THE AMERICAN UNIVERSITY IN CAIRO

الجامعة الأمريكية بالقاهرة

CSCE 2301

Section 2

Dr. Mohamed Shalan

Yasmina Mahdy

Rana Taher

Haya Shalaby

Digital Alarm Clock - Project Report

Table of contents

Introduction

This project is an implementation of a digital alarm clock on the Basys3 FPGA board, using Verilog HDL. It is an application of our understanding of sequential circuits, finite state machines and other concepts learned in the Digital Design I course, as well as the applied skills gained through its lab.

Project Description

The Alarm-Clock works as a typical digital clock, with functionality that allows for setting alarms as well as adjusting the value of the clock. The Alarm has two main modes of operation: Clock and Adjust. In the clock mode, the Alarm-Clock works as a normal HH:MM clock, with a blinker for the seconds, whereas in the adjust mode the Alarm-Clock allows the user to either adjust the time of the running clock, or set an alarm that goes off when it reaches the same value as the running clock. When the alarm goes off, the right-most LED blinks and, if an active buzzer is connected, it goes off at the same frequency as the LED. An extra snooze option was implemented to allow the user to snooze the alarm for 30 seconds.

Design Process

We have used a bottom-up approach for our design. First, we figured out the logic that would drive a possible implementation of our project, using elements we were familiar with such as counters, binary decoders and FFs. Then, we modeled the design on logisim evolution in order to gain a better understanding of the design's functionality and test the correctness of our logic and assumptions. Afterwards, we created ASM and CU & DP models to represent how we envisioned our project would work at a more abstract level. Finally, we used these abstracted models to code our Verilog program using behavioral modeling, finite state machines and sequential logic.

Model Descriptions

Logisim Evolution: [Appendices A, B, C & D]

This model's aim was to visualize how the circuit could be implemented as a low level design, in order to help us better understand the driving signals behind it. We used up-down mod counters for the four HH:MM digits of both the Time and Alarm modes, and only up counters for the SS digits of the clock mode. Note that we decided to use the same set of counters for both the running clock and setting clock modes, both labeled under Time, due to the fact that the setting value needs to be seen by the running value and vice versa.

We connected the left and right signals, which are used to switch between the 4 adjust modes, to a 2-bit counter that was then connected to a 2x4 decoder that is enabled by the adjust signal, and which produces the 4 enables needed to switch between the different modes. As for the up and down buttons, their disjunction was used as part of the enable of the counters to enable either up or down counting, depending on the value of the up-down signal received by the

counters. Finally, the center button was connected to a T Flip-Flop that allowed us to switch between the clock and the adjust modes.
Regarding the display, we used a seven segment multiplexed display that receives the 4 digits that are to be displayed–determined by a separate 2x1 multiplexor, and rapidly switches between them using a 4x1 binary decoder in order to conserve hardware resources while maintaining visibility of all 4 digits.
Detecting the alarm signal is achieved through 4 subtractors whose outputs are NOR'd to produce the Z flag. This flag is used to enable a T Flip-Flop that toggles the alarm LED, but only if in the running clock mode.

When implementing the actual design using Verilog, we did not exactly follow this model, due to the fact that using an HDL allows for more flexibility, and so we opted to use behavioral modeling instead of a structural one that would have closely followed the logisim model.

ASM: [Appendices E, F, G & H]
The ASM diagram describes the decisions made to reach t each of the stages that we have in our design. It begins in the normal clock mode and checks if there is a signal to go to the adjust mode or not if not then it starts counting the seconds up to 59 and that enables the minutes until we reach 59 minutes and 59 seconds then the modulo 24 hours counter starts and runs until we reach 23:59:59 then it resets to 00:00:00. If the adjust signal was 1 then we can increase or decrease the values of the time and alarm in the following states TH, TM, AH, and AM. It starts in the time hours state and up and down the values we want for setting the normal clock time hour. Then it could either go back to the normal mode if it does not receive a change in the adjust signal and choose to change to another state by giving a right or left signal. For Example, as we are in TH if we select right then we go to TM but if we select left we go to AM. After adjusting the diagram goes back to normal clock state and keeps counting normally until the value set in the AH and AM states is reached in the time and that then generates a signal to start the buzzer and the blinking LED.

