***Adham Mohamed***

***Abdallah Afifi***

***Jana Fadl***

***Rodayna Mamdouh***

***CSCE 2301 Spring 2024***

***Dr. Mohamed Shaalan***

**Digital Design I - Project 2 Report**

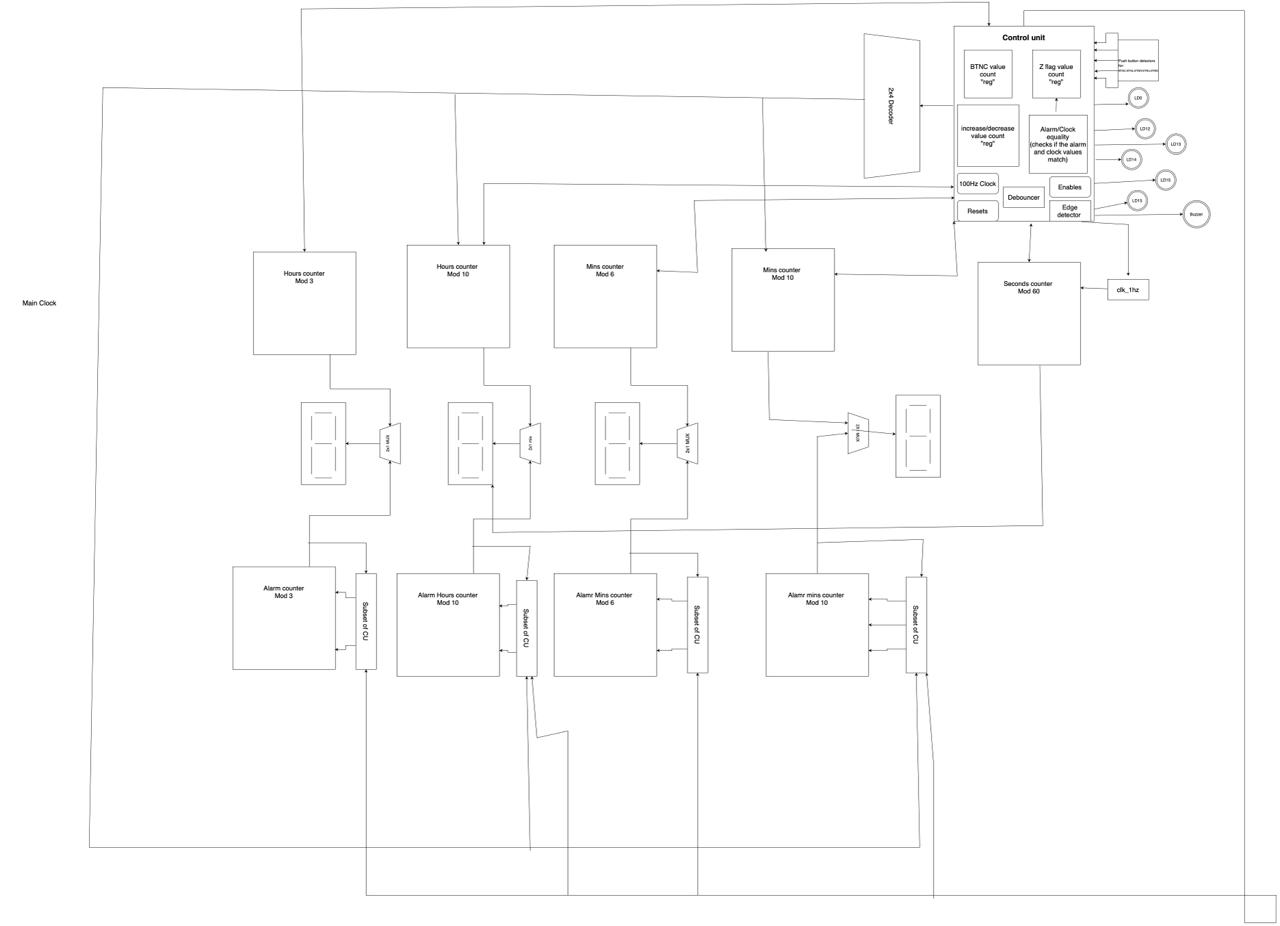
**Outlining the design:**

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.

