Aly Elaswad (900225517) - Islam Abdeen (900225835)- Ismail Sabry (900222002)

Project Report

Digital Alarm Clock

Digital Design 1

The American University in Cairo

Presented to: Dr. Mohamed Shalan

Department of Computer Science and Engineering

20 May 2024

Outline:

In this project, we designed and implemented a digital alarm clock using the BASYS-3 FPGA board. The project encompasses various stages of digital design, including the creation of block diagrams, ASM charts, and the development of Verilog code using Vivado to drive the hardware components. The primary objective is to utilize the BASYS3 board's peripherals, including the LEDs, 7-segment display, and push buttons, to create a user-friendly and efficient alarm clock system, as well as apply the knowledge that we gained throughout the course to a real hands-on project to test our understanding.

Project Description:

The digital alarm clock uses the following BASYS3 board peripherals:

LEDS:

LD0: blinks when alarm is buzzing, is on while in adjust mode
LD12: To adjust the time hour parameter
LD13: To adjust the time minutes parameter
LD14: To adjust the alarm hour parameter
LD15: To adjust the alarm minutes parameter

Buttons:

BTNU: Increments the selected parameter
BTND: Decrements the selected parameter
BTNR: Switches to the next parameter to the right
BTNL: Switches to the next parameter to the left
BTNC: Changes mode from clock to adjust mode or vice versa

7-Segment Display:

Displays either the alarm clock or the normal clock based on the selected parameter in a 24-hour clock format
The decimal point represents the seconds, and when on means that the clock is on and enabled.

Display Configuration

- 7-Segment Display:
    - The two leftmost digits will display hours.
    - The two rightmost digits will display minutes.

Operating Modes

The alarm clock will have two modes of operation:

1. Clock/Alarm Mode (Default Mode)
    - LED LD0: OFF
    - Second Decimal Point: Blinks at a frequency of 1Hz
    - Alarm Activation: When the current time matches the set alarm time, LD0 blinks. Pressing any button will stop the blinking.
2. Adjust Mode
    - LED LD0: ON
    - Second Decimal Point: Does not blink
    - Parameter Selection:
        - Press BTNL or BTNR to cycle through the parameters: "time hour", "time minute", "alarm hour", and "alarm minute".
        - LEDs LD12, LD13, LD14, and LD15 indicate the current parameter being adjusted.
        - Example: When adjusting "alarm hour", LD14 is ON; pressing BTNL will
        - switch to "time minute", turning LD13 ON and LD14 OFF.

The contributions of each member:

Aly: Logism initial counters (MUX Display) Alarm and clock, increment, ⅓ data path. Added hours counters. Increment and decrement on Vivado.

Islam: Control unit, ⅓ of the data path, UP DOWN buttons synced with right and left buttons and counters. ½ of the ASM chart. FSM on Vivado, and aligning the LEDs to each state.

Ismail: Decrement in Logisim, right-left buttons to choose, ⅓ of the data path. ½ of Asm chart. In Vivado, adjusted enablers of the counters after increment and decrement , Alarm mode.

Done by All: Buzzer on Logisim, fixed errors on Logisim. Initial counters were found in the lab. Buzzer and decimal point on Vivado.

The challenges we faced:

We faced several problems in Logisim first and the most important one was that the file was heavy and large so any change we would make wouldn’t be saved as a .circ file but saved as a .BAK file. This didn’t allow us to save which took much time to retry the circuit to be saved by trying our Luck each time.

The second problem we faced in Logisim was that when we tried to enable using a 2x1 mux containing a push button and the CLK, the counters did not operate regularly. This was fixed but using muxes of power and trial and error method.

We also faced several problems in the implementation of the code on the FPGA. The first one was that enables took us a lot to implement properly on 6 counters and how to react when it is in adjust mode whether it will increment or decrement.

The second problem we faced in the implementation was that when the clock is in alarm mode. The clock stops counting which means it stays in the alarm state whether minutes or hours infinitely. This was fixed by adjusting the FSM to work in this way.