DP and CU: [Appendices I, J & K]
The Datapath diagram consists of four big components: Clock registers (two hour registers, two minute registers, and two second registers), Alarm registers (two hours registers and two minte registers), Alarm, and Display. Each register in the Clock or Alarm registers has a clock, an enable, and an up/down signal. In the case of hour registers, there are also loads. The Alarm component takes four digits from the Clock registers and four digits from the Alarm registers and subtracts bit-by-bit using subtractors. The outputs of the subtractors are NOR-ed to check the Z flag which can set off the buzzer and LED. The Display also takes four digits from the Clock registers and four digits from the Alarm registers where they are multiplexed to choose which digit will be display. The most prominent things we changed was that we added a Snooze feature and that we simplified the subtractors to only a comparison in Verilog. The Control Unit is drawn

as a black box which receieves the button signals after going through the *push_detector.v* and outputs the individual loads and enables.

Implementation

This section will explain the purpose and function of each major (original) module in the project. We used a modular approach when designing and coding our digital alarm clock in order to ensure organization and smooth functionality and testing of the program. Additionally, it helped us break it down into smaller parts and work bottom-up. All the following files can be found in the 'Source Files' folder on our project's Github repository.

Time.v: This module is responsible for the normal clock functionality as well as setting the clock time. It uses three instances of the Up-Down-Modulo Counter, a module we took from the lab and modified to fit our counter criteria, one for seconds, one for minutes, and one for hours. The parameters for the Up-Down-Modulo Counter are set accordingly. The enables for each counter is determined by either the adjust signal (whether the user is adjusting the time or not). If it is the latter, a counter is enabled when the previous is finished counting to its maximum value, otherwise, it is controlled by whether the user is incrementing or decrementing (pressing up or down – which also determines whether the counter will be up-counting or down-counting) while in the setting time mode (indicated by LEDs). This module continuously outputs the resulting four digits (hours and minutes) from the counters.

AlarmMode.v: This module is responsible for the alarm functionality, i.e. setting the clock alarm. It uses two instances of the Up-Down-Modulo Counter, one for minutes, and one for hours. The parameters for the Up-Down-Modulo Counter are set accordingly. A counter's enable is controlled by whether the user is incrementing or decrementing (pressing up or down – which also determines whether the counter will be up-counting or down-counting) while in the setting alarm mode (indicated by LEDs). This module continuously outputs the resulting four digits (hours and minutes) from the counters.

Time_Alarm.v: This module combines the *Time.v* and *AlarmMode.v* modules and picks which set of digits (hours and minutes) to output (to be used for the display later on). It makes that choice based on the alarm mode enables, i.e. if the user is currently adjusting the alarm, it will display the alarm hours and minutes while it would display the time hours and minutes otherwise. It also assigns different clocks to *Time.v* and *AlarmMode.v* depending on whether the user is adjusting the alarm or setting the time (and so we need a faster clock) or the digital alarm clock is carrying out its normal clock functionality (and so we need a slower clock). This module also compares between the clock and alarm registers in order to check for the Z flag, i.e. if the alarm previously set is up. It outputs that along with the current value of the seconds which is an output from the *Time.v* instance.

stateLogic.v: This module is the finite state machine (FSM) of our digital alarm clock. It creates and manages seven states: TH (Time Hours), TM (Time Minutes), AH (Alarm Hours), AM (Alarm Minutes), Clock, Alarm, and Snooze. It switches between the states based on the signals received as a result of pressing one or more of the buttons. If we are in the Clock state, and the user presses the center button, adjust is activated and the first state when Adjusting is TH (setting the clock hour). Pressing the left or right buttons navigates between setting the clock hours, the clock minutes, the alarm hours, and the alarm minutes, from left to right respectively. At any point while adjusting, pressing the center button returns the state to the normal, counting Clock state. In the event that an alarm is set off, the state becomes the Alarm state, and any button from left, right, up, or down can be pressed to return the state to Clock (i.e. stop the alarm). Pressing the center button switches to the Snooze state. The Snooze state uses a Z flag to indicate when the snoozing period (30 seconds) is over, and then, switches to the Alarm state. The Snooze state can be exited before the snooze period is up through pressing any button; this will switch to the Clock state. This module outputs the adjust signal and enables which will be later used for the display, both based on the states.