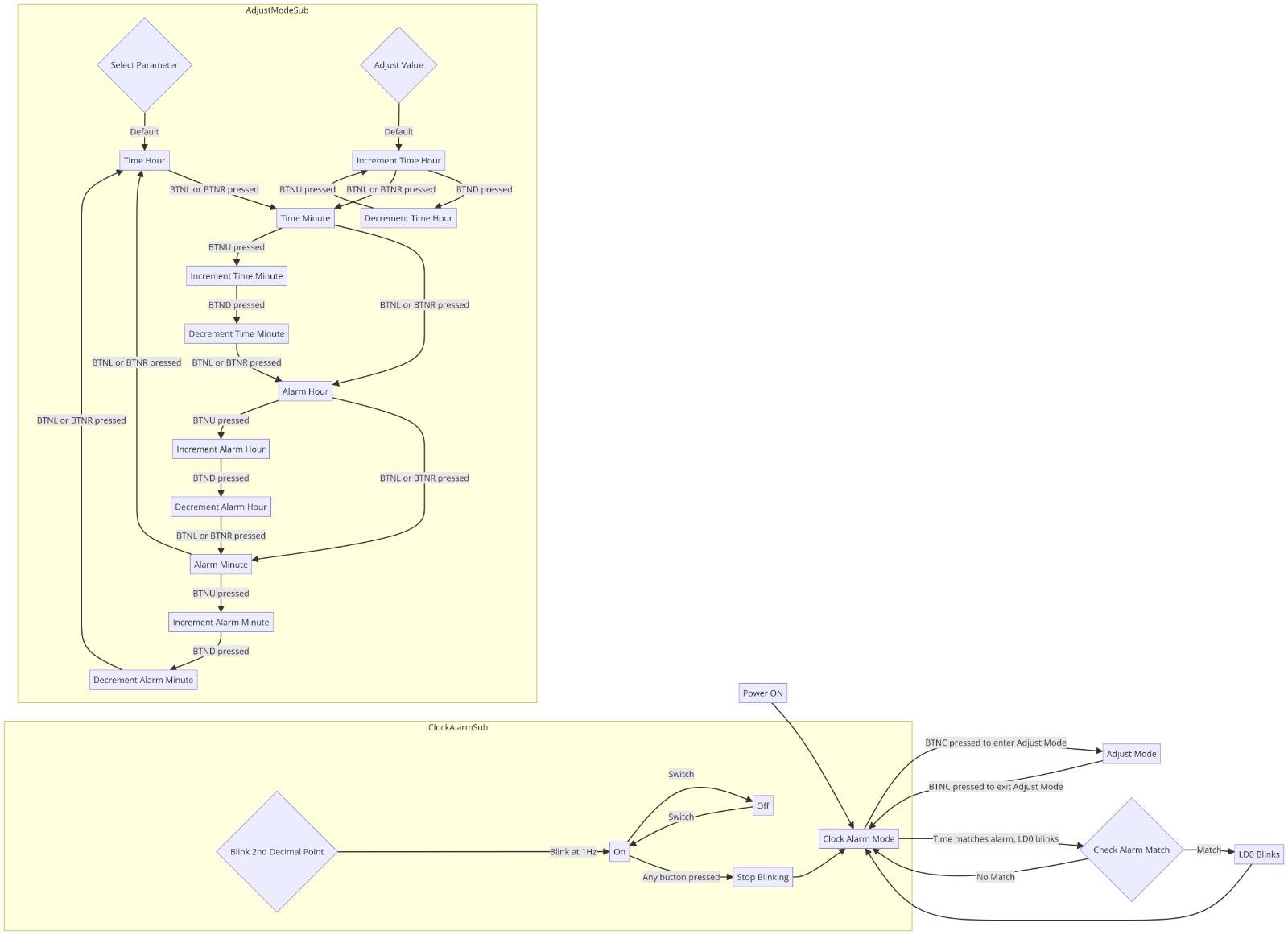
The system integrates pushbutton detectors to process button presses, enabling the transition between states. The clock divider modules generate the necessary clock frequencies. In *Clock\_State,* the clock operates normally, displaying the current time. Pressing the center button (BTNC) transitions 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 values.

