THE AMERICAN UNIVERSITY IN CAIRO

الجامعة الأمريكية بالقاهرة

School of Sciences and Engineering

Department of Computer Science and Engineering

Spring 2024

CSCE 2301: Digital Design I

Project 02: Digital Alarm Clock

Dr. Mohamed Shalan

Submitted By:

Mark Kyrollos - 900211436
Aabed Elghadban - 900223106

Date: May 17, 2024

TABLE OF CONTENTS

GitHub Repository link: https://github.com/MarkKyrollos/Digital-Alarm-Clock

1. **Introduction**
2. **Description**
3. **Diagrams**
4. **Code Design**
5. **Implementation Obstacles**
6. **Validation Activities**
7. **Contribution**
8. **References**

1. Introduction

For our project, we implemented a digital alarm clock over multiple phases each having its own sequence of events. We started out by dividing our project into several milestones. For our first milestone, we first designed our digital alarm clock visually utilizing various block diagrams such as the ASM chart, the Data Path, and the Control Unit for our whole system. Afterwards, we implemented our digital clock circuit using the powerful Logisim software interface. Our work throughout using Logisim felt seamless due to the software's powerful tools and organization that help in designing every detail of the digital circuit, making circuit creation easier and quicker. After creating our general outline of how our project should look, we had to put our work in action by coding our actual digital alarm clock. Our main software interface for coding that we used was Vivado software, where we coded in Verilog programming language and programmed it onto our FPGA Basys-3 board.

2. Description

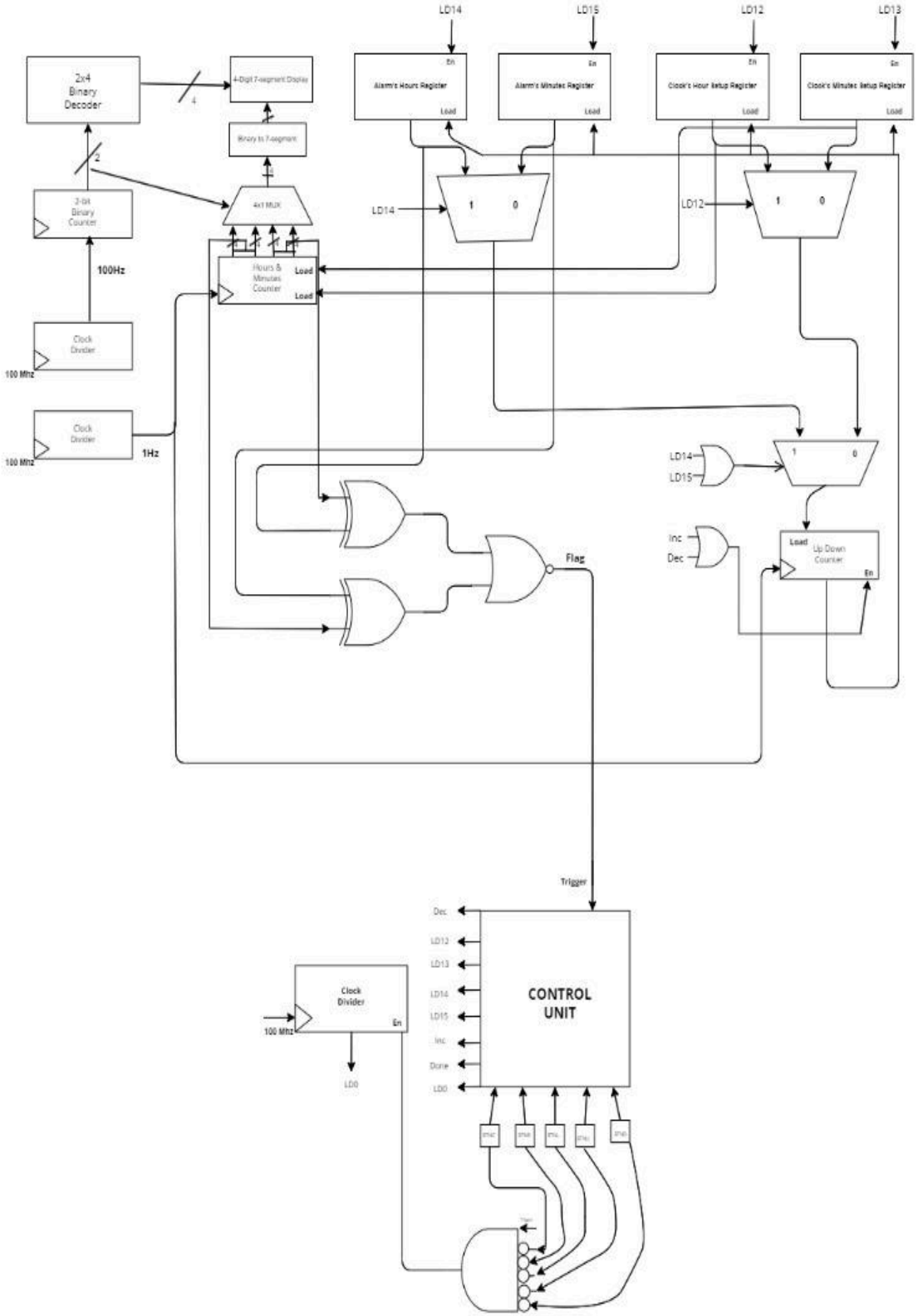
For our digital alarm clock, the only components we used on our FPGA Basys-3 board were the 5 LEDs (LD0, LD12, LD13, LD14, and LD15), the 7-segment display, the 5 push buttons (BTNC, BTNL, BTNR, BTNU, BTND) and reset and enable toggle switches. For the 7-segment display, it is used to display the hours (0-23) and minutes (0-59) according to the given situation. For our clock/alarm system, there are 2 modes the user gets to choose from. There is the clock mode, and the adjust mode. The user is able to switch between these two modes as many times as they desire by just pressing on the interactive push button (BTNC) in order to switch between these two modes.

