Alarm Clock Report

Clock Mode

sevenSeg module:

The sevenSeg module is crucial for our digital clock display. It manages the lighting of segments in a 7-segment display based on the input values. This implementation covers displaying digits from 0 to 9, achieved by determining which segments should be illuminated for each digit through FPGA segments analysis [refer to appendix 1&2]. In addition, it analyzes its inputs in order to pick which of the four digits to light up.

The module functions as two decoders:
1. **2x4 Decoder**: This decoder selects which of the four digits to light up in a multiplexed display. For example, if we want the third digit to be active, the anode active value would be `1101` since it works in active low conditions.
2. **4x7 Decoder**: This decoder receives a 4-bit input (a digit from 0-9) and translates it into 7 bits to control the seven segments. Each bit corresponds to a segment of the display, and both the anode and segment activations work in active low conditions (a bit value of 0 lights up the segment).

ClockDivider Module:

This module is responsible for handling frequency adjustments within the project. It receives an input clock (`clk`) and a reset signal (`rst`), and it outputs a modified clock signal (`clk_out`) at a desired frequency. The module uses the formula fout=fin/2n where n is the function parameter. This functionality is crucial in our project, as we require different clock frequencies for different components. For example, the main clock operates at 1Hz, while the ring counter operates at 200Hz, all derived from the FPGA's default frequency of 100MHz.

SPModuloCounter:

This module implements a modified modulo counter operating with clock (`clk`), reset (`reset`), enable (`en`), count down (`count_down`), and count up (`count_up`) signals. It takes two parameters: `n`, setting the maximum count value in decimal, and `x`, setting the counter's width

in bits. The counter has three modes: normal counting, count up, and count down. When `reset` is asserted, the counter resets to zero. If `enable` is active, the counter increments on the clock's positive edge in normal mode, wraps to zero if the maximum count (`n-1`) is reached, and decrements or increments based on (`count_down`) or `(count_up`) signals. This design ensures flexible and versatile counting, for digital systems requiring sequential operations, such as clocks and timers.

3x8 decoder

It serves as a bridge between user-triggered increment/decrement signals and adjusting the selected state. The 3-bit input determines the active state, continuously updated by the FSM. The decoder also takes `btnu` (button up) and `btnd` (button down) as inputs to connect to the active state. It has an 8-bit output, with two bits dedicated to each state of the clock: adjust time hour, adjust time minute, adjust alarm hour, and adjust alarm minute. The decoder checks which state is active among the four and connects it to the up and down buttons, allowing the user to adjust values in that state. If the system is in clock mode, the 8-bit output is all zeros, as no adjustments are made in that state even if the user clicked on btnu or btnd..

State Selection:

- 3'b000: Adjust time hour with `btnu` and `btnd`.
- 3'b001: Adjust time minute with `btnu` and `btnd`.
- 3'b010: Adjust alarm hour with `btnu` and `btnd`.
- 3'b011: Adjust alarm minute with `btnu` and `btnd`.
- 3'b100: Clock mode, no adjustments, output is zero.

In our implementation, we use a 2-bit binary counter and a 2x4 decoder as a ring counter, which plays a crucial role in determining which digit to display at any given time, as our design follows a multiplexed display approach to conserve energy. The 2-bit binary counter is driven by a 200Hz clock, provided by a clock divider. Although the maximum flicker rate detectable by the human eye is around 0.009 seconds (approximately 90Hz), we observed flickering at this frequency during implementation. Therefore, we increased the frequency to 200Hz to ensure a flicker-free display.

The ring counter's output acts as a selector for a 4x1 multiplexer (mux), where each input is a 4-bit value representing a digit. This digit is then processed by the seven-segment module to display its corresponding segments. Additionally, since we manage both clock and alarm displays, we use two 2x1 multiplexers. One mux selects between `segments_out_alarm` and `segments_out_clock`, while the other selects between `anode_active_alarm` and `anode_active_clock`. Both multiplexers share the same selector(display_selector), ensuring that either the alarm segments and anode or the clock segments and anode are displayed simultaneously. This setup allows for efficient and clear multiplexed display control, ensuring the correct segments and anodes are activated for either the clock or alarm display based on the selected mode.