Display.v: This module is responsible for choosing and outputting the correct display for the FPGA's seven-segment display. It uses an instance of *SevenSegDecWithEn.v* and an instance of *Binary_Counter.v*, modules we had previously done in labs but then modified slightly to fit our program. As a result, we get a hardware-efficient multiplexed display, and the digit displayed at a time is picked using the binary counter.

Top.v: This is the top module that connects all the major and minor modules in the project, and tha center point of operation for our program. It receives the up, down, left, right, and center signals as input, and uses an instance of *Buttons.v* which operates five instances of push_detector.v, a lab module that combines a rising edge detector, a synchronizer, and a debouncer. After these signals are output from the *Buttons.v* instance, they are fed into a *stateLogic.v* instance so that the state can be determined, and consequently, the enables needed for the instance of *Time_Alarm.v*. The four digits output from that are input into an instance of *Display.v* which finally outputs the seven segments ad anode actives needed for correctly displaying on the FPGA. The alarm LED and buzzer are also assigned here, mostly using enables from the stateLogic.v instance output. Additionally, it determines whether the seconds'dedicated decimal point on the display blinks or not.

<u>*Modules directly from 2302 or with minor modifications*</u>
Binary_counter.v
ClockDivider.v
SevenSegDecWithEn.v: added the decimal point (blinking) functionality
debouncer.v

push_detector.v
Rising_edge.v
Sync.v
Up_Down_Mod_Counter.v: added an up/down functionality to the modulo counter
Buttons.v

Validation Activities

For testing and validation purposes we have used the following test benches [Available on the project's repository under the folder Verification/TestBenches]:

- Up_Down_Mod_TB.v: used to test the functionality of the adapted version of the mod counter module under the name Up_Down_Mod_Counter.v
- secMinHr_TB.v: used to test the functionality of the HH:MM:SS counter module that was adapted from the lab, under the name Time_Mode.v, which has since been abandoned and replaced by Time.v
- Time_TB: used to test the new version of the HH:MM:SS counter module under the name Time.v
- FSM_TB: used to test the correctness of the finite state machine module, previously called FSM.v but now called stateLogic.v

We have also utilized constraint files that have been since merged with the Top.xdc constraint file, in order to test our design at various stages of the implementation

Challenges

At the beginning of the project our main issue was when we were trying to figure out the logic for the ASM and the DP & CU and that made us decide to create the whole project on logisim to be able to figure out the logic of the code. Then when we were trying to implement the code we were faced with a number of challenges. The first of these was that at the beginning we thought of calling the Time.v and AlarmMode.v modules on their own in the Top module as we tried to follow the ASM. However, we soon discovered that we can't do that as we need to choose which output of which module needs to be displayed and compare between their values and use that signal in other modules that need it. Therefore, we decided to create one module to combine both modules and do these tasks. Moreover, when we were trying to implement the stop and the snooze and turning on for the alarm we thought we could implement that in a separate finite state machine. However, The program did not recognize any of the changes done in that FSM so we added Two new states to the main FSM one for the alarm and one for the snooze.Also, instead of having a separate state for the stop we decided to keep stop as a signal coming from the buttons. We also faced issues with the snooze option being not functional at the beginning as we thought that we could keep the snooze counter enabled and use the reset to control its start and stop but we figured out that we also need to control the enable. In addition to that some of the lab modules that we used had clock dividers in them and that complicated the

speed of the pushbuttons and the display. Thus, we decided to remove clock dividers from any module and have two clock dividers in the top module that generates two clocks, one for the buttons and one for the normal clock mode. Finally, what helped us be able to fix most of these issues was the modular approach that we used when creating our code.

Future Improvements

In the future, we would like to improve on this project by implementing a Timer mode along with the available clock and alarm options, which would have utilized the down counting functionality from the Up_Down_Mod_Counter.v module in order to produce an alarm signal when the Timer expires. We also wanted to expand on the timer idea, by combining it with the snooze option to allow the user to pick the snooze duration they would like, instead of the preset 30 seconds. Moreover, we would have liked to use a clocking wizard in our project to decrease the power consumption of the design and allow for a smoother implementation, however we were unable to due to time constraints. Finally, we had hopes to use a passive buzzer and produce a more varied sound than the one currently in use, which was achieved by connecting an oscillating frequency to an active buzzer, yet had to abandon the idea due to time constraints and the difficulty of determining the appropriate tones to use.