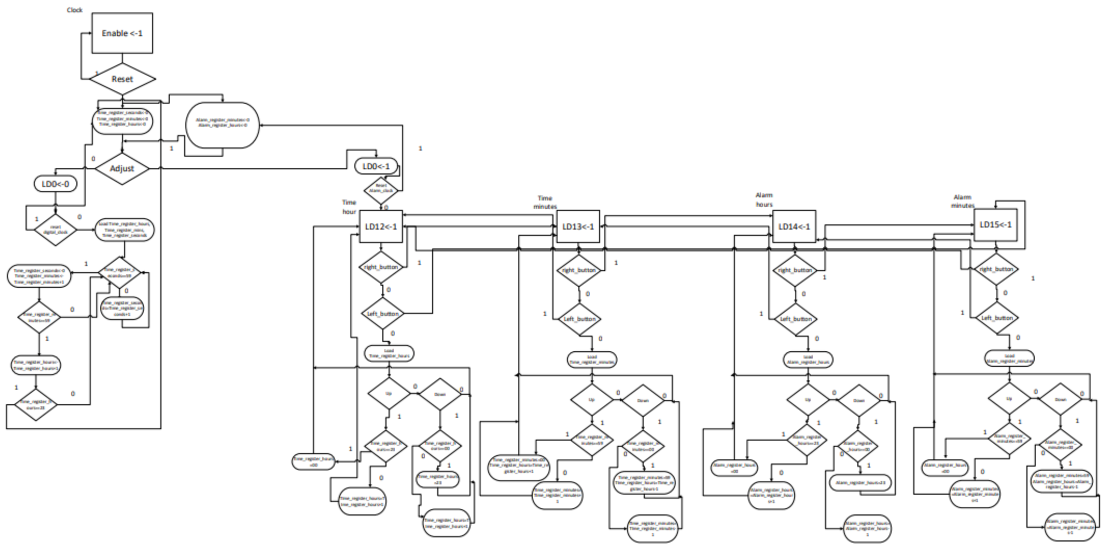
Also we faced an issue in implementing that any pushbutton pressed stops the alarm buzzer or led light. We tried to do it firstly outside the states using conditions but it didn't work at all. After it went to the states we faced a problem that it doesn't go through a loop between clock and alarm time hour and alarm mode which was later corrected by our design of using a visited register.

Last problem we faced was that the registers did not save the data from one clock cycle to the other. We recognized this problem quite late when two different and independent (decimal point blinking and decrement), did not work together. If one of these cases were corrected the other doesnt work properly. This was corrected by adding the registers value assignment statements in each registers to ensure that data in registers does not corrupt and take random variables.

The First Milestone

ASM:

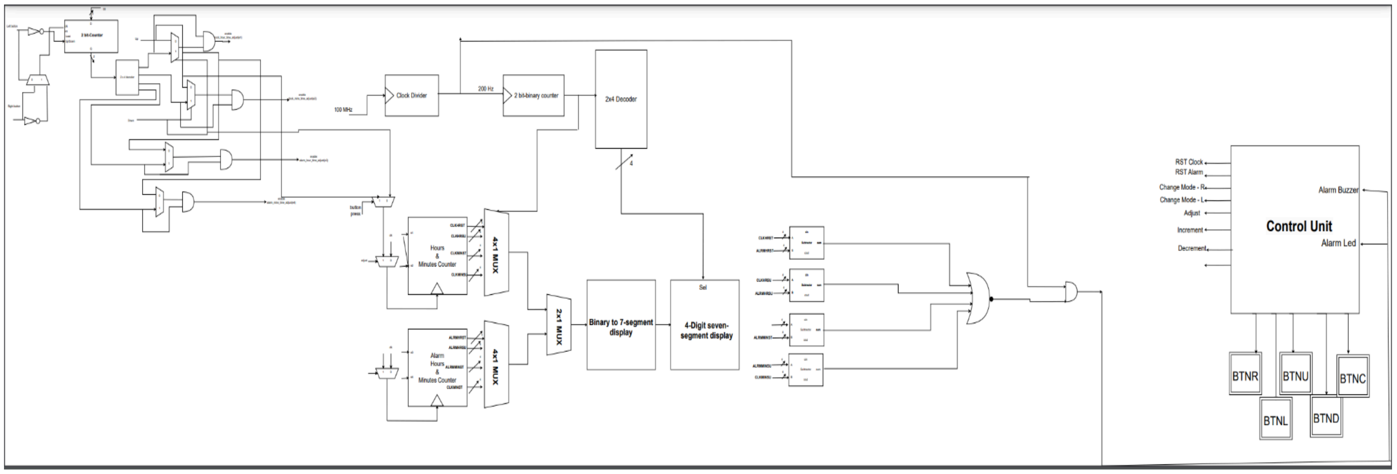
We started with designing the ASM for the Clock-Alarm project by first specifying the states which were five, including the Clock state, the Time hour state, the Time minute state, the Alarm hour state, and the Alarm minute state. We handled the clock mode where we had six counters: second units counter, second tens counter, minute units counter, minute tens counter, hour units counter, and hour tens counter. We included decision and output boxes to handle the clock logic. In the Adjust mode, we had four states: Time hour, Time minute, alarm hour, and alarm minute. There were 4 counters for the alarm which were, alarm minute units, alarm minute tens, alarm hour units, and alarm hour tens.