Adjust Mode

Alarm/clock module

2 instances of the clock module was used however some changes was made to the original digital_clock module, the functionality of a press button count up and press button count down was added to assist with the functionality of the alarm. Since one clock counter represents the alarm, this clock counter takes and enable of zero and only works with the count up and count down buttons during its appropriate state. The other clock counter takes both enable when its in the counting state outside the adjust mode and zero otherwise. Moreover, an additional functionality of adding a load to the clock was implemented but wasn’t used in the end which prompted its removal from the final submision

Multiplexed clocks were used with the second clock associated with time where 1Hz clock cycle was used when its enable was zero or in other words when it was outside the adjust mode.When it enters adjust mode the multiplexer changes it so that the clock counter works with a 200Hz clock so that its synched with the push buttons.

FSM and Buttons

Module of main pushbutton file:

In the push button there are several engraved modules that are used in order to allow the push button to work. First and foremost comes the clock divider, which as previously mentioned that it is used for the sake of lowering the frequency. The debouncer is then used for the sake of filtering any noise that is coming with the transition X. Usually, this noise comes when someone presses the mechanical button or when someone releases it.

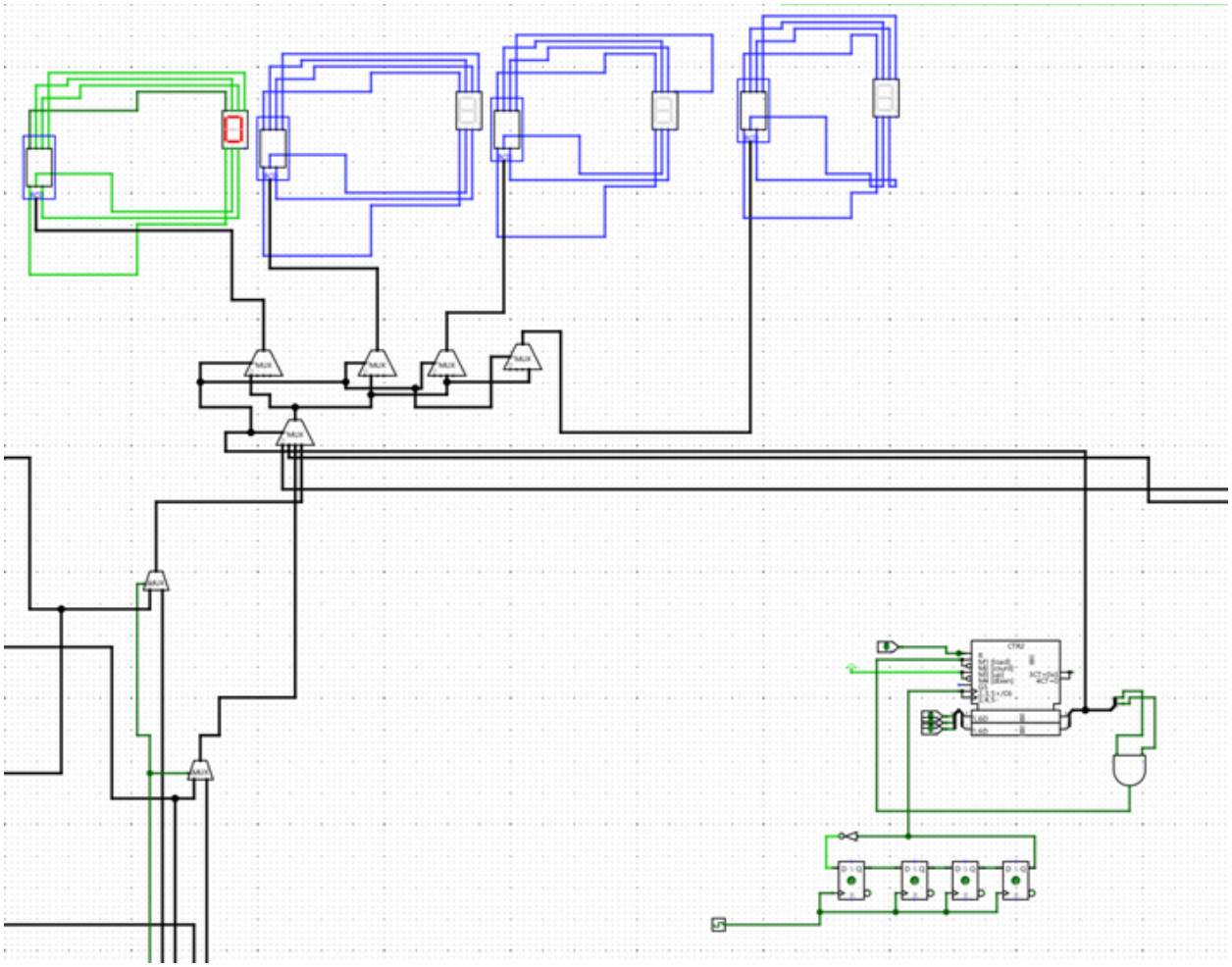
Logisim

Multiplexed seven-segment display:

This section covers the multiplexed display of the four digits representing minutes and hours (both tens and units places).

