THE AMERICAN UNIVERSITY IN CAIRO

Digital Alarm Clock

CSCE 2301

Spring 2024

By:

Jana fadl 900221734

Rodayna Elkhouly 900221860

Abdallah Afifi 900225283

Adham Salem 900222465

CSCE 2301 Spring 2024

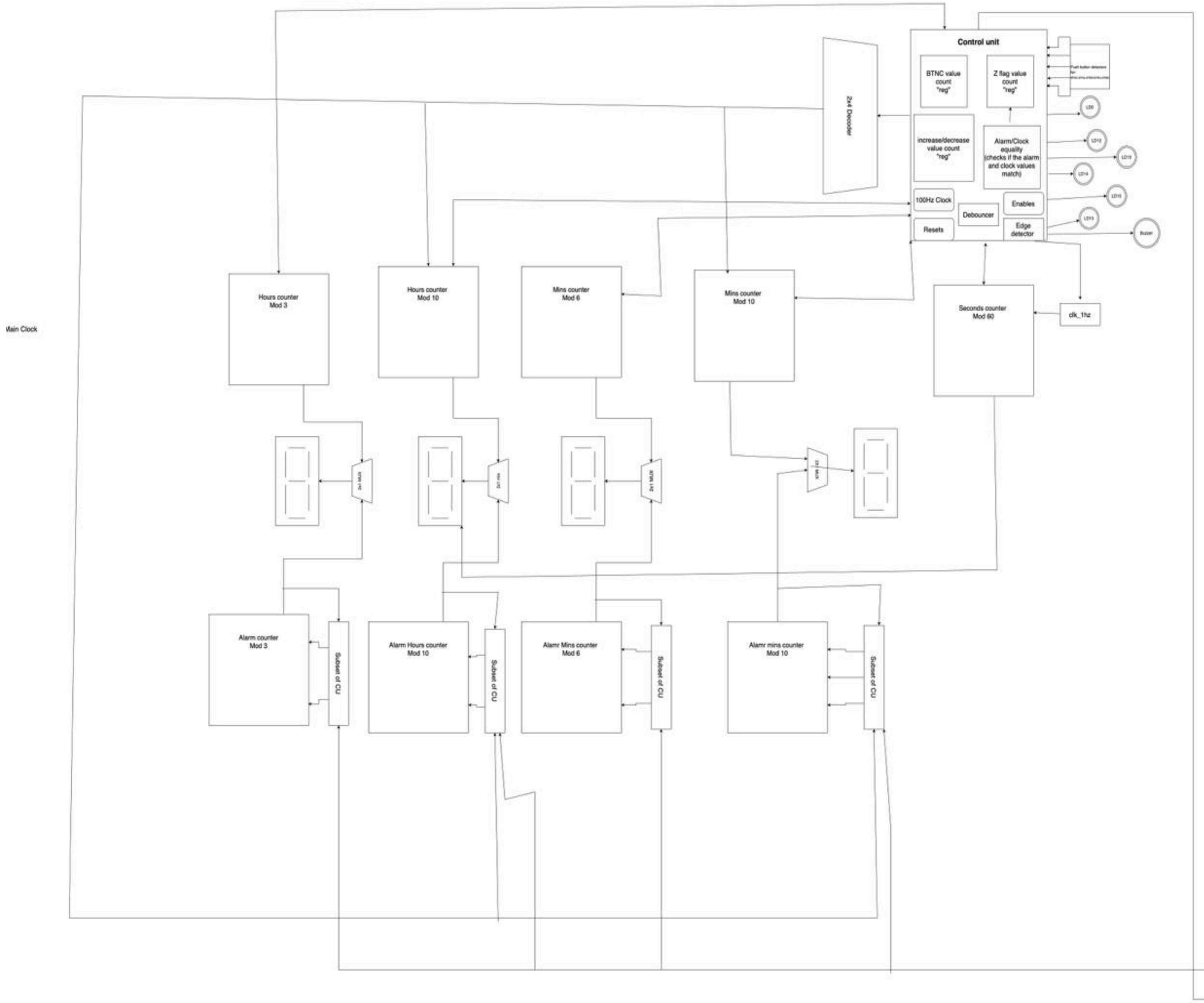
Dr. Mohamed Shaalan

Digital Design I - Project 2 Report

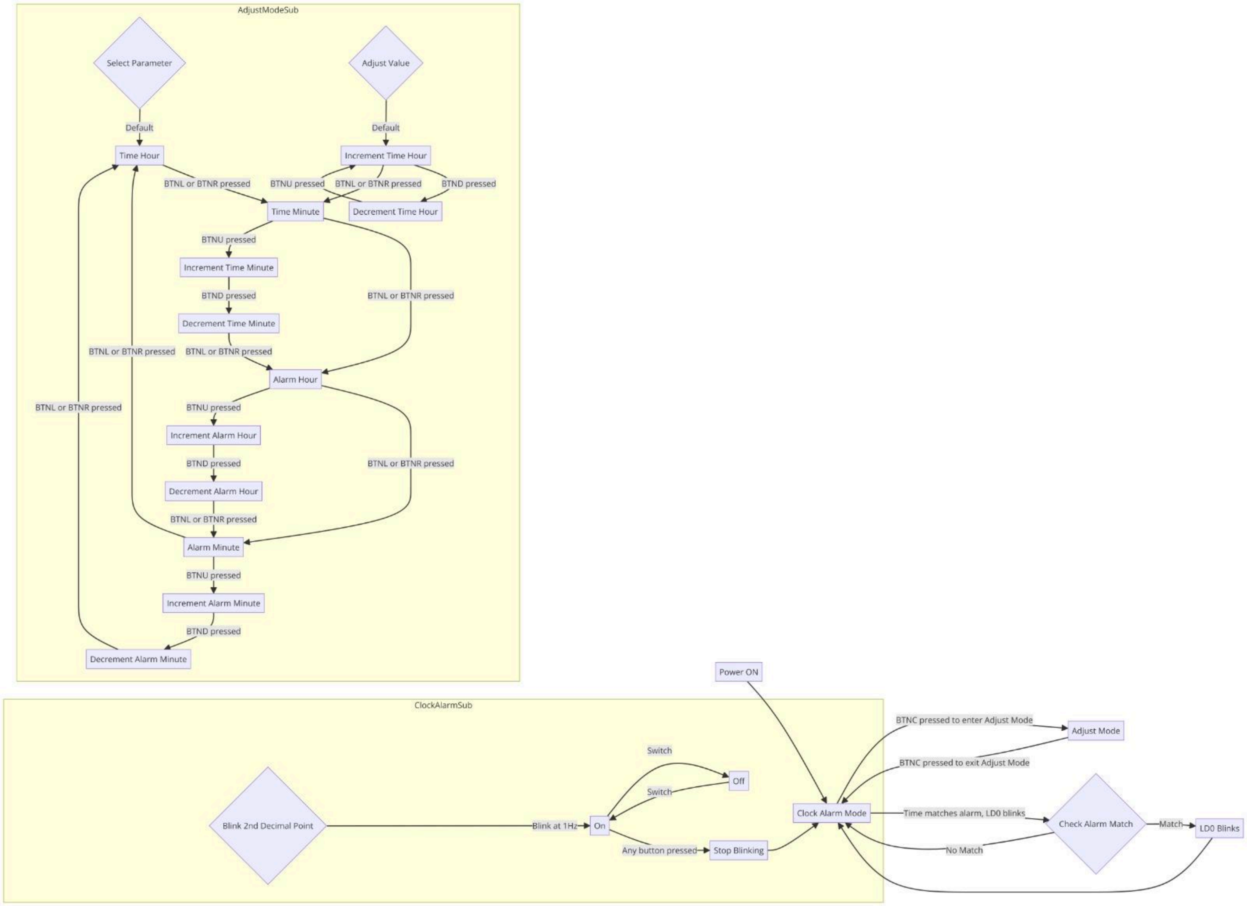
Outlining the design:

The design for the Digital_Clock_Mod module is structured around a mealy finite state machine (FSM) with multiple states to handle different modes of operation for a digital clock with adjustable time and alarm features. The main states include *Clock_State, Adjust_mode, Adjust_mins_clock, adjust_alarm_hours, adjust_alarm_minutes* and *Adjust_hours_clock,* among others. Each state governs the behavior of the clock based on button inputs for incrementing or decrementing time values. The clock operates with a 1 Hz clock for regular timekeeping and a 200 Hz clock for faster adjustments during time setting. The system integrates pushbutton detectors to process button presses, enabling the transition between states. The clock divider modules generate the necessary clock frequencies. In *Clock_State,* the clock operates normally, displaying the current time. Pressing the center button (BTNC) transitions the system into *Adjust_mode*, allowing for time adjustments. The right (BTNR) and left (BTNL) buttons navigate between adjusting minutes and hours. In the adjustment states, the up (BTNU) and down (BTND) buttons increment or decrement the time valuesThe state machine uses LEDs to indicate the active mode and adjustment status. The digital clock's state transitions and adjustments are visually represented in the provided flowchart, which outlines the interactions between states based on button inputs. The flowchart illustrates the hierarchical structure of the FSM, detailing the transitions from time display to adjustment modes for both time and alarm settings.