Data Reports and Clocks

Refer to appendices L-O for the utilization and timing & delay information provided by the Vivado software.

Our design utilizes only two clocks in the top module:

- Sec_clock which operates at 1Hz and drives the running clock
- funct_clk which operates at 200Hz and drives all the other components which require a relatively fast clock for proper operation, such as the seven-segment display and the push button detectors.

Conclusion

Overall, this project managed to challenge our digital design and hardware-programming skills. We were able to learn from our shortcomings during the first project and applied the suitable fixes this time. Most prominently, we learned proper use of Github (todos, branches, pull requests and merging, etc.), modular design approach, and improved our debugging skills. Early on, we also learned how to professionally and neatly create design diagrams such as ASM, CU, and DP. We are glad to have went through that experience, and we will be taking the previously mentioned future improvements into consideration.

Contribution & Borrowed Modules

<u>Models</u>

Haya Shalaby: logisim & DP

Rana Taher: logisim & CU & ASM

Yasmina Mahdy: logisim & DP & ASM

Original Modules

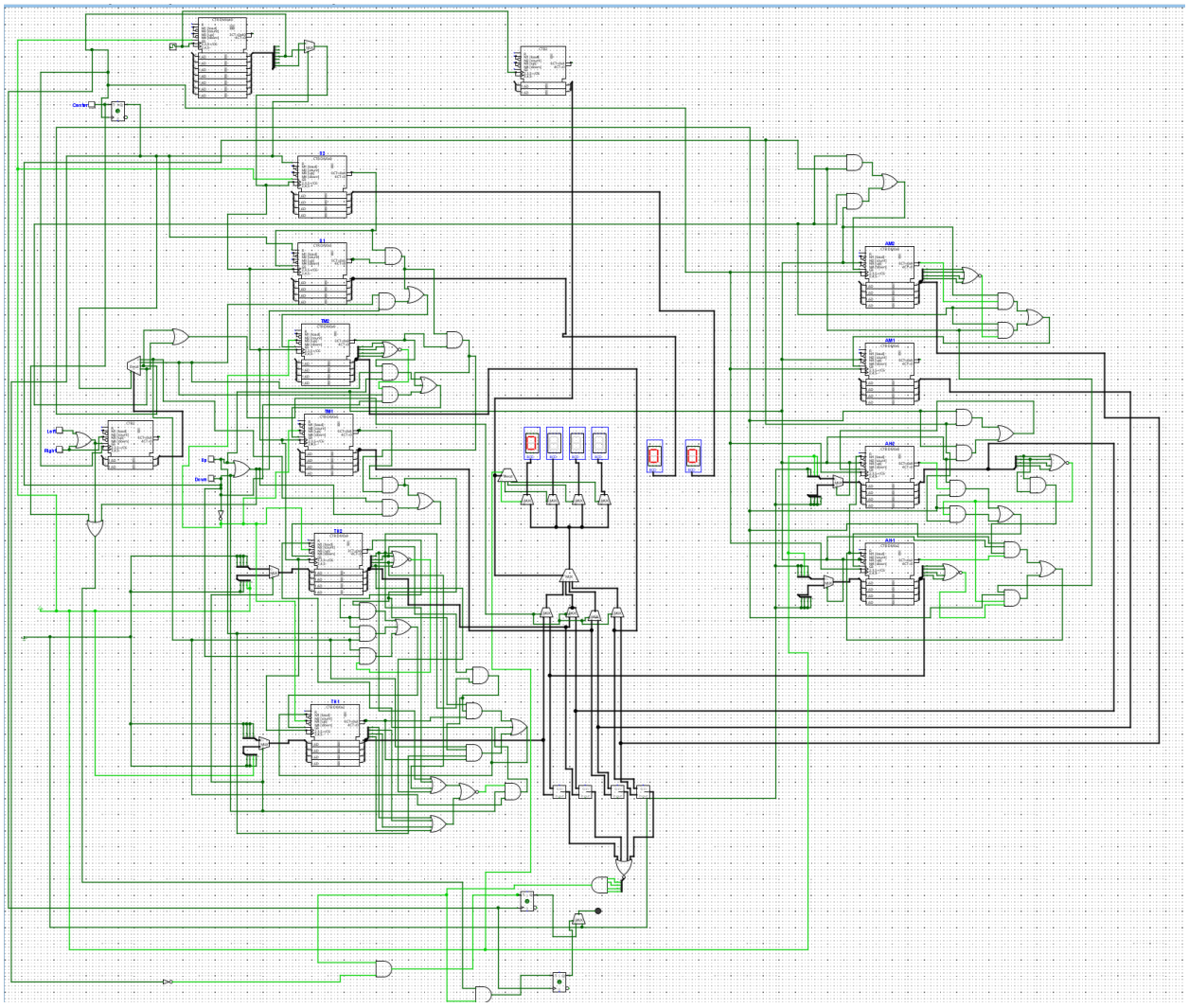
Note that this isn't representative of the overall contribution and work of each member, as we all actively participated in testing, debugging and figuring out the logic behind the modules, as well as editing them later on. This is simply a list of the modules that were originally assigned to each member.