The system includes a modulo-4 counter that cycles through four states, corresponding to the four inputs for a 4x1 multiplexer (MUX). The 4x1 MUX receives four inputs (each 4 bits) from the counters and the adjust mode. The select signal for the MUX is provided by the modulo-4 counter.

The MUX selects one of the four digits and routes it to one of four different MUXes. Each of these four MUXes corresponds to a specific seven-segment display. The select signal (from the modulo-4 counter) determines which seven-segment display will show the selected digit.



The current time matches the set alarm time case:

When the current time matches the set alarm time the LED blinks. This is done through a quality comparator which it takes input from the Adjust Mode black box and the other from the digit counter.

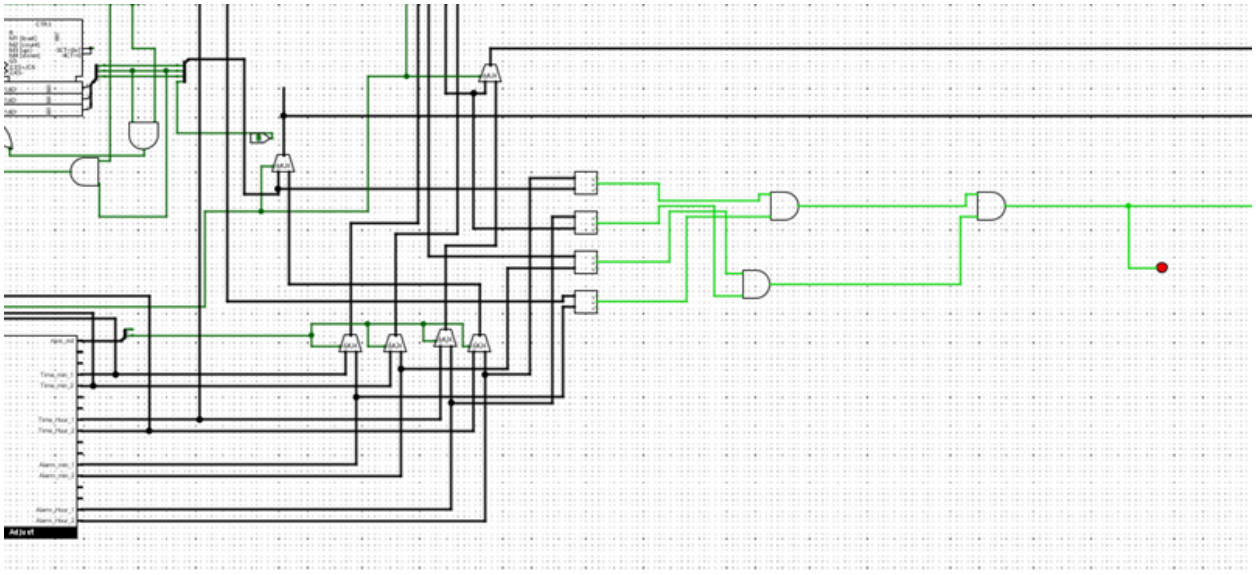
So, it checks if

Alarm min_units == Clock min_units

Alarm min_tens == Clock min_tens

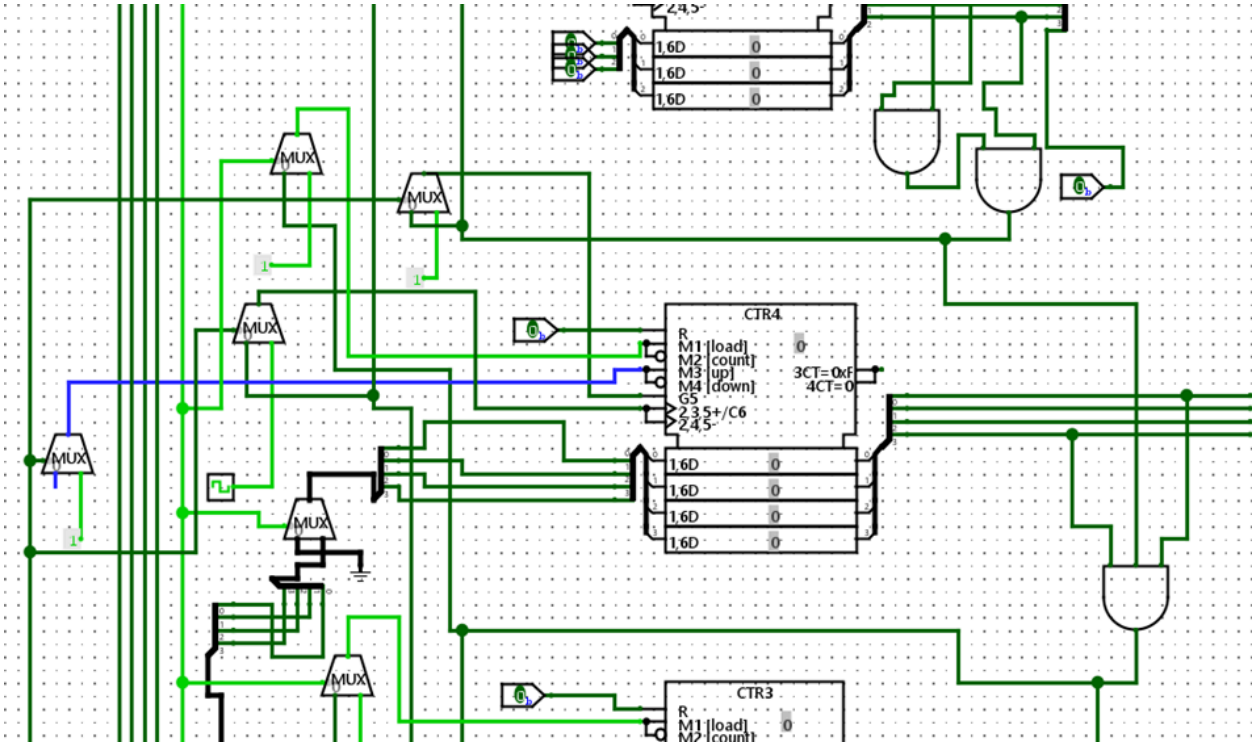
Alarm hour_units == Clock hour_units

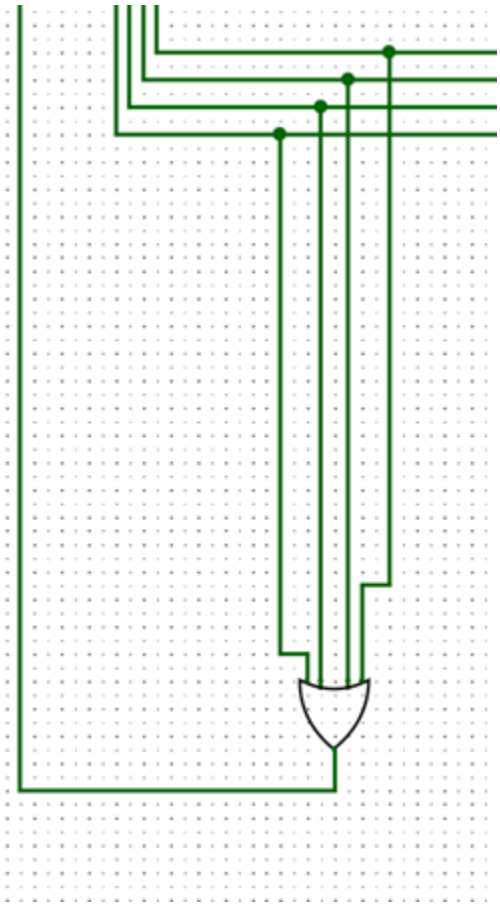
Alarm hour_tens == Clock hour_tens



It uses 3 AND gates to check if all these checks are true, then the LED will blink and the clock will stop.

Pressing any button will increment the clock's minutes units, causing it to stop blinking and resume ticking. This is achieved by connecting the four buttons to an OR gate, whose output is connected to the MUX select line. This setup loads a 1 into the count-up signal of the counter and also connects the clock to the counter, enabling it to count up successfully.