For our clock/alarm mode, this is where the main clock is displayed on our 7-segment display, displaying the hours and minutes. While being in the clock/alarm mode, LD0 is constantly off until the clock time reaches the alarm time. Only then does LD0 start to blink at a constant 1 second until any of the 5 push buttons are pressed. When any of the push buttons are activated, LD0 stops blinking. However, I did not stop here. I added a buzzer component that would ring at the same 1Hz rate as our LD0 when the alarm time is reached. This makes our project more realistic and increases user immersion for our system as a whole. This is because a typical real-world alarm is supposed to ring or create any kind of sound when an alarm is triggered, whether it is a physical clock or a digital built-in clock on our cellphones. In addition to the time digits displayed on the 7-segment display, I added a blinking decimal point that oscillates every 1Hz to simulate our clock and distinguish the hours time from the minutes time.

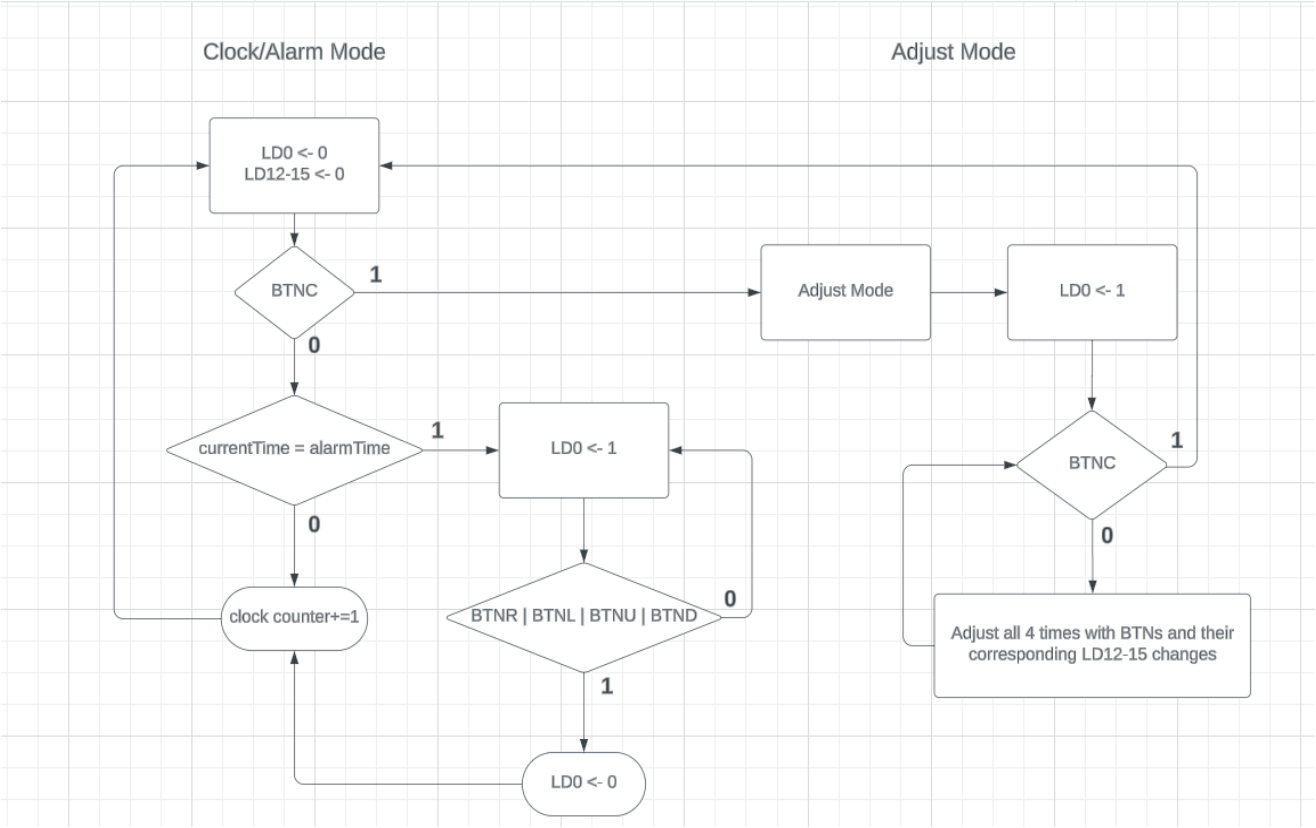
On the other hand, we enter the adjust mode as soon as the user presses the BTNC button. In adjust mode, LD0 is constantly turned on with no blinking, the decimal point stops oscillating and stops being displayed at all. During this mode, the user shall use the 4 other interactive push buttons BTNL, BTNR, BTNU, BTND where at each instance one of the 4 LEDs is turned on, indicating which anode is active at the current moment. I set our default anode to be the far-most left anode, it shall be the first anode visible as soon as we enter the adjust mode. LD15 is turned on when adjusting the time hours, LD14 is on when adjusting the time minutes, LD13 is on when adjusting the alarm hours and LD12 is on when adjusting the alarm minutes. Pressing BTNL moves the current anode by one to the left, and BTNR by one anode to the right. This controls the transition between anodes. BTNU is used to increment a single digit at a time, on the other hand, BTND is used to decrement a single digit at a time. Pressing the BTNC allows us to traverse back to the clock/alarm mode, with its associated features.