The 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.

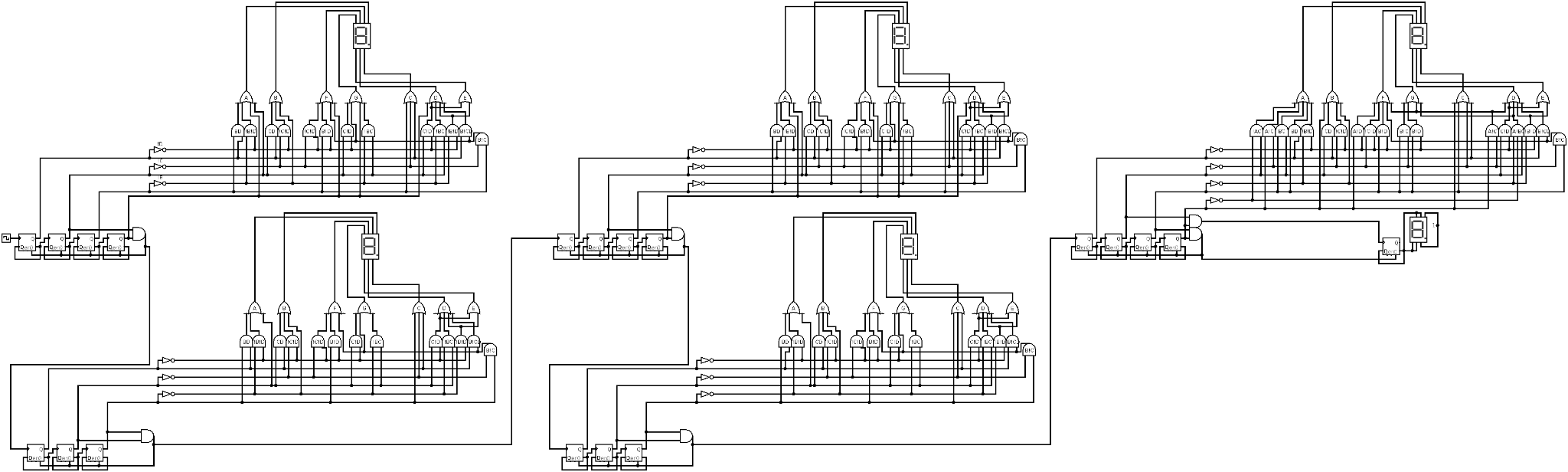
**Final\_block\_diagram:**



**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.

1. **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);