MUX
MUX
MUX
MUX
MUX
MUX
CTR4
R
M1 [load]
M2 [count]
M3 [up]
M4 [down]
G5
2,3,5+/C6
2,4,5-
3CT=0xF
4CT=0
1,6D
1,6D
1,6D
1,6D
CTR3
R
M1 [load]

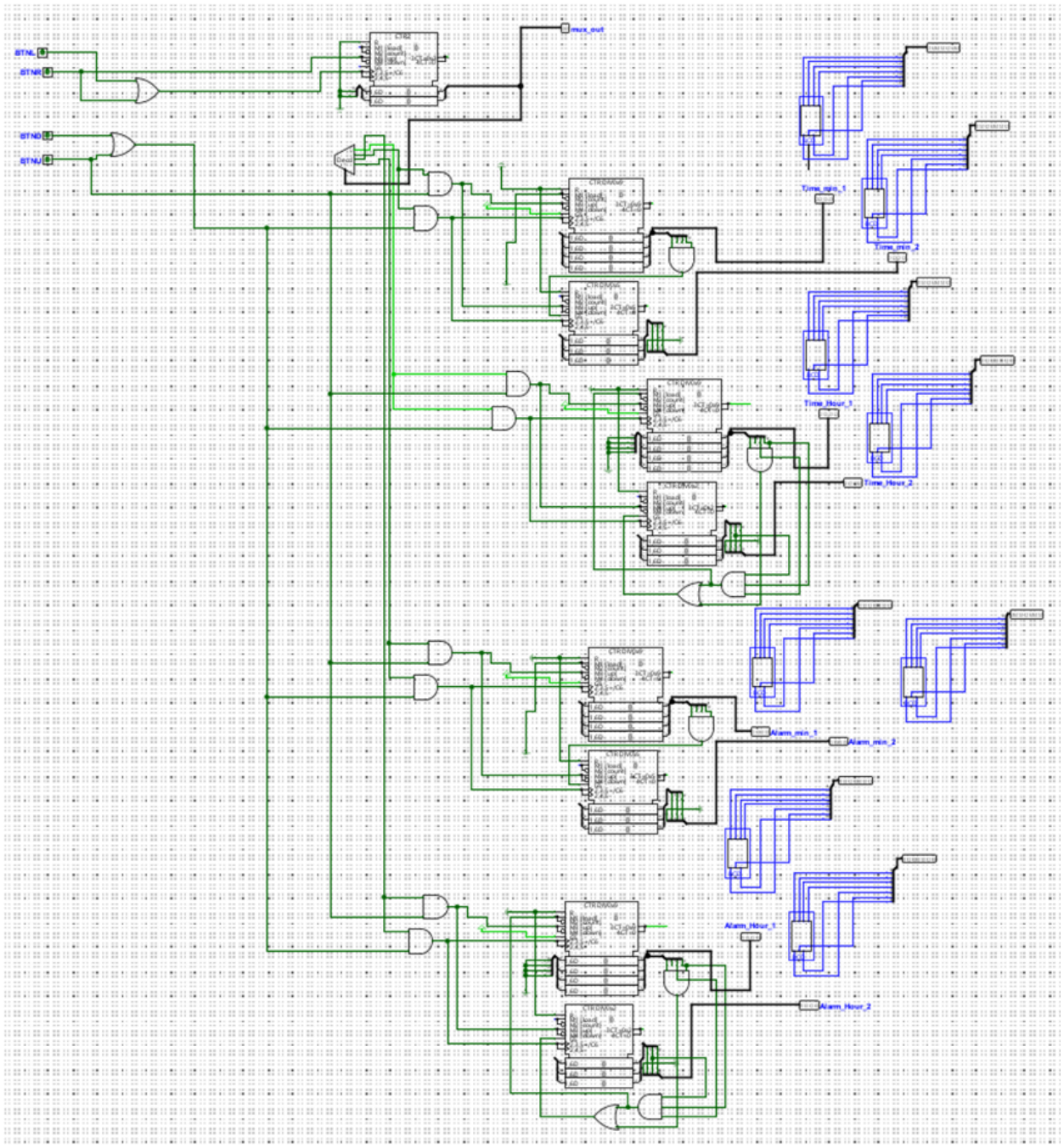


Adjust mode black box:

- Pressing BTNC exits the adjust mode to the clock/alarm mode.