After completing the ASM, we started designing the block diagrams for the datapath and the control unit based on the ASM.

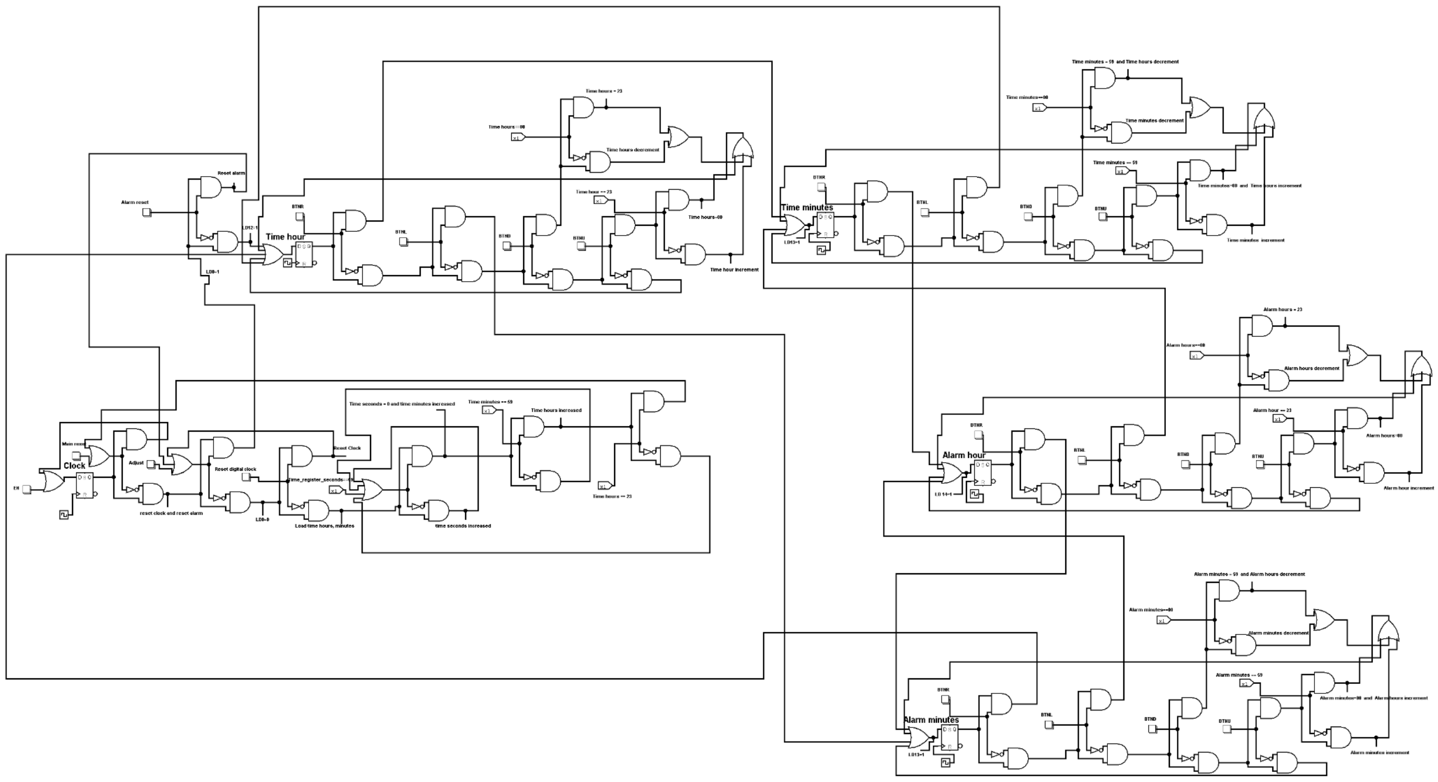
Datapath:

We used three clocks (the regular one, one for the 2x4 decoder and one for the second minute counter) and implemented the multiplexed display. We used subtractors for the Z flag that is used to indicate that the alarm count and the clock count are equal. The control unit’s signals were used in the datapath.



