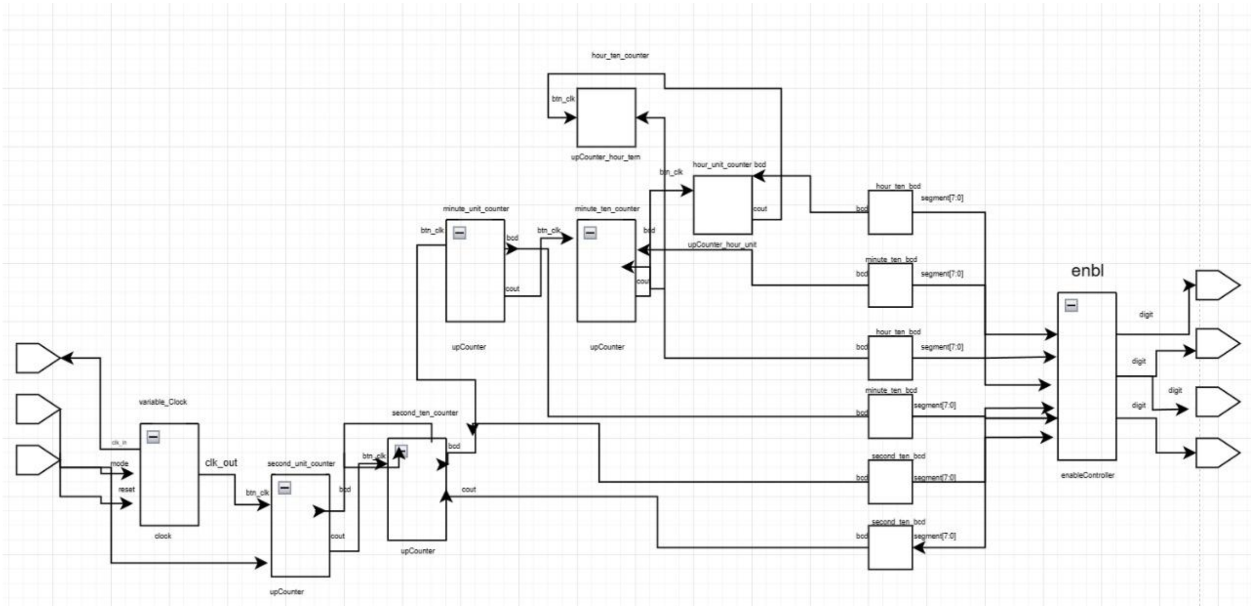
Digital Logic Design

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

1. **Topic**: 24 Hour Clock Display

2. **Design structure:**



3. **Code**:

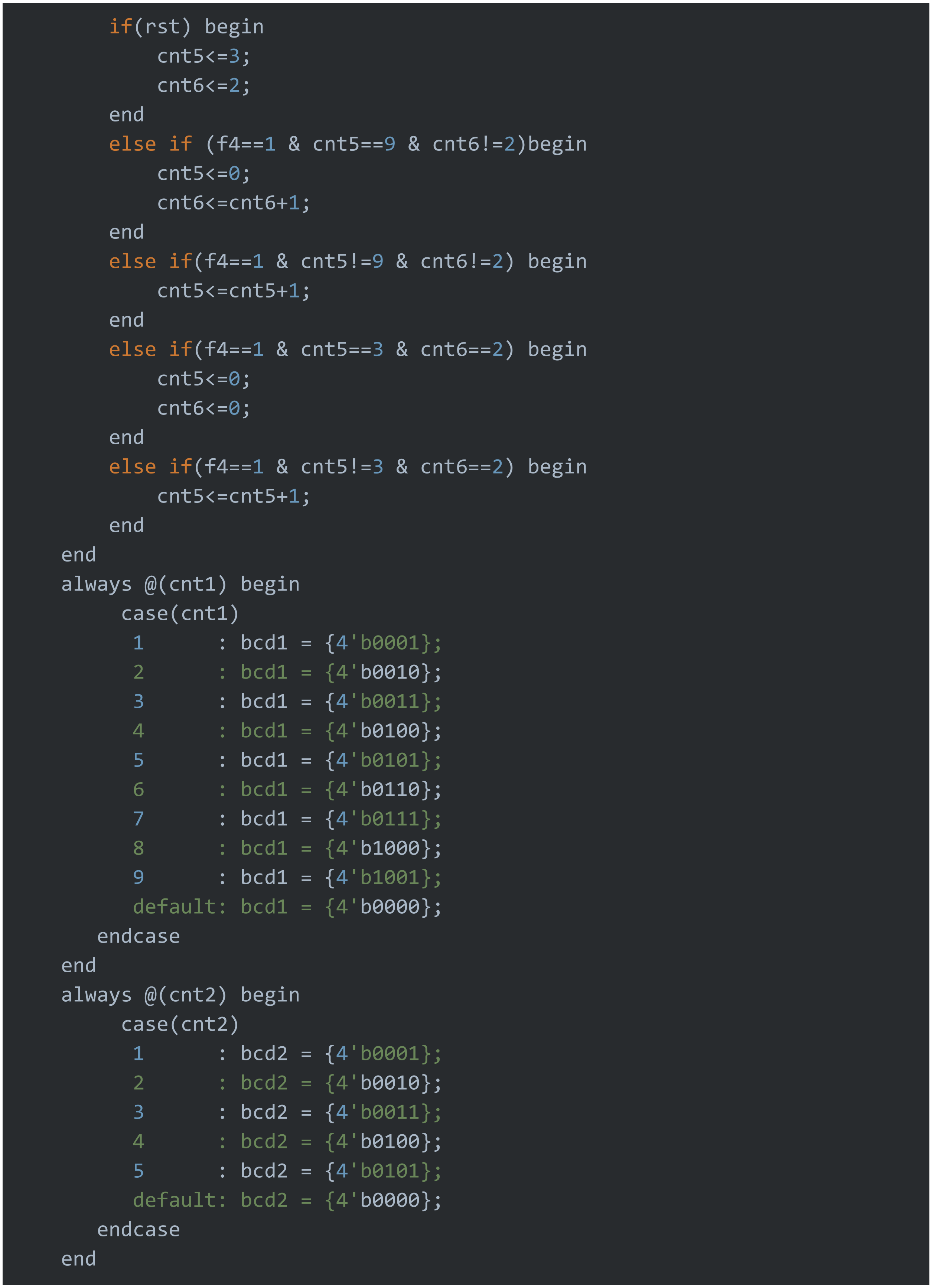
```
module clock_24h(clk, rst, clk_select, Y, Z, scan);
    input clk, rst, clk_select;
    output [7:0] Y, Z;
    output reg [5:0] scan;
    wire clk_fast_out, clk_1hz_out, clk_turbo_fast_out;
    reg clk_selected;
    wire [3:0] bcd1, bcd2, bcd3, bcd4, bcd5, bcd6;
    integer x;
    reg [3:0] bcd_out;
    clk_even_faster(clk, rst, clk_turbo_fast_out);
    clk_fast(clk, rst, clk_fast_out);
    clk_divider_1hz(clk,rst,clk_1hz_out);
    always @(clk_select)begin
        if (clk_select)begin
            clk_selected <= clk_turbo_fast_out;
        end
        else begin
```

```
            clk_selected <= clk_1hz_out;
        end
    end
    counters(clk_selected, rst, bcd1, bcd2, bcd3, bcd4, bcd5, bcd6);
    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);
```