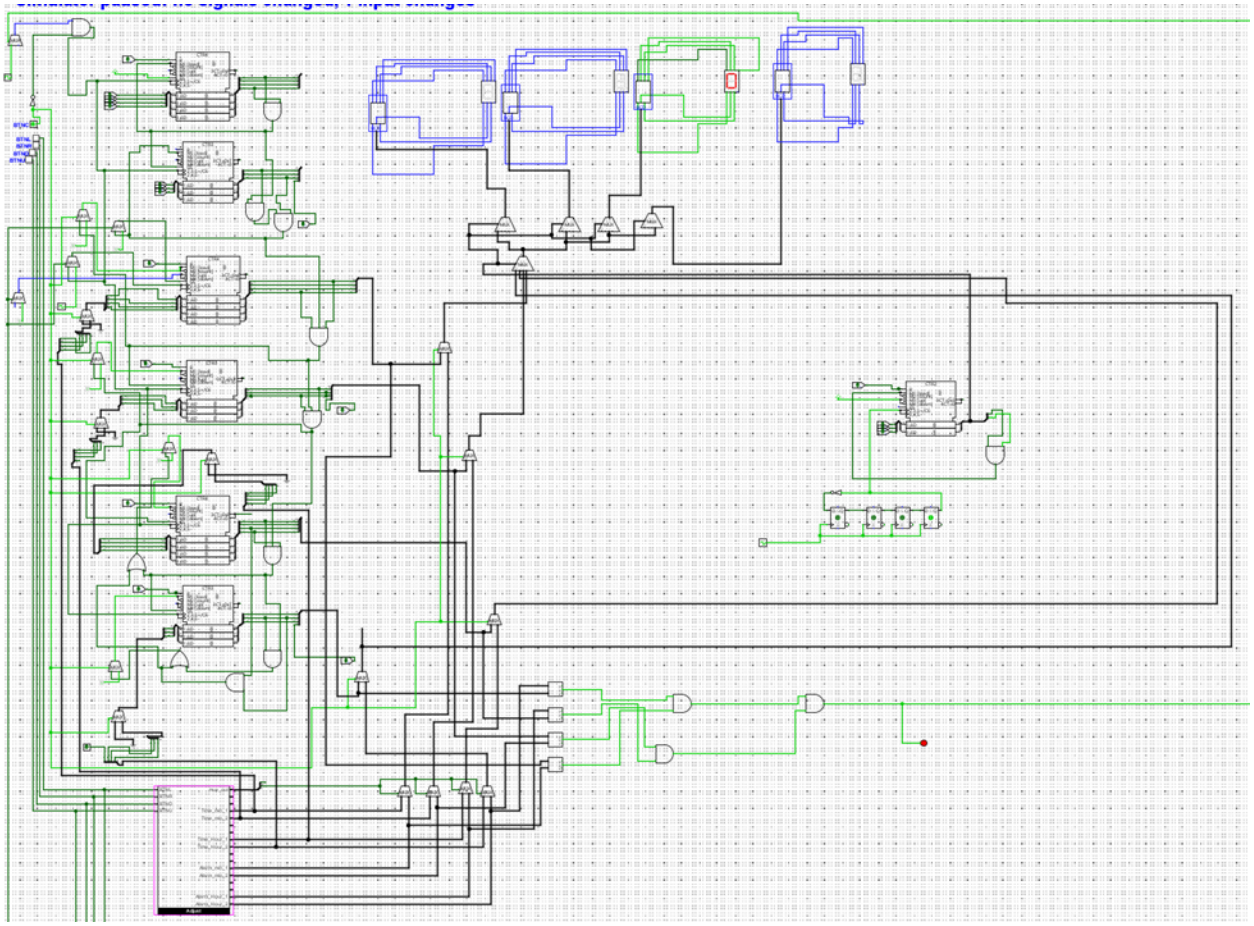
- Pressing BTNL or BTNR selects what to adjust in order as follows “time hour”, “time minute”, “alarm hour”, and “alarm minute” which is done through a decoder.

- Pressing BTNU increments the selected parameter through the counter and pressing BTND decrements the selected parameter.