Haya Shalaby:	Top.v &	stateLogic,	
Rana Taher:	Top.v &	Time_Alarm.v	&AlarmMode.v,
Yasmina Mahdy:	Top.v &	Time_Alarm.v	&Time.v

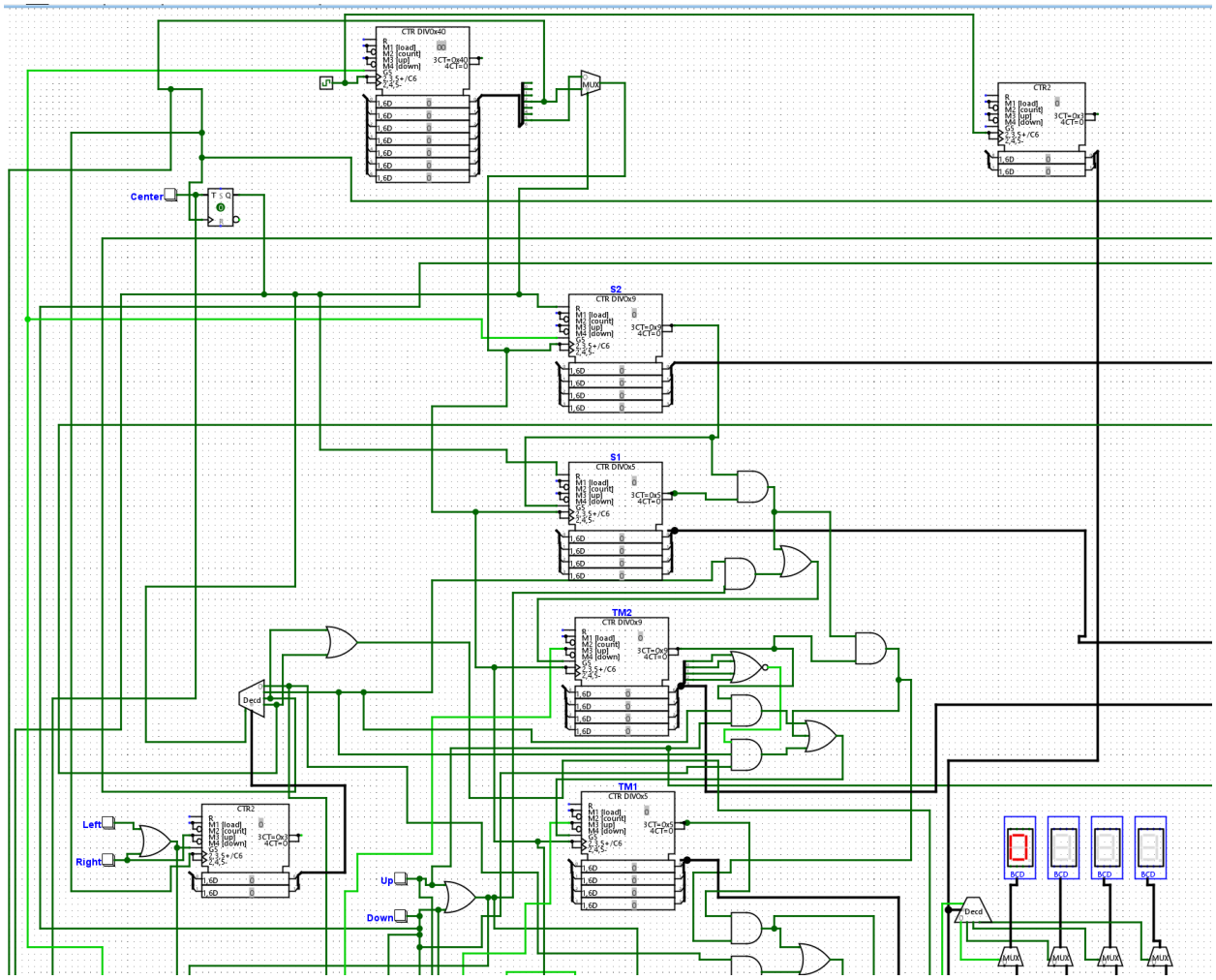
Modules directly from 2302 or with minor modifications
Binary_counter.v
ClockDivider.v
SevenSegDecWithEn.v
debouncer.v
push_detector.v
Rising_edge.v
Sync.v
Display.v
Up_Down_Mod_Counter.v
Buttons.v