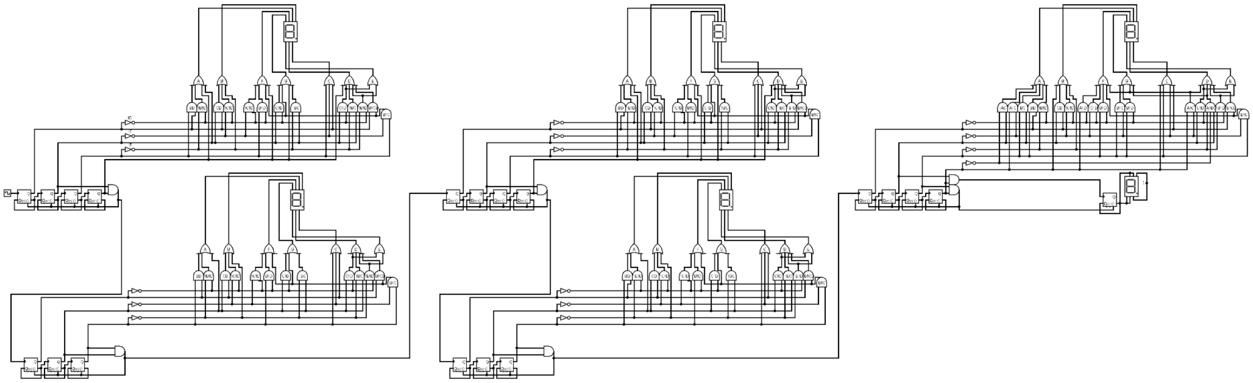
Final_block_diagram:



ASM Chart:



Logism Simulation:



Implementation Issues:

1) Error in increment/decrement

During the process of implementing the code, we faced some problems in implementing the increment and decrement which lead to the clock not working.

Code Before:

```
  if (debounced_BTNU) begin
            case (adjust_selection)
                0: hours <= (hours < 23 ? hours + 1 : 0);
                1: minutes <= (minutes < 59 ? minutes + 1 : 0);
                2: alarm_hours <= (alarm_hours < 23 ? alarm_hours + 1 : 0);
                3: alarm_minutes <= (alarm_minutes < 59 ? alarm_minutes + 1 : 0);
            endcase
        end
        if (debounced_BTND) begin
            case (adjust_selection)
                0: hours <= (hours > 0 ? hours - 1 : 23);
                1: minutes <= (minutes > 0 ? minutes - 1 : 59);
                2: alarm_hours <= (alarm_hours > 0 ? alarm_hours - 1 : 23);
                3: alarm_minutes <= (alarm_minutes > 0 ? alarm_minutes - 1 : 59);
            endcase
        end
```

After:

So in order to fix this issue we implemented a module with inputs of clock, enable, reset and button inputs for controlling the clock's systems (BTNC,BTNR,BTNL,BTNU,BTND) , the outputs are the 7-segment display outputs (segments), decimal point control for the 7-segment display (DP) , and manage LED indicators. The internal logic consists of a state machine with states for normal clock operation and various adjustment modes for minutes and hours. The Hours_Mins_Secs submodule handles the counting of time, while multiplexers and counters manage the display logic. This design allows users to easily switch between viewing the current time and adjusting the minutes and hours using button inputs, with visual feedback provided by LEDs.

2) Hours, Minutes and Seconds Counter

While designing the reset by implementing the counter for the hours, minutes, and seconds counter, we faced some problems

To fix it We made the up and down flag in the Mod_N_Counter :

```
Mod_N_Counter #(2, 4) BC (clk_200, reset, en,Up_Down_en, sel);
```

3) Adjust Mode

Another challenge we faced was in implementing the adjust mode, we specifically had errors in switching states.

Code Before:

```
Adjust_mode: if (BNTC==1'b1) nextState = Clock_State;
 else if(BTNR==1'b1) nextState = Adjust_mins_clock;
 else if(BTNL==1'b1) nextstate=Adjust_mins_alarm;
 else nextState=Adjust_mode;
```

```
Adjust_mins_clock: if (BTNC==1'b1) nextState = Clock_State;
 else if(BTNR==1'b1) nextState = Adjust_hours_clock;
 else if(BTNL==1'b1) nextState =Adjust_mins_alarm;  //incomplete code, we need to make the state where when we push BTNU or BTNL, we add/dec
 else nextState=Adjust_mins_clock;
Adjust_hours_clock: if (BTNC==1'b1) nextState = Clock_State;
 else if (BTNR==1'b1) nextState = Adjust_mins_clock;
 else if (BTNL==1'b1) nextState=Adjust_hours_alarm; //incomplete code, we need to make the state where when we push BTNU or BTNL, we add/dec
 else nextState=Adjust_hours_clock;
 Adjust_hours_alarm: if (BTNC==1'b1) nextState = Clock_State;
 else if (BTNL==1'b1) nextState = Adjust_mins_alarm;
 else if (BTNR==1'b1) nextState=Adjust_mins_clock;
 else nextState=Adjust_hours_alarm;


Adjust_mins_alarm:  if(BNTC==1'b1) nextState = Clock_State;
                    else if (BTNR==1'b1) nextState=Adjust_mins_alarm;
                    else if (BTNL==1'b1) nextState=Adjust_mins_alarm;
                    else nextState=Adjust_mins_alarm;
default: nextState = Clock_State;
endcase
```