Final look:

The adjust mode outputs is MUXed to select (using the buttons BTNL and BTNR) which mode to be outputted (Alarm or clock) then the output is MUXed with the actual counters of the 4 digits and its select is BTNC to choose whether we are in the adjust mode or in the normal clock.



Mostafa's Contribution

ASM chart

Following the DP/CU algorithm provided by Malek, Mostafa designed an ASM chart representing the states and the movement from one state to another.

Logiscm

Implemented the blackboc mentioned above which takes in buttons as an input and outputs streams of bits representing the clock time and the alarm time as well as a selector highlighting the current state.

3x8 Binary Decoder

Deciding what digit of which lock to adjust using btnd and btnu relies on the current state of the system however the clocks takes 4 inputs regardless of what happens. After contemplating about this problem a decoder seemed the only viable option. Mostafa with the help of Malek implemented a 3x8 Binary decoder that takes a selector of 3 bits representing the current state of the system and 2 inputs namely btnu and btnd and assigns the inputs to the clock accordingly.

FSM

Building on what Habiba and Malek wrote, Mostafa built up an FSM including all states with if statements expressing what happens when every button is pressed for every state to accommodate for state transitioning. Also, it was ensured that transitions was associated with correct changes in selectors to accommodate for the seven segments display and the selector of the 3x8 binary decoder. Moreover, the blinding 2nd decimal was linked to the fsm by adjusting the digital clock module to tie its functionality to the enable.

Digital_clock added functionalities

Developed extra functionalities to work with the digital clock namely the count up. This was done by first incorporating the count up and down functionalities to the modulo counters then simply this was passed down as 4 inputs relating to up hour, up min, down hour, down min for

every clock. The conditions for resetting and counting was then adjusted to accommodate for the new functionality resulting in a seamless adjustable clock

Buzzer

Incorporated the Buzzer functionality using an active buzzer directly related to LD0 as it buzzes with the same frequency as the LD0 during the alarm state, it stops buzzing and the relation to LD0 is broken after we get out of the alarm state

Saif's Contribution

Algorithm:

Saif edited the algorithm used to implement the diagrams and the Algorithmic State Machine (ASM) chart.

Verilog Code Contributions:

1. Alarm State in FSM Module:

Saif created an Alarm state within the finite state machine (FSM) module. This handles the activation and deactivation of the alarm.

2. **Counter Modification:**

He further enhanced the counter's functionality by editing it to incorporate a load signal and a load value input. This modification allows the counter to be more flexible and responsive.

3. **Seven-Segment Display with DP:**

Saif successfully implemented the Seven-Segment display module, a critical component for user interaction. His work ensured that the display correctly interprets and shows the relevant numerical values.

4. **Multiplexed Seven-Segment Display:**

Moreover, Saif implemented the multiplexed seven-segment display feature in the FSM module, which allows blinking.

5. **Logic of FSM States:**

Saif also contributed to the logic of the FSM states by commenting and debugging the code.

Logisim Contributions:

1. Hour Units and Tens Counters, BCD, and Seven-Segment Display:

In Logisim, Saif implemented the hour units and tens counters, along with the Binary-Coded Decimal (BCD) conversion and the seven-segment display functionality. These elements are essential for the accurate representation of time, ensuring that the clock displays the correct hours.

2. Multiplexed Seven-Segment Display:

He also implemented a module 4 counter connected to 4x1 4-bits MUX that makes the 4 digits blink.

3. Alarm Functionality:

Saif designed the clock to stop once it reaches the alarm time, with the LED blinking as a visual alert. Additionally, he made the clock continue ticking and the LED stop blinking when any button is pressed.

4. Loading the adjusted clock:

By enabling the clock to enter the adjust mode through click BTNC button, he made it possible for users to modify the clock easily. Once the adjustments are made, the new clock value is loaded into the counters, ensuring that the system reflects the updated time accurately when the BTNC is off.

Habiba's Contribution:

1. **Algorithm:**

Just like it has been taught to us in the lecture, before starting any DP or ASM chart one should have an algorithm that can be considered as the DP or the ASM chart. I did two versions of the algorithm, one that is mere pseudocode and the other one is hardcore C++ just for the sake of being able to look at the different cases and how the terminal outputs them. As previously mentioned, Saif did edit on it.