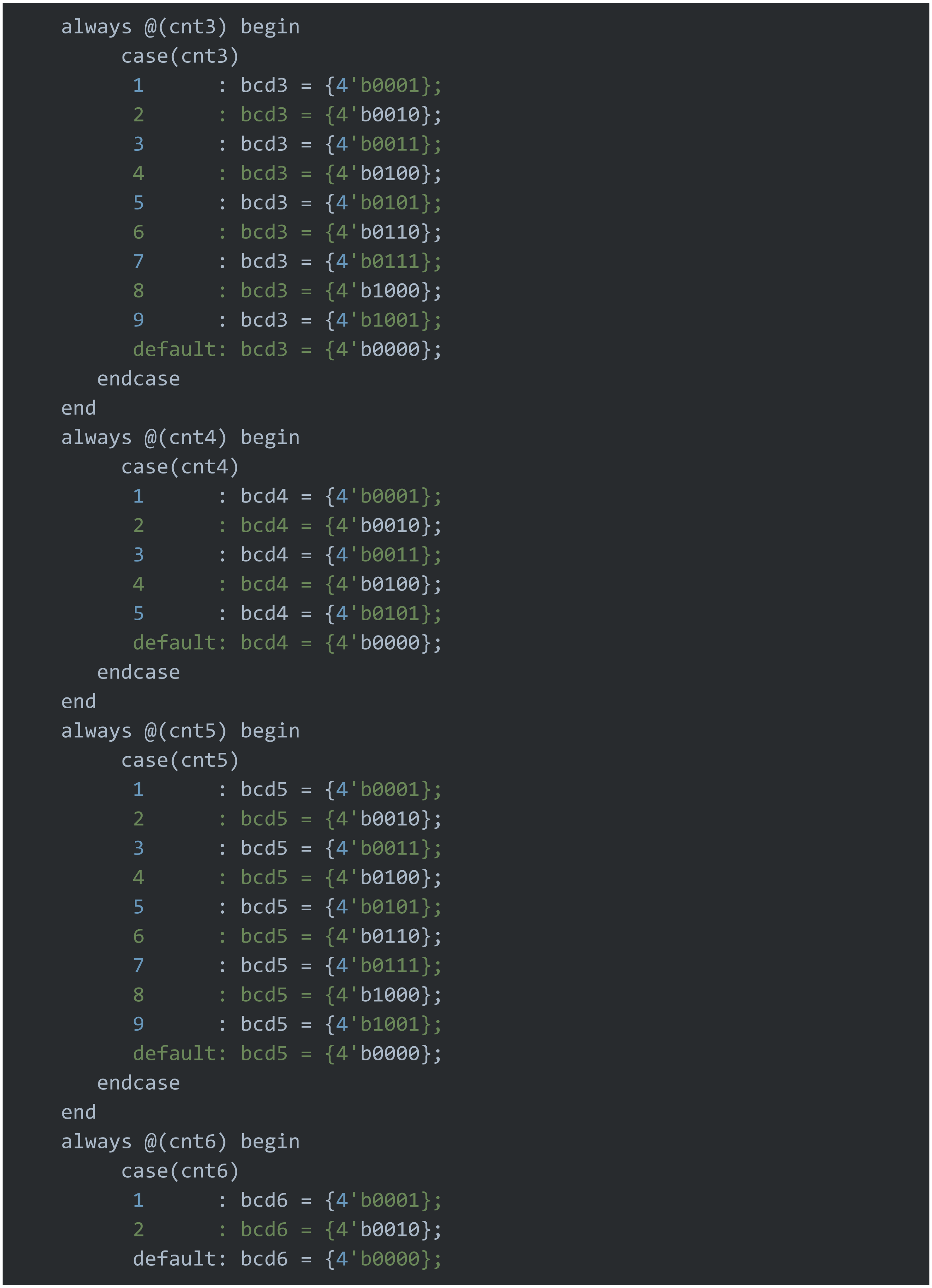
```
endmodule
```

```
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_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==10000)
        cnt<=0;
        else
        cnt<=cnt+1;
      end
    always @ (posedge(clk) or posedge rst)
    begin
        if (rst) clk_out <= 0;
        else if(cnt<=5000)clk_out<=1'b1;
        else clk_out<=1'b0;
    end
endmodule
module clk_even_faster(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==10000)
        cnt<=0;
        else
        cnt<=cnt+1;
      end
    always @ (posedge(clk) or posedge rst)
    begin
        if (rst) clk_out <= 0;
        else if(cnt<=5000)clk_out<=1'b1;
        else clk_out<=1'b0;
    end
endmodule
```