Code After:

```
Adjust_mode:
    if (outbutton == 5'b00001) begin // BTNC
      clk_input = clk_1hz;
      nextState = Clock_State;
      Up_Down_en = 1;

      enable_clock = 1;

      enMins = 0;

      enHours = 0;

      led[0] = 1'b0;

      led[1] = 1'b0;

      led[2] = 1'b0;

      led[3] = 1'b0;

      led[4] = 1'b0;

      match=0;

      enMins_a = 0;

      enHours_a = 0;

    end

    else if (outbutton == 5'b00010) begin  // BTNR

      nextState = Adjust_mins_clock;

      clk_input = clk_200;

      led[0] = 1'b1;

      led[1] = 1'b1;

      led[2] = 1'b0;

      Up_Down_en = 1;

      enMins = 0;

      enHours = 0;

      led[3] = 1'b0;

      led[4] = 1'b0;

      enable_clock = 0;

      enMins_a = 0;
      enHours_a = 0;
      match=0;
    end
    else if (outbutton == 5'b00100) begin  // BTNL
      nextState = Adjust_hours_alarm;
      clk_input = clk_200;
      led[0] = 1'b1;
      led[1] = 1'b0;
      led[2] = 1'b0;
      Up_Down_en = 1;
      enMins = 0;
      enHours = 0;
      led[3] = 1'b0;
      led[4] = 1'b1;
      led[5]=1'b0;
      enable_clock = 0;
       enMins_a = 0;
      enHours_a = 0;
      match=0;
    end
    else begin
      nextState = Adjust_mode;
      led[0] = 1'b1;
      clk_input = clk_200;
      led[1] = 1'b0;
      led[2] = 1'b0;
```

```
      led[3] = 1'b0;

      led[4] = 1'b0;

      enMins = 0;

      enHours = 0;

      Up_Down_en = 1;

      enable_clock = 0;

       enMins_a = 0;

      enHours_a = 0;

      match=0;

    end


Adjust_hours_clock:

    if (outbutton[0] == 1'b1) begin

      nextState = Clock_State;

      clk_input = clk_1hz;

      Up_Down_en = 1;

      enable_clock = 1;

      enMins = 0;

      enHours = 0;

      led[0] = 1'b0;

      led[1] = 1'b0;

      led[2] = 1'b0;

      led[3] = 1'b0;

      led[4] = 1'b0;

      led[5]=1'b0;

       enMins_a = 0;
    enHours_a = 0;
    match=0;
end
else if (outbutton[1] == 1'b1) begin
    nextState = Adjust_mins_clock;
    clk_input = clk_200;
    led[0] = 1'b1;
    led[1] = 1'b0;
    led[3] = 1'b0;
    led[4] = 1'b0;
    led[2] = 1'b1;
    led[5]=1'b0;
    enable_clock = 0;
    Up_Down_en = 1;
    enMins = 0;
    enHours = 0;
    enMins_a = 0;
    enHours_a = 0;
    match=0;
end
else if (outbutton == 5'b00100) begin  // BTNL
    nextState = Adjust_mins_alarm;
    clk_input = clk_200;
    led[0] = 1'b1;
    led[1] = 1'b0;
    led[2] = 1'b0;
    enMins = 0;
    enHours = 0;
    led[3] = 1'b1;
    Up_Down_en = 1;
    led[4] = 1'b0;
    enable_clock = 0;
    enMins_a = 0;
    enHours_a = 0;
    match=0;
end
else if (outbutton[3] == 1'b1) begin // BTNU
    nextState = Adjust_hours_clock;
    enable_clock = 0;
    Up_Down_en = 1;
    enMins = 0;
    enHours = 1;
    clk_input = clk_200;
    led[0] = 1'b1;
    led[1] = 1'b0;
    led[2] = 1'b1;
    led[3] = 1'b0;
    led[4] = 1'b0;
    enMins_a = 0;
    enHours_a = 0;
    match=0;
end
```

```
else if (outbutton[4] == 1'b1) begin // BTND
   nextState = Adjust_hours_clock;
   clk_input = clk_200;
   enable_clock = 0;
   Up_Down_en = 0;
   enMins = 0;
   enHours = 1;
   led[0] = 1'b1;
   led[1] = 1'b0;
   led[2] = 1'b1;
   led[3] = 1'b0;
   led[4] = 1'b0;
    enMins_a = 0;
   enHours_a = 0;
   match=0;