Appendices

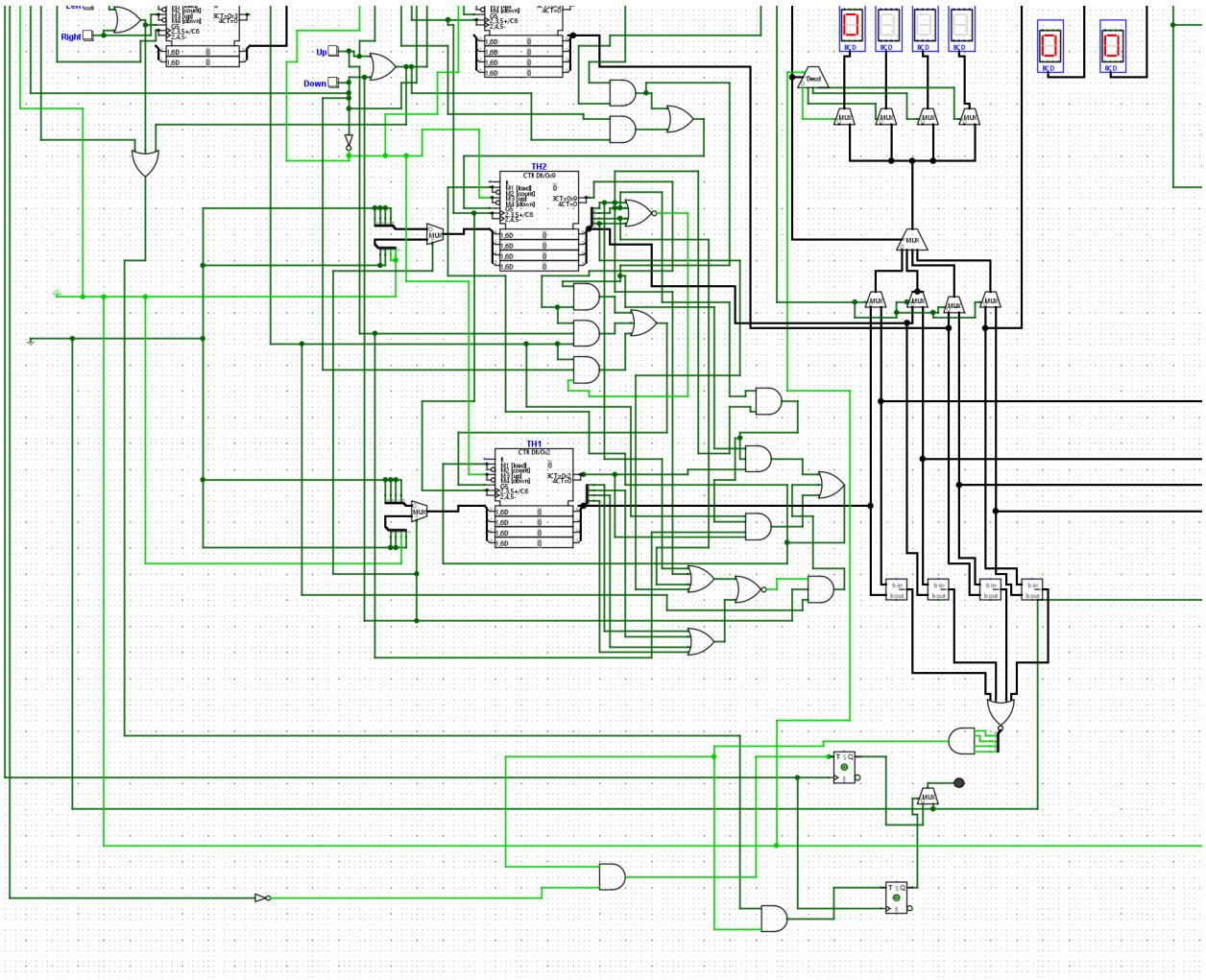
Appendix A



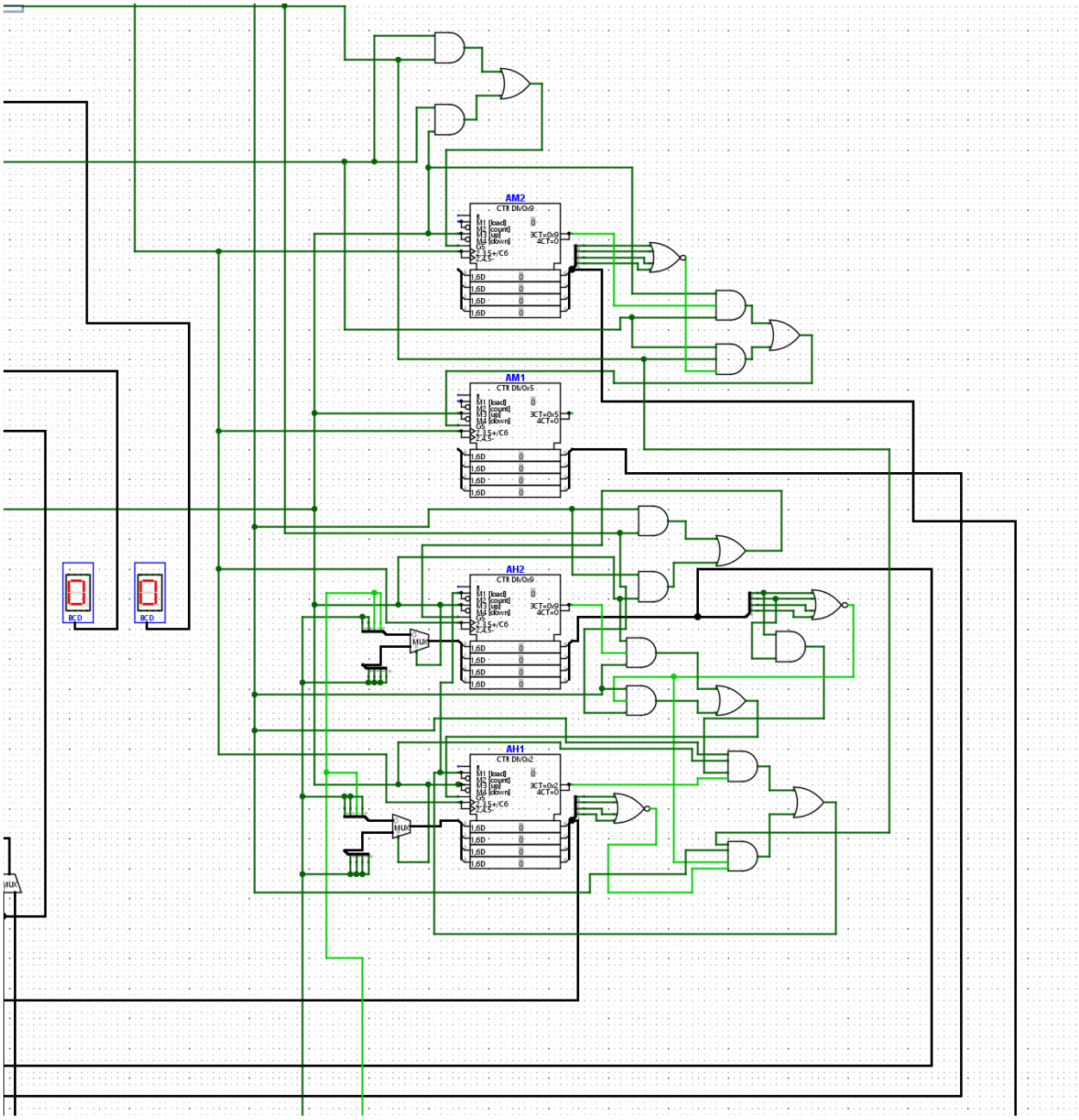
Appendix B


CTR DIV0x40
MUX
CTR2
Center
S2
CTR DIV0x9
S1
CTR DIV0x5
TM2
Decd
TM1
Left
Right
Up
Down
BCD
MUX