2. **Logisim:**

For the logisim in the first submission, it was separated into three parts. A part for the minutes and seconds, the second part for the addition of the hours and the third part for the muliplexed Seven Segment Display. The first part was handed to Habiba (which was the minutes and seconds). The difficulty that I faced was altering the frequency of the clock in order to try to use the D flip flops but I couldn't. When I tried to use 7 D Flip Flops by assuming that the clock had 128Hz, I found that the seven segment display moved at a more exaggerated slow pace. Unfortunately, I removed the D FF and used the clock directly to the counter as an input and behaved as a normal clock.

3. **Clock-Mode:**

The files of the seven seg, the modulo 6 and 10 and binary counter, the clock divider were Habiba's but they faced major issues that both Malek and Habiba solved (these were previously discussed above in the prerequisites part). However for the case of reminder it mainly included

1.anode active issue [minutes was mixed with seconds]

2. counter issue (the minutes reach 8 then renews)

3.wrong clock divider (it makes the seven seg increment at a very slow rate) when taking the write frequency

4. **Decimal Point blinking:**

The addition of the decimal point was not an issue. But the blinking of the decimal point was the one who faced the problem. The problem was a consequence of the clock divider issue. Instead of the decimal point blinking at a moderately fast rate, it usually takes a very long period of time

to turn off when it's on and vice versa. After altering the code of the clock divider, I was able to put the right frequency to make it blink in the correct frequency.

5. **BTNC and Enable:**

In Habiba's original implementation regarding the 7 seg, the rst was a toggle switch. Then we moved the implementation of the enable as a toggle. There was an issue Habiba resolved by putting the enable as an output parameter. Then moving onto the BTNC, Malek took part in the implementation but there was a bug(the BTNC was not working just like the enable) Habiba debugged this part after debugging the part that the clock was not incrementing after we added the FSM (this was solved by entering the right frequency to the clock)

Malek's Contribution:

DP & CU:

Implemented the DP & CU. Following Lab 7's work in the digital design lab I learned how to deal with counters and multiplexed displays. From there, I built on the additional requirements in order to implement a digital clock with alarm mode and adjusting time mode. Moreover, the use of the pseudocode of my colleagues served as the foundation. The design was done on the website drawio in order to ensure clear visibility and well done structure.

Clock-Mode:

As my colleague Habiba discussed above we faced some issues in clock mode but after some debugging we were able to implement a 24 hour system clock using 6 counters. Along with a ring counter, multiplexed display, clock dividers with suitable frequencies, and making it compatible with two clocks.

Integrated FSM:

I contributed to the adjust mode by designing and implementing the integrated Finite State Machine (FSM). After drawing the FSM diagram to outline the six different states, I translated this design into functional code. My responsibilities included ensuring that the buttons correctly adjusted the system to the appropriate states and that the corresponding LED lights accurately

reflected these states. This involved extensive testing and debugging to guarantee seamless state transitions and correct visual indicators.

Alarm Blink:

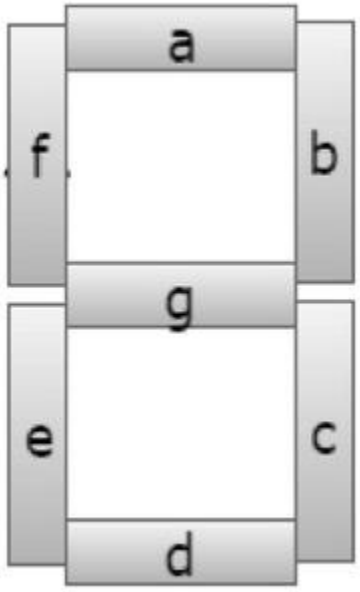
I continued from my colleague Mostafa's work by implementing the logic for the LDO blink with alarm feature. Building upon his work in the alarm state, I integrated the necessary logic to control the LED light to blink at a frequency of 1Hz. This involved synchronizing the blinking pattern with the alarm state's timing and ensuring that the LED toggles on and off at the desired rate.

Displaying Two Clocks:

Initially, Mostafa and I planned to manage the display of the two clocks using two 4x1 multiplexers (mux). However, upon Mostafa's suggestion, we chose to implement a 3x8 decoder along with two 2x1 multiplexers. This approach provided better handling of the adjust states, offering increased flexibility in managing the display selections. By integrating the 3x8 decoder, we were able to achieve more control in incrementing and decrementing in the adjust states.

Appendix:

1.



2.

	Input				Output						
Decimal Digit	X3	X2	X1	X0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0