end
else begin
   nextState = Adjust_hours_clock;
   clk_input = clk_200;
   led[0] = 1'b1;
   led[1] = 1'b0;
   led[2] = 1'b1;
   Up_Down_en = 1;
   enMins = 0;
   enHours = 0;
   led[3] = 1'b0;
```

```
            led[4] = 1'b0;

            enable_clock = 0;

            enMins_a = 0;

            enHours_a = 0;

            match=0;

        end


Adjust_mins_clock:

        if (outbutton[0] == 1'b1) begin

            nextState = Clock_State;

            clk_input = clk_1hz;

            Up_Down_en = 1;

            enable_clock = 1;

            enMins = 0;

            enHours = 0;

            led[0] = 1'b0;

            led[1] = 1'b0;

            led[2] = 1'b0;

            led[3] = 1'b0;

            led[4] = 1'b0;

             enMins_a = 0;

            enHours_a = 0;

            match=0;

        end

        else if (outbutton[1] == 1'b1) begin

            nextState = Adjust_hours_alarm;
```

```
   clk_input = clk_200;
   led[0] = 1'b1;
   led[1] = 1'b0;
   led[2] = 1'b0;
   led[3] = 1'b0;
   led[4] = 1'b1;
   enable_clock = 0;
   Up_Down_en = 1;
   enMins = 0;
   enHours = 0;
   enMins_a = 0;
   enHours_a = 0;
   match=0;
end
 else if (outbutton == 5'b00100) begin  // BTNL
   nextState = Adjust_hours_alarm;
   clk_input = clk_200;
   led[0] = 1'b1;
   led[1] = 1'b0;
   led[2] = 1'b0;
   enMins = 0;
   enHours = 0;
   led[4] = 1'b0;
   Up_Down_en = 1;
   led[4] = 1'b0;
   enable_clock = 0;
```

```
    enMins_a = 0;
    enHours_a = 0;
    match=0;
end

else if (outbutton[3] == 1'b1) begin // BTNU
    nextState = Adjust_mins_clock;
    clk_input = clk_200;
    enable_clock = 0;
    Up_Down_en = 1;
    enMins = 1;
    enHours = 0;
    led[0] = 1'b1;
    led[1] = 1'b1;
    led[2] = 1'b0;
    led[3] = 1'b0;
    led[4] = 1'b0;
     enMins_a = 0;
    enHours_a = 0;
    match=0;
end
else if (outbutton[4] == 1'b1) begin // BTND
    nextState = Adjust_mins_clock;
    clk_input = clk_200;
    enable_clock = 0;
    Up_Down_en = 0;
    enMins = 1;
    enHours = 0;
    led[0] = 1'b1;
    led[1] = 1'b1;
    led[2] = 1'b0;
    led[3] = 1'b0;
    led[4] = 1'b0;
     enMins_a = 0;
    enHours_a = 0;
    match=0;
end
else begin
    nextState = Adjust_mins_clock;
    clk_input = clk_200;
    led[0] = 1'b1;
    led[1] = 1'b1;
    led[2] = 1'b0;
    led[3] = 1'b0;
    led[4] = 1'b0;
    enMins = 0;
    enHours = 0;
    enable_clock = 0;
    Up_Down_en = 1;
     enMins_a = 0;
    enHours_a = 0;
    match=0;
    end



Adjust_mins_alarm:
    if (outbutton[0] == 1'b1) begin
      nextState = Clock_State;
      clk_input = clk_1hz;
      Up_Down_en = 1;
      enMins=0;
      enHours=0;
      enMins_a = 0;
      enHours_a = 0;
      led[0] = 1'b0;
      led[1] = 1'b0;
      led[2] = 1'b0;
      led[3] = 1'b0;
      led[4] = 1'b0;
      match=0;

    end
    else if (outbutton[1] == 1'b1) begin
      nextState = Adjust_hours_clock;
      clk_input = clk_200;
     enMins=0;
     enHours=0;
      Up_Down_en = 1;
    enMins_a = 0;
    enHours_a = 0;
    led[0] = 1'b1;
    led[1] = 1'b0;
    led[2] = 1'b1;
    led[3] = 1'b0;
    led[4] = 1'b0;
    match=0;
end
 else if (outbutton == 5'b00100) begin  // BTNL
    nextState = Adjust_hours_clock;
    clk_input = clk_200;