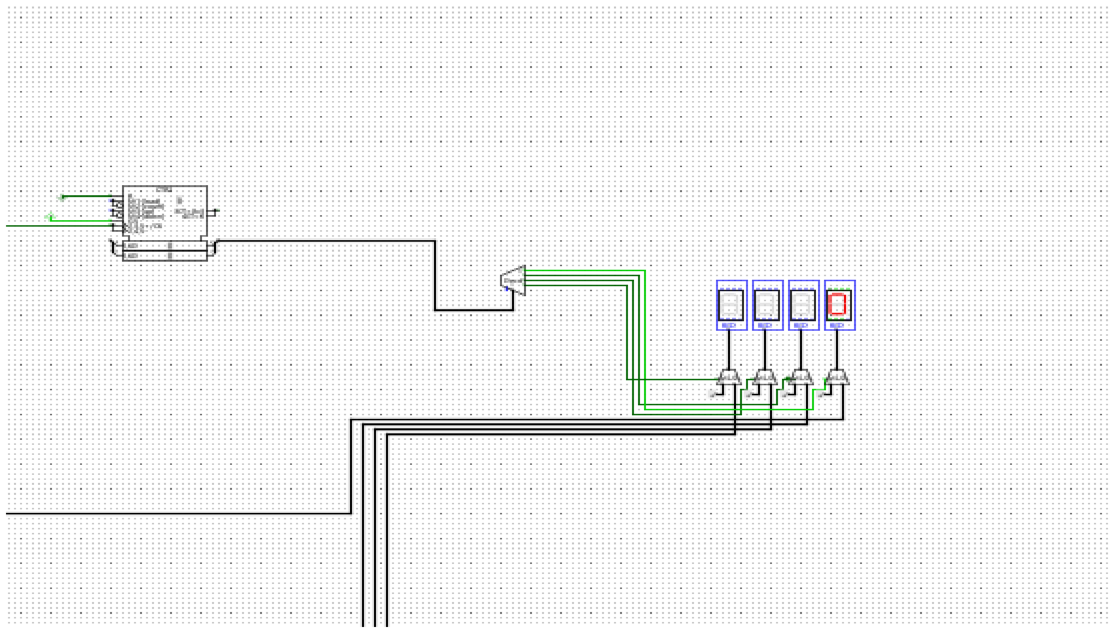
Control unit:

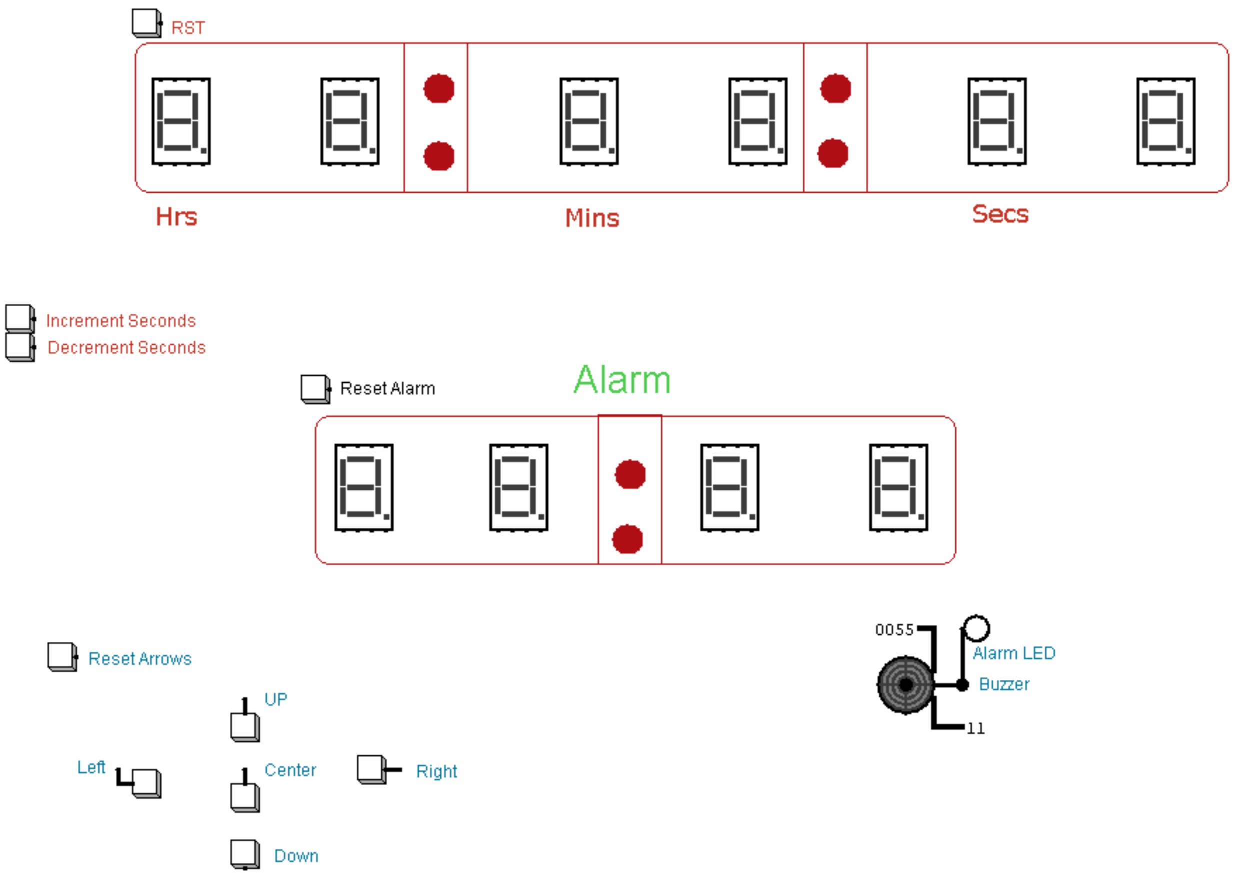
We, then, extracted the control unit from the ASM, designed the circuit, and included the signals that were necessary for the datapath and the states we used.



Logisim:

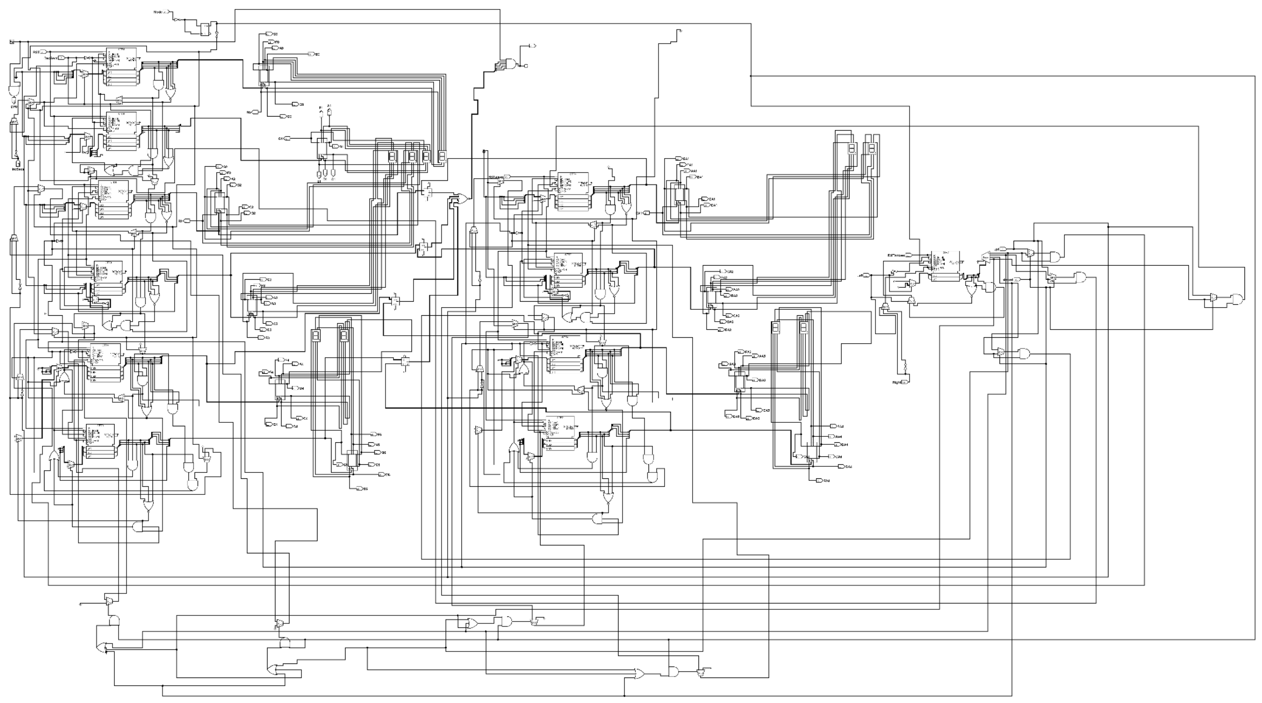
After completing the ASM, datapath, and control unit, we tested the circuit in logisim. We implemented the multiplexed display for the seconds-minutes-hours counter. Furthermore, we implemented a Clock-Alarm demo in which 6 counters were used for the clock and 4 for the alarm. We also implemented the up-and-down increment logic using a 2x4 decoder and MUXs. Finally, we displayed the alarm clock demo in a way that resembles an FPGA where buttons were used, and a buzzer was included.