3. Diagrams

As shown in the block diagram below, the BTNC is the selection line of the multiplexer that chooses between the clock module and the adjust-time module. The output of both modules is the hour's tens, hour's units digits, minutes' tens digits, and the minutes' units digits. This output is the input of a multiplexer with a selection line from a counter with a frequency of 100 Hz. This counter is also a selection line for the decoder that switches the digits displayed across the four seven-segments. Here is our complete Data Path and the Control Unit for our whole clock system:


LD14
LD15
LD12
LD13
2x4 Binary Decoder
4-Digit 7-segment Display
Binary to 7-segment
Alarm's Hours Register
Alarm's Minutes Register
Clock's Hour Setup Register
Clock's Minutes Setup Register
En
Load
2-bit Binary Counter
4x1 MUX
Hours & Minutes Counter
100Hz
Clock Divider
100 Mhz
1Hz
LD14
LD12
1
0
LD14
LD15
Inc
Dec
Load
Up Down Counter
Flag
Trigger
CONTROL UNIT
Dec
LD12
LD13
LD14
LD15
Inc
Done
LD0
Clock Divider
100 Mhz
LD0

In addition to our Data Path and the Control Unit provided, we developed an Algorithmic State Machine chart that describes what happens at every state of our system. Here is our complete ASM chart design:



4. Code Design

Here are all the Verilog modules I made with their usages in our project. Each module is preceded by its description:

- **Clock Divider**: a clock divider that takes an input clock signal and generates a lower frequency output clock signal by dividing the input frequency by n

```
module ClockDivider #(parameter n = 50000000) (input clk, rst, output reg clk_out);
```