1. **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:**

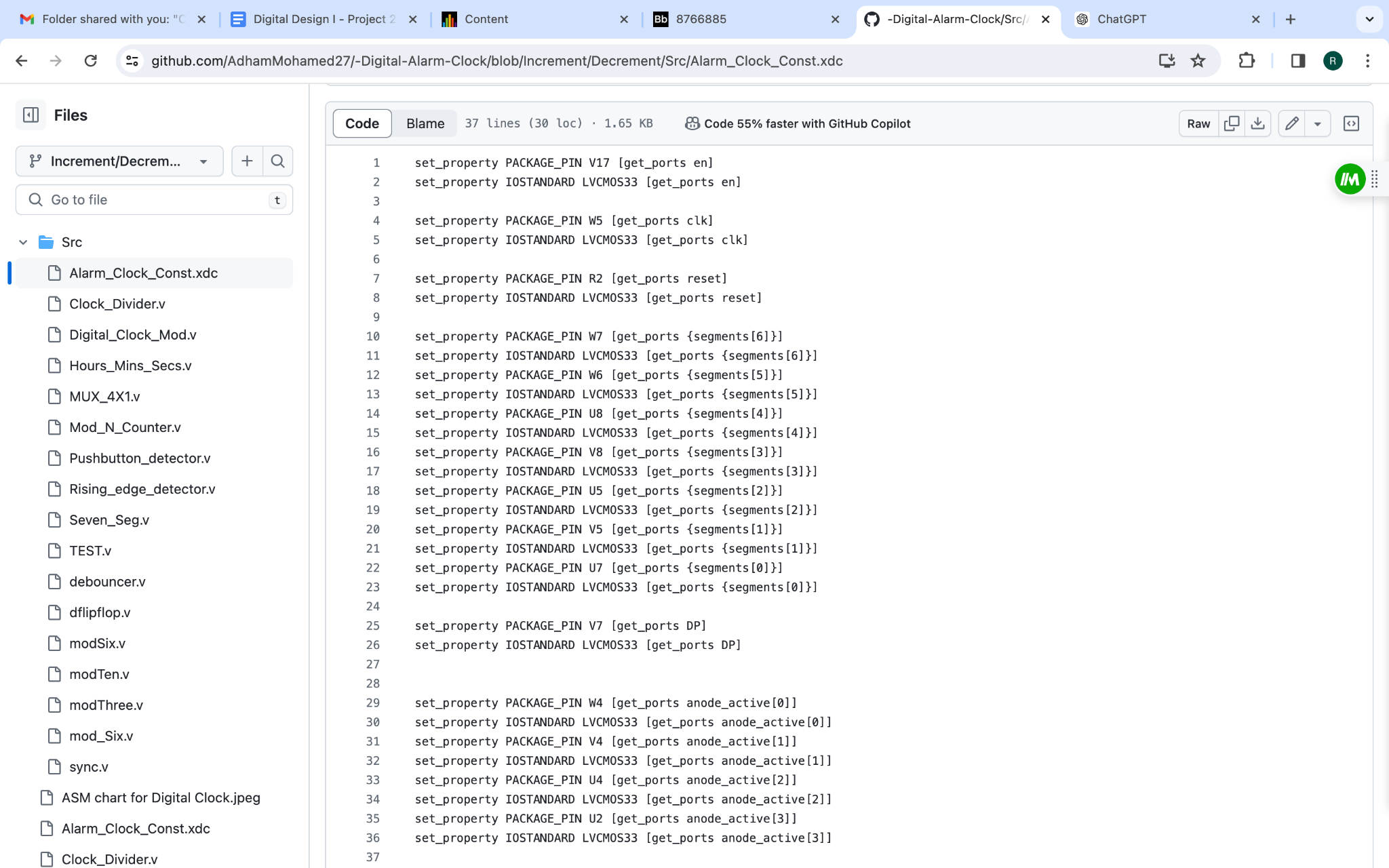
We began our evaluation of the Digital\_Clock\_Mod module by examining each state within the finite state machine individually to ensure they functioned correctly when isolated. We then tested how well key components, such as the clock divider and pushbutton detectors, worked together, ensuring they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