    led[0] = 1'b1;
    led[1] = 1'b0;
    led[2] = 1'b1;
    Up_Down_en = 1;
    enMins = 0;
    enHours = 0;
    led[3] = 1'b0;
    led[4] = 1'b0;
    enable_clock = 0;
     enMins_a = 0;
    enHours_a = 0;
    match=0;
end
```

```
else if (outbutton[3] == 1'b1) begin // BTNU
   nextState = Adjust_mins_alarm;
   clk_input = clk_200 ;
   enMins=0;
   enHours=0;
   Up_Down_en = 1;
   enMins_a = 1;
   enHours_a = 0;
   led[0] = 1'b1;
   led[1] = 1'b0;
   led[2] = 1'b0;
   led[3] = 1'b1;
   led[4] = 1'b0;
   match=0;
end
else if (outbutton[4] == 1'b1) begin // BTND
   nextState = Adjust_mins_alarm;
   clk_input = clk_200 ;
   enMins=0;
   enHours=0;
   Up_Down_en = 0;
   enMins_a = 1;
   enHours_a = 0;
   led[0] = 1'b1;
   led[1] = 1'b0;
   led[2] = 1'b0;
```

```
            led[3] = 1'b1;
            led[4] = 1'b0;
            match=0;
        end
        else begin
            nextState = Adjust_mins_alarm;
            led[0] = 1'b1;
            led[1] = 1'b0;
            led[2] = 1'b0;
            led[3]=1'b1;
            enMins_a = 0;
            enHours_a = 0;
            clk_input = clk_200 ;
            enMins=0;
            enHours=0;
            Up_Down_en = 1;
            match=0;
        end

    Adjust_hours_alarm:
        if (outbutton[0] == 1'b1) begin
            nextState = Clock_State;
            clk_input = clk_1hz  ;
            enMins=0;
            enHours=0;
            Up_Down_en = 1;
```

```
   enMins_a = 0;
   enHours_a = 0;
   led[0] = 1'b0;
   led[1] = 1'b0;
   led[2] = 1'b0;
   led[3] = 1'b0;
   led[4] = 1'b0;
   match=0;
end
else if (outbutton[1] == 1'b1) begin
   nextState = Adjust_mins_alarm;
   clk_input = clk_200  ;
   enMins=0;
   enHours=0;
   Up_Down_en = 1;
   enMins_a = 0;
   enHours_a = 0;
   led[0] = 1'b1;
   led[1] = 1'b0;
   led[2] = 1'b0;
   led[3] = 1'b1;
   led[4] = 1'b0;
   match=0;
end
 else if (outbutton == 5'b00100) begin  // BTNL
   nextState = Adjust_mins_clock;
```

```
    clk_input = clk_200;

    led[0] = 1'b1;

    led[1] = 1'b1;

    led[2] = 1'b0;

    Up_Down_en = 1;

    enMins = 0;

    enHours = 0;

    led[3] = 1'b0;

    led[4] = 1'b0;

    enable_clock = 0;

     enMins_a = 0;

    enHours_a = 0;

    match=0;

end

else if (outbutton[3] == 1'b1) begin // BTNU

    nextState = Adjust_hours_alarm;

    clk_input = clk_200  ;

    enMins=0;

    enHours=0;

    Up_Down_en = 1;

    enMins_a = 0;

    enHours_a = 1;

    led[0] = 1'b1;

    led[1] = 1'b0;

    led[2] = 1'b0;

    led[3] = 1'b0;
    led[4] = 1'b1;

    match=0;

end

else if (outbutton[4] == 1'b1) begin // BTND

    nextState = Adjust_hours_alarm;

    clk_input = clk_200  ;

    enMins=0;

    enHours=0;

    Up_Down_en = 0;

    enMins_a = 0;

    enHours_a = 1;

    led[0] = 1'b1;

    led[1] = 1'b0;

    led[2] = 1'b0;

    led[3] = 1'b0;

    led[4] = 1'b1;

    match=0;

end

else begin

    nextState = Adjust_hours_alarm;

    clk_input = clk_200 ;

    enMins=0;

    enHours=0;

    Up_Down_en = 1;

    enMins_a = 0;

    enHours_a = 0;
```

```
            led[0] = 1'b1;
            led[1] = 1'b0;
            led[2] = 1'b0;
            led[3] = 1'b0;
            led[4] = 1'b1;
            match=0;
        end
endcase
```