- **counter_MOD_N**: a counter that takes x number of bits and increments the counter

until it reaches n-1 (output the count mod n).

```
module counter_MOD_N #(parameter x = 3,parameter  n = 5) (input clk, rst, enable, load, input[x-1:0] data, output reg [x-1:0] count);
```

- **counter_modN_updown:** an up and down counter that takes x number of bits and increments and decrements the counter mod n accordingly

```
module counter_modN_updown #(parameter x = 3,parameter  n = 5) (input clk, reset, up, down, load, input[x-1:0] data, output reg [x-1:0] count);
```

- **Debouncer**: a debouncer is used to remove unwanted input noise from buttons to help in the detecting of a click

```
module Debouncer(input clk, reset, in, output out);
```

- **Synchronizer**: a synchronizer is used to delay the output so it is synchronized with the clock

```
module Synchronizer(input clk, reset, SIG, output reg SIG1);
```

- **PushBTN:** a push button detector is used to output a signal of one clock cycle for a single button click. It uses the debouncer, synchronizer, and the rising edge detector all together

```
module PushBTN(input clk,
input rst,
input in,
output out
);
```

- **SevSegDisplay:** this is the main 7-segment display for displaying all 4 digits

```
module SevSegDisplay(input current_mode,
input[3:0] numbers,
input[1:0] sw,
input clk_sec,
input enable,
output reg[0:6] segment,
output reg[3:0] anodes,
output reg decimal_point);
```

- **Clock_Alarm_Times:** we used this for the adjust mode that takes as input the current time and the current set alarm and outputs the digits after they're updated in the adjust, and return the new alarm and time values

```
module Clock_Alarm_Times( input clk, rst, en, input[4:0] buttons,
input[4:0] in_time_hours, in_alarm_hours, input[5:0] in_time_minutes, in_alarm_minutes,
output[3:0] adj_min_units, adj_hr_units, output[2:0] adj_min_tens, adj_hr_tens, output[1:0] adj_status,
output[4:0] out_time_hours, out_alarm_hours, output[5:0] out_time_minutes, out_alarm_minutes,
output reg[3:0] led);
```

- **clock_counter**: this is our main 24-hour clock used to display from 00:00 up till 23:59

```
module clock_counter(input clk, reset, enable, load_out,
input [5:0] time_mins,
input [4:0] time_hrs,
output [3:0] sec_units, min_units, hour_units,
output [2:0] sec_tens, min_tens, hour_tens);
```

- **MainProgram:** This is our main program where all modules are instantiated here

```
module MainProgram(input clk, rst, enable,
input[4:0] buttons_input,
output[0:6] segment,
output[3:0] anodes,
output reg led_mode,
output seg_dec_point,
output [3:0] led,
output sound);
```

- **rise_edge_det:** rising clock edge detector that uses FSM

```
module rise_edge_det(input clk, rst, w, output z);
```

- **rise_edge_det_ext:** this module gives an output of one only when the signal changes from 0 to 1 (rising edge)

```
module rise_edge_det_ext #(parameter n = 2) (input clk, rst, in, output reg out);
```

- **buzzer:** this is the module for the buzzer connected to our FPGA board

```
module buzzer(input clk, enable, output sound);
```

5. Implementation Obstacles

One of the main obstacles I faced while using Vivado is that the software itself gives an error on its own even though the code is correct and everything. However, when we generate bitstream again over the same untouched code it does not give an error this time, even though there was not a single line of code changed. Also, one of the other main issues I faced was that the buzzer itself was broken from the first place, which caused a lot of confusion since I could not indicate whether the buzzer is not working because of a software error from my end or a hardware malfunction from the buzzer component itself.

6. Validation Activities

We gradually spread throughout the project's various modules. We use Vivado to design the module first, then build test benches to verify the signal's validity before implementing the design on the FPGA board. To ensure that the code behaves as we intended, the adjust and clock modules ASMs were drawn before any code was written.

7. Contributions

Mark Kyrollos:

- Created the GitHub repository
- Designed the ASM chart
- Designed the Logisim for the digital clock
- Created the constraint file
- Designed all of the following source files: clock divider, Clock_Alarm_Times, the main program, the push buttons, the seven segment display, the buzzer, the clock_counter, the counter_MOD_N and counter_modN_updown, both rising edge detectors.
- Report

Aabed:

- Designed the Data Path and Control Unit
- Designed the following source files: the debouncer and the synchronizer

8. References

https://github.com/chingyi071/Basys3_buzzer/blob/master/schematic.png