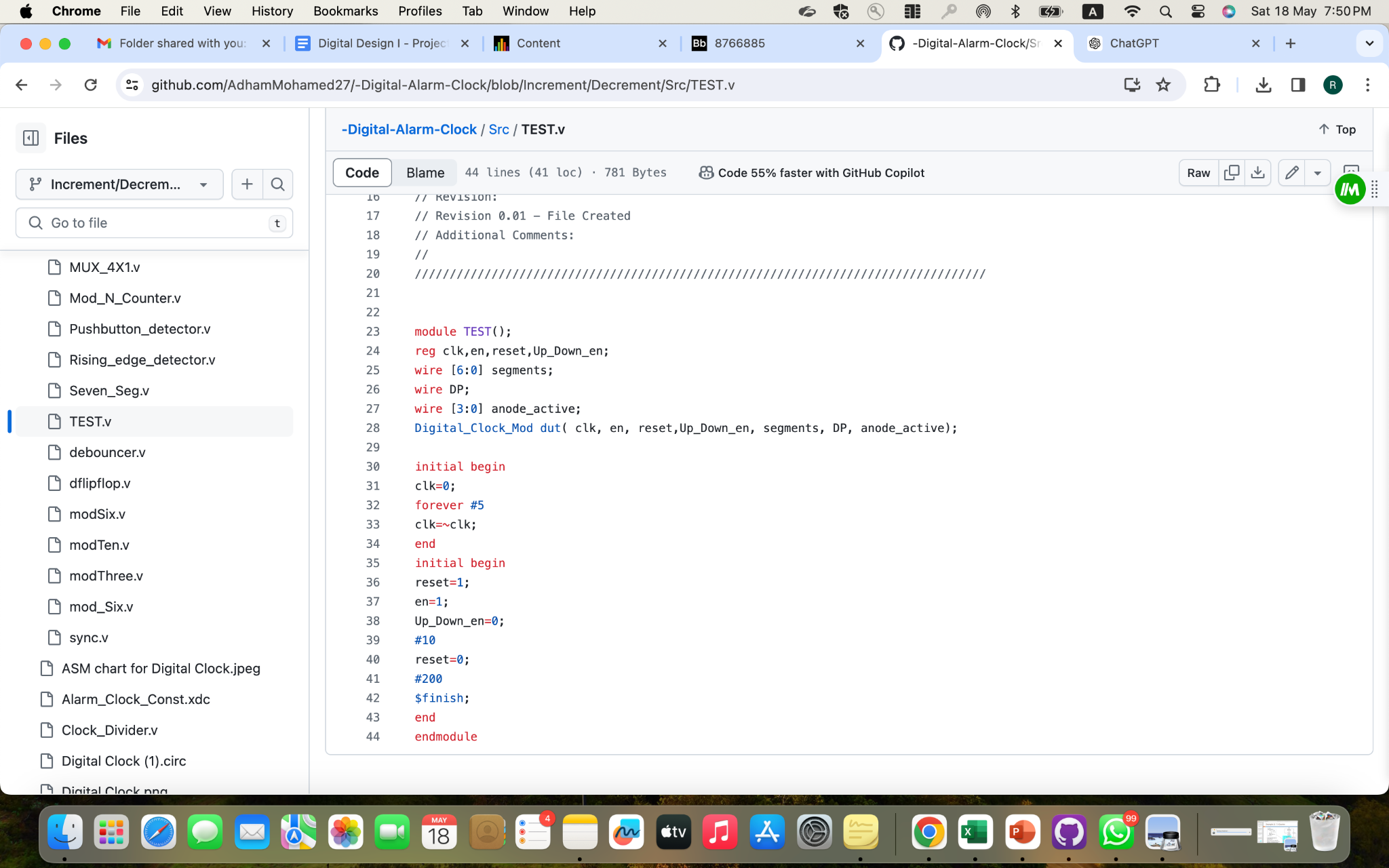
We thoroughly tested all possible button press scenarios to confirm that each button action resulted in the correct state change, simulating real-world user interactions. We also focused on the overall operation of the clock, particularly its accuracy and responsiveness to time adjustments, by running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



**TestBench** 

**Buzzer:**

assign buzzer = (state == AlarmMod) ? clk\_buzz:0;

**Contributions:**

Adham Mohamed:

Worked on the code, debugging, diagrams, and the report.

Abdallah Afifi:

Worked on the code, debugging, diagrams, and the report.

Jana Fadl:

Worked on the report and sat with the group during debugging, diagrams and coding.

Rodayna Mamdouh:

Worked on the report and sat with the group during debugging, diagrams and coding.