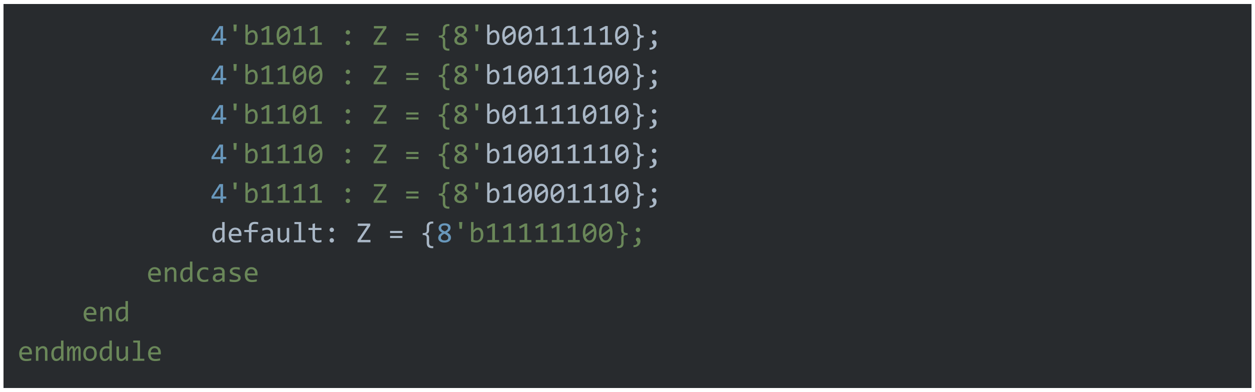
```
module counters(clk, rst, bcd1, bcd2, bcd3, bcd4, bcd5, bcd6);
    input clk, rst;
    output reg [3:0] bcd1, bcd2, bcd3, bcd4, bcd5, bcd6;
    integer cnt1, cnt2, cnt3, cnt4, cnt5, cnt6;
    reg f1, f2, f3, f4;
    always @(posedge clk or posedge rst) begin
        if(rst) begin
            cnt1<=4;
            f1<=0;
        end
        else if(cnt1==9) begin
            cnt1<=0;
            f1<=1;
        end
        else begin
            cnt1<=cnt1+1;
            f1<=0;
        end
    end
    always @(posedge rst or posedge f1) begin
        if(rst) begin
            cnt2<=4;
            f2<=0;
        end
        else if (f1==1 & cnt2==5)begin
            cnt2<=0;
            f2<=1;
        end
        else if(f1==1 & cnt2!=5) begin
            cnt2<=cnt2+1;
            f2<=0;
        end
    end
    always @(posedge rst or posedge f2) begin
        if(rst) begin
            cnt3<=9;
            f3<=0;
        end
        else if (f2==1 & cnt3==9)begin
            cnt3<=0;
            f3<=1;
        end
        else if(f2==1 & cnt3!=9) begin
            cnt3<=cnt3+1;
            f3<=0;
        end
    end
    always @(posedge rst or posedge f3) begin
        if(rst) begin
            cnt4<=5;
            f4<=0;
        end
        else if (f3==1 & cnt4==5)begin
            cnt4<=0;
            f4<=1;
        end
        else if(f3==1 & cnt4!=5) begin
            cnt4<=cnt4+1;
            f4<=0;
        end
    end
    always @(posedge rst or posedge f4) begin
```



```
        if(rst) begin
            cnt5<=3;
            cnt6<=2;
        end
        else if (f4==1 & cnt5==9 & cnt6!=2)begin
            cnt5<=0;
            cnt6<=cnt6+1;
        end
        else if(f4==1 & cnt5!=9 & cnt6!=2) begin
            cnt5<=cnt5+1;
        end
        else if(f4==1 & cnt5==3 & cnt6==2) begin
            cnt5<=0;
            cnt6<=0;
        end
        else if(f4==1 & cnt5!=3 & cnt6==2) begin
            cnt5<=cnt5+1;
        end
    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 = {4'b0001};
          2      : bcd2 = {4'b0010};
          3      : bcd2 = {4'b0011};
          4      : bcd2 = {4'b0100};
          5      : bcd2 = {4'b0101};
          default: bcd2 = {4'b0000};
       endcase
    end
```



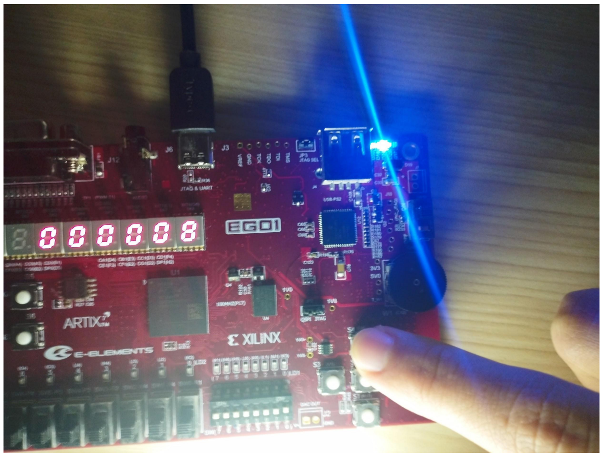
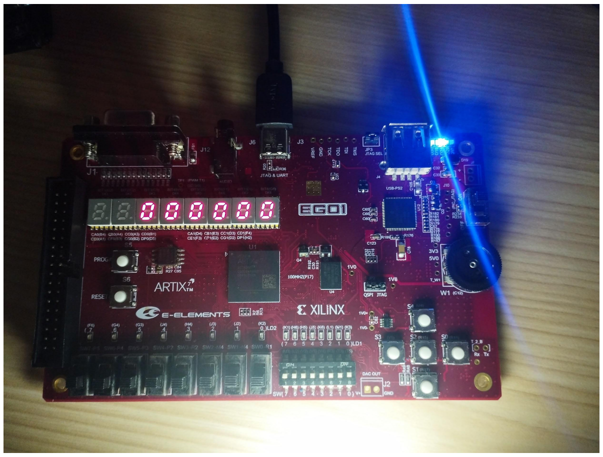
```
    always @(cnt3) begin
         case(cnt3)
          1      : bcd3 = {4'b0001};
          2      : bcd3 = {4'b0010};
          3      : bcd3 = {4'b0011};
          4      : bcd3 = {4'b0100};
          5      : bcd3 = {4'b0101};
          6      : bcd3 = {4'b0110};
          7      : bcd3 = {4'b0111};
          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};
```