Validation Activities:

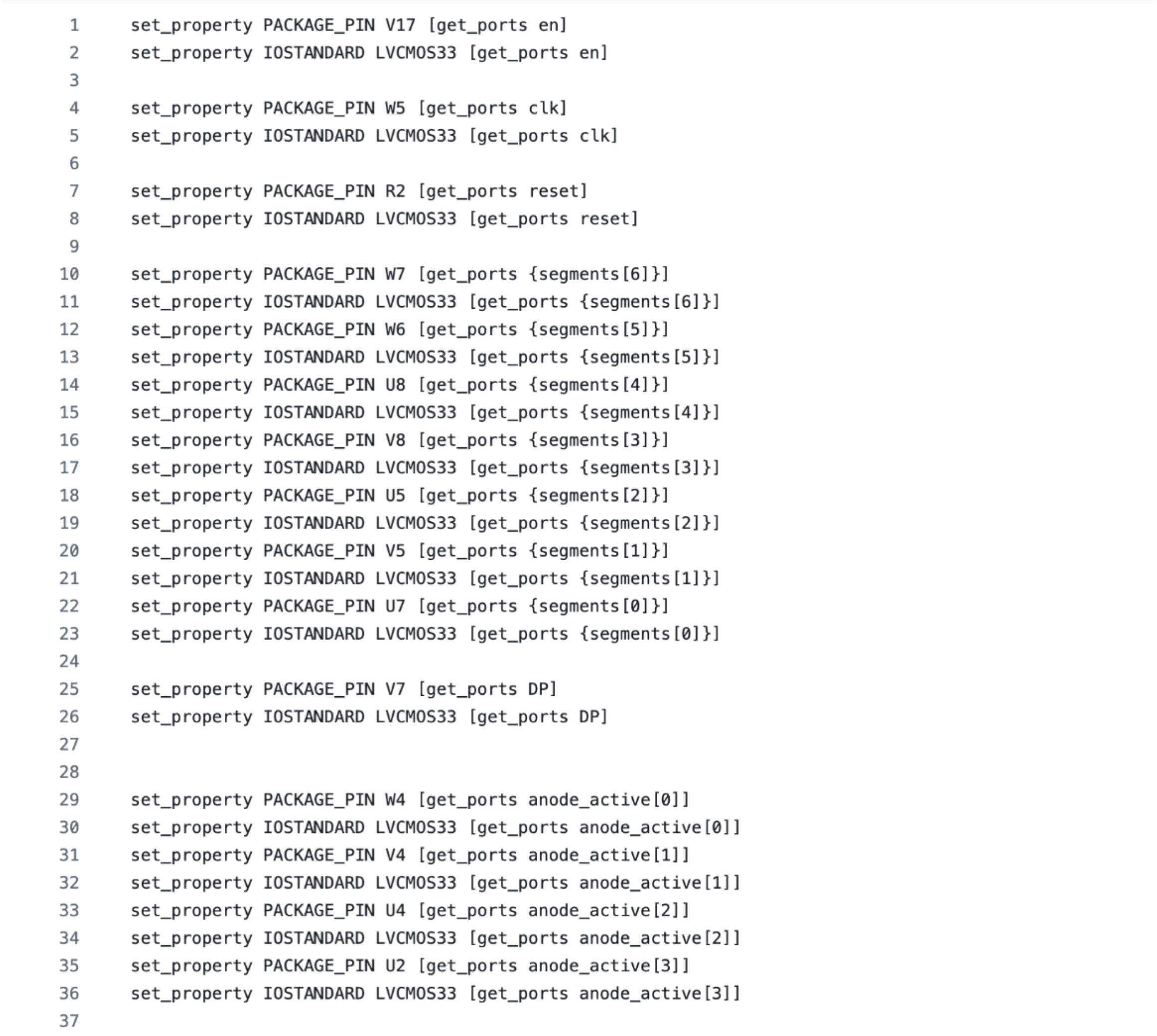
We began our evaluation of the Digital_Clock_Mod module by examining each state within the finite state machine individually to ensure they functioned correctly when isolated. We then tested how well key components, such as the clock divider and pushbutton detectors, worked together, ensuring they effectively triggered state transitions.