Digital Design Project 2

Digital Alarm Clock



The Second Milestone:

Vivado:

For our code there were several functions used:

Debouncer: This module stabilizes the input signal from mechanical switches or push buttons. The debouncer makes any single stable and clean.

MUX4x1: This module selects the output depending on the combinational binary of the selection line. Each combination of the selection line (00, 01, 02, 03) corresponds to one of the four input signals.

Clock Divider: This module takes an input clock signal and divides its frequency to produce a slower clock signal. The division factor is typically a power of two, and it is used to generate different clock speeds required for various parts of the circuit.

Synchronizer: This module ensures that asynchronous input signals are properly synchronized with the systems clock cycle. It prevents stability issues by using several Dflipflops.

Rising Edge Detector: This module detects the rising edge of an input signal, which is the transition from a low state (0) to a high state (1).When a rising edge is detected, the module generates an output that will be used in the Push button.

Push button: Push button utilizes several functions all of them defined before-hand in the above statements. This component is a physical input device that, when pressed, completes an electrical circuit and sends a signal to the system. The push button is typically connected to a debouncer or a synchroniser with a rising edge detector to ensure a clean signal is sent to the other module which in this case is Digital Clock.

HoursMin counter: We improved the modules in the lab and added counters for the hours units and hours tens. This module contains the FSM of states and its inputs come from which buttons are pressed. This module focuses on which counters are enabled and when exactly. This also includes a condition for the buzzer and the decimal point of when to be on and when to be off. We gave the dp, the clock and the buzzer a frequency of 1 Hz.

Digital Clock: In digital clock we called the push button module five times. The five times represented the center, right, left, up and down buttons on the FPGA. This module relies heavily on the HoursMin counter Module. It gets the output from the counter module and then uses a mux to display the counters muxed and beside each other at the same time. In this part we used a counter controlled by a clock divider (200 Hz) and a seven segment display to be able reach what the requirement of the project wants. For the push buttons we gave them a frequency of (100 Hz)

Validation Activities:

Moving onto how we checked if what we have achieved was the same as the requirements specified by the project. Firstly, we increased the clock to make sure that it increases correctly and there is no error from going from 00:00 back to 00:00. We then made sure that the clock after being adjusted continues counting from the adjusted mode. By the moving of states on the LED lights we ensured that our FSM is conducted properly. We then incremented n+1 (depending on minute hours) times to see if it goes back to 0. Same goes for decrementing, we did this for all states and made sure that the alarm state is displayed correctly on the fpga. We then did the led light of the alarm first then we tested the buzzer to check if the alarm time is the same as the clock time. Every step going forward we checked the previous steps from before to make sure that we did not miss anything important.