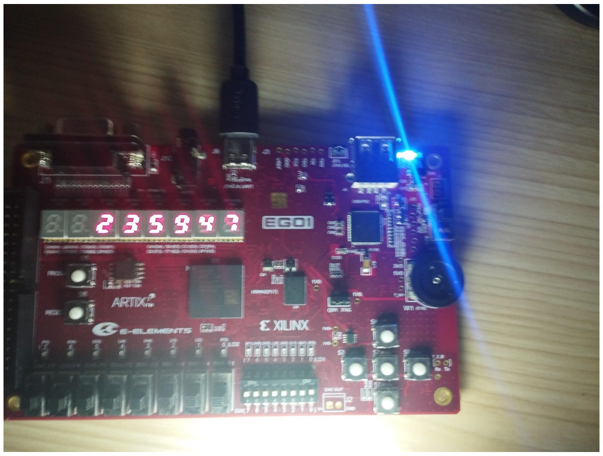
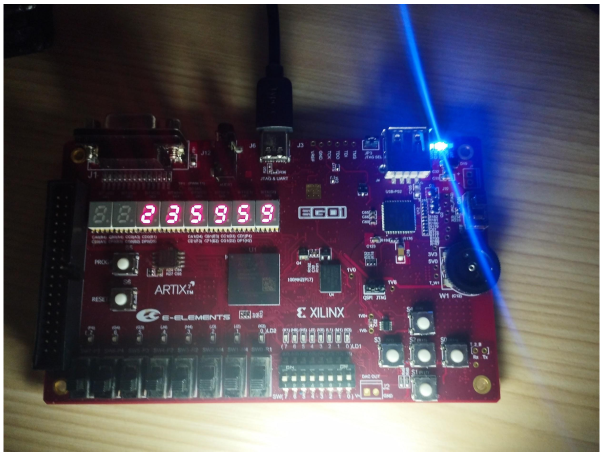
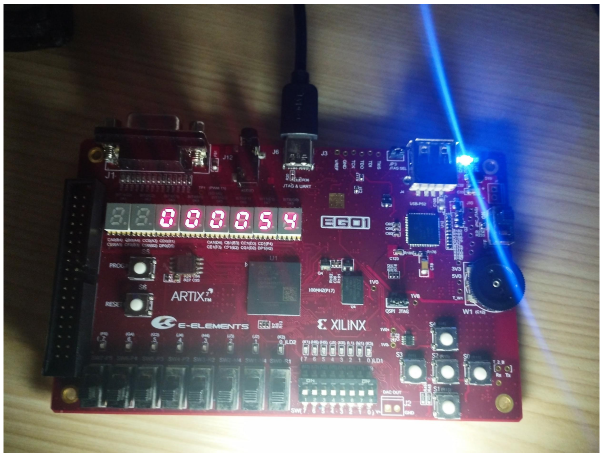


```
            4'b1011 : Z = {8'b00111110};
            4'b1100 : Z = {8'b10011100};
            4'b1101 : Z = {8'b01111010};
            4'b1110 : Z = {8'b10011110};
            4'b1111 : Z = {8'b10001110};
            default: Z = {8'b11111100};
        endcase
    end
endmodule
```

4. **Result**:

This adder can sum four bits, the improvement is that it can count to 2^4. The sum of the last state generates all sum bits equal to one and also the carry.

Counting normal	When U4 is pressed reset the counter
	
Is r1 is pressed, the clock is set to 23:59:45	When reset button is pressed start with 00:00:00

	
Normal counter	
	

5. **Reflection**:

It was very interesting to work again with the seven segment and implement the different counter states.