We thoroughly tested all possible button press scenarios to confirm that each button action resulted in the correct state change, simulating real-world user interactions. We also focused on the overall operation of the clock, particularly its accuracy and responsiveness to time adjustments, by running long-term simulations to detect any drift in timekeeping.

User experience was evaluated by assessing the clarity of the LED indicators, which are crucial for intuitive operation. Additionally, we tested the clock's resilience by simulating power outages to see if it could recover correctly and maintain accurate settings.

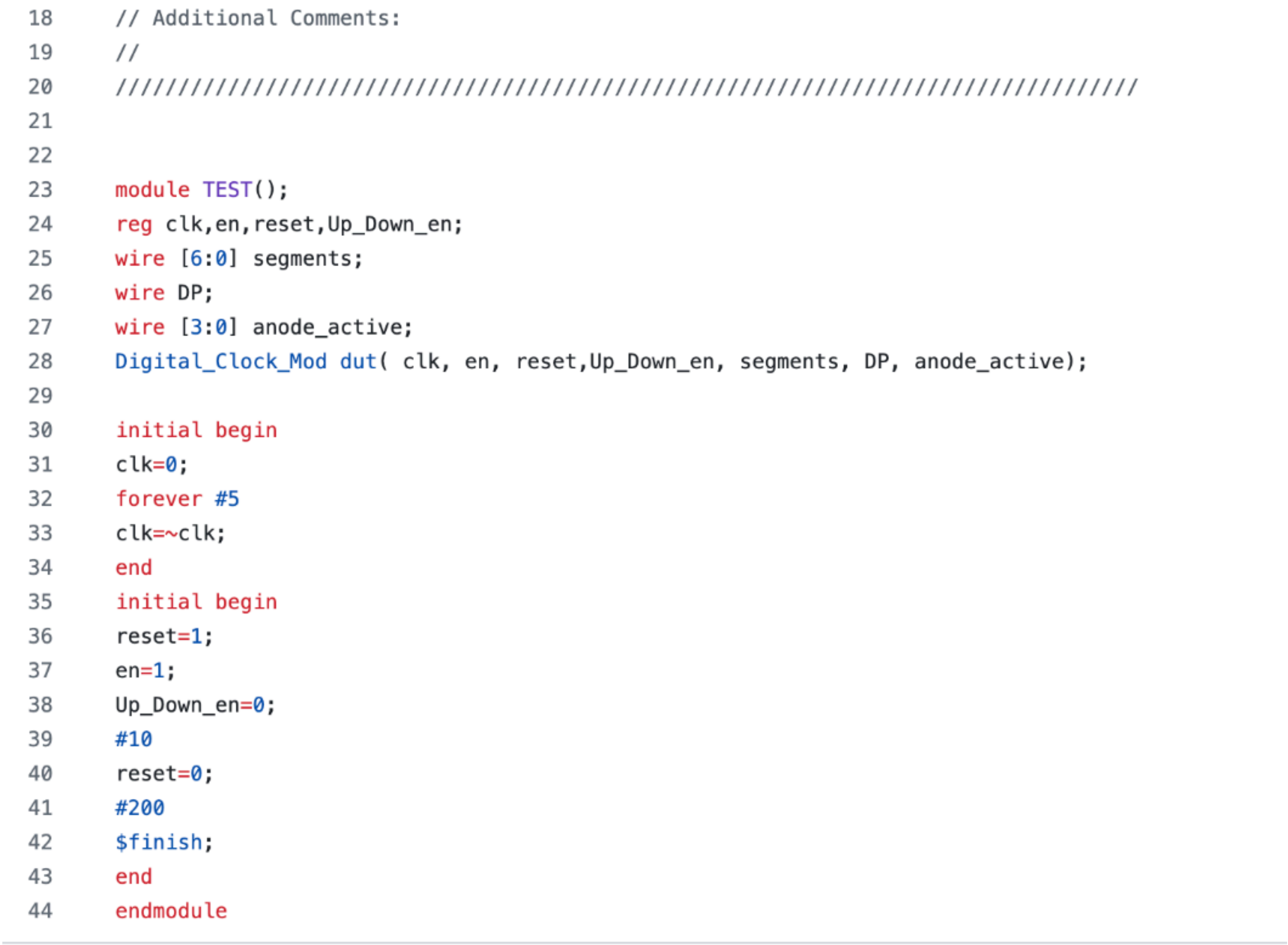
Lastly, we checked the system's responsiveness and long-term reliability to ensure that it would continue to function accurately and respond promptly to user inputs over time.

Constraint:



```
set_property PACKAGE_PIN V17 [get_ports en]
set_property IOSTANDARD LVCMOS33 [get_ports en]

set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

set_property PACKAGE_PIN R2 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports reset]

set_property PACKAGE_PIN W7 [get_ports {segments[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[6]}]
set_property PACKAGE_PIN W6 [get_ports {segments[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[5]}]
set_property PACKAGE_PIN U8 [get_ports {segments[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[4]}]
set_property PACKAGE_PIN V8 [get_ports {segments[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[3]}]
set_property PACKAGE_PIN U5 [get_ports {segments[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[2]}]
set_property PACKAGE_PIN V5 [get_ports {segments[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[1]}]
set_property PACKAGE_PIN U7 [get_ports {segments[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[0]}]

set_property PACKAGE_PIN V7 [get_ports DP]
set_property IOSTANDARD LVCMOS33 [get_ports DP]


set_property PACKAGE_PIN W4 [get_ports anode_active[0]]
set_property IOSTANDARD LVCMOS33 [get_ports anode_active[0]]
set_property PACKAGE_PIN V4 [get_ports anode_active[1]]
set_property IOSTANDARD LVCMOS33 [get_ports anode_active[1]]
set_property PACKAGE_PIN U4 [get_ports anode_active[2]]
set_property IOSTANDARD LVCMOS33 [get_ports anode_active[2]]
set_property PACKAGE_PIN U2 [get_ports anode_active[3]]
set_property IOSTANDARD LVCMOS33 [get_ports anode_active[3]]
```

TestBench



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


module TEST();
reg clk,en,reset,Up_Down_en;
wire [6:0] segments;
wire DP;
wire [3:0] anode_active;
Digital_Clock_Mod dut( clk, en, reset,Up_Down_en, segments, DP, anode_active);

initial begin
clk=0;
forever #5
clk=~clk;
end
initial begin
reset=1;
en=1;
Up_Down_en=0;
#10
reset=0;
#200
$finish;
end
endmodule
```

Buzzer:

assign buzzer = (state == AlarmMod) ? clk_buzz:0;

Contributions:

Adham Mohamed: Worked on the FSM and switching the states form clock to alarm state and the switching of the display so it has different displays for the clock mod and adjust mod, and worked on the bonus.

Abdallah Afifi: Modified the Mod N counters so it can accurately display the clock and resets at 23:59 to 00:00, Adjusted the Mod_N_Counter so it can increment and dicrement “Adjust the clock”, and worked on the Alarm_Mod.

Jana Fadl: Initial Mod_N_Counter, initial mod counters for the calculation of the minutes and hours and decoder.

Rodayna Mamdouh: Initial 7_Segement display and 4X1 Mux