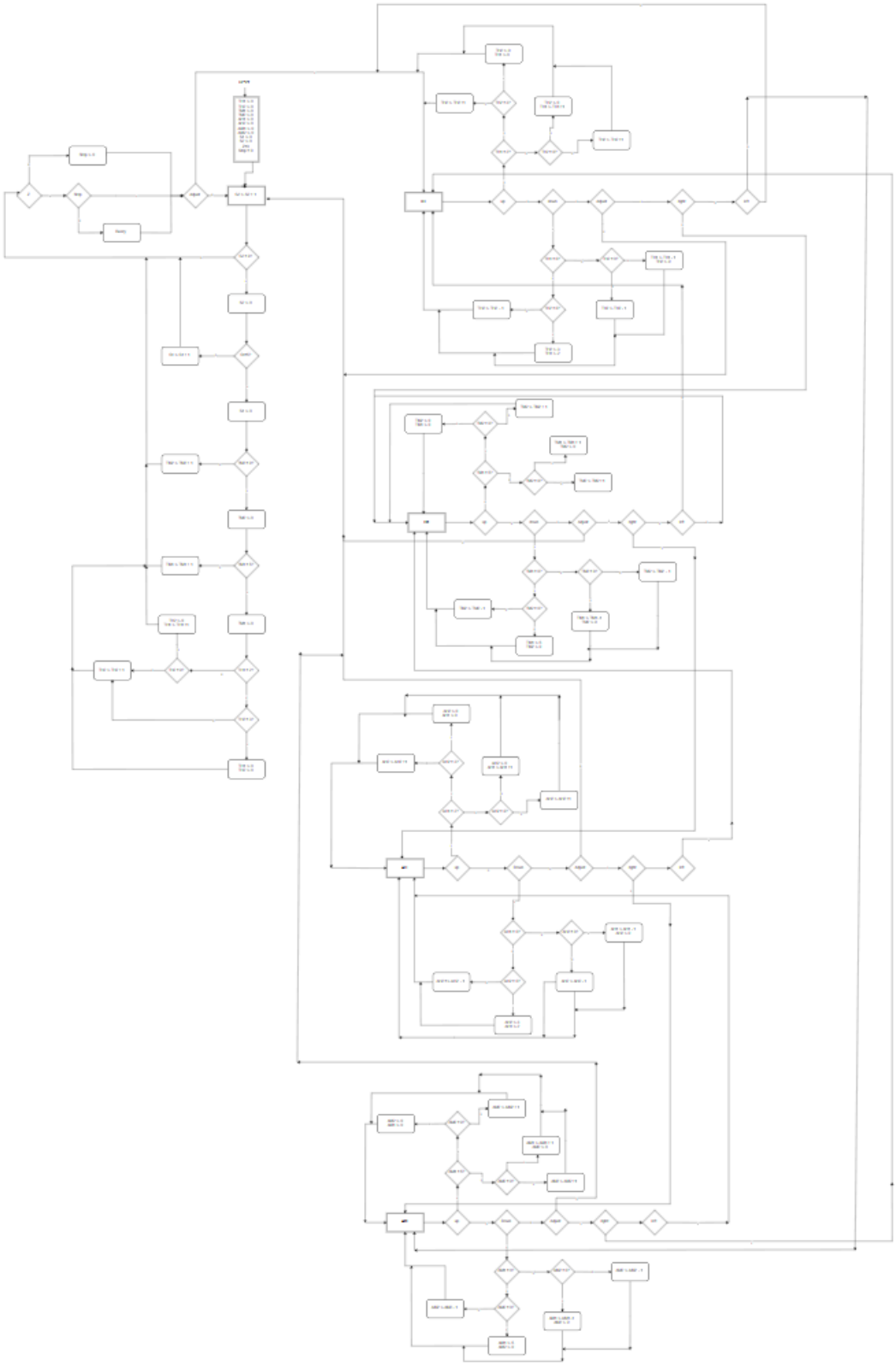
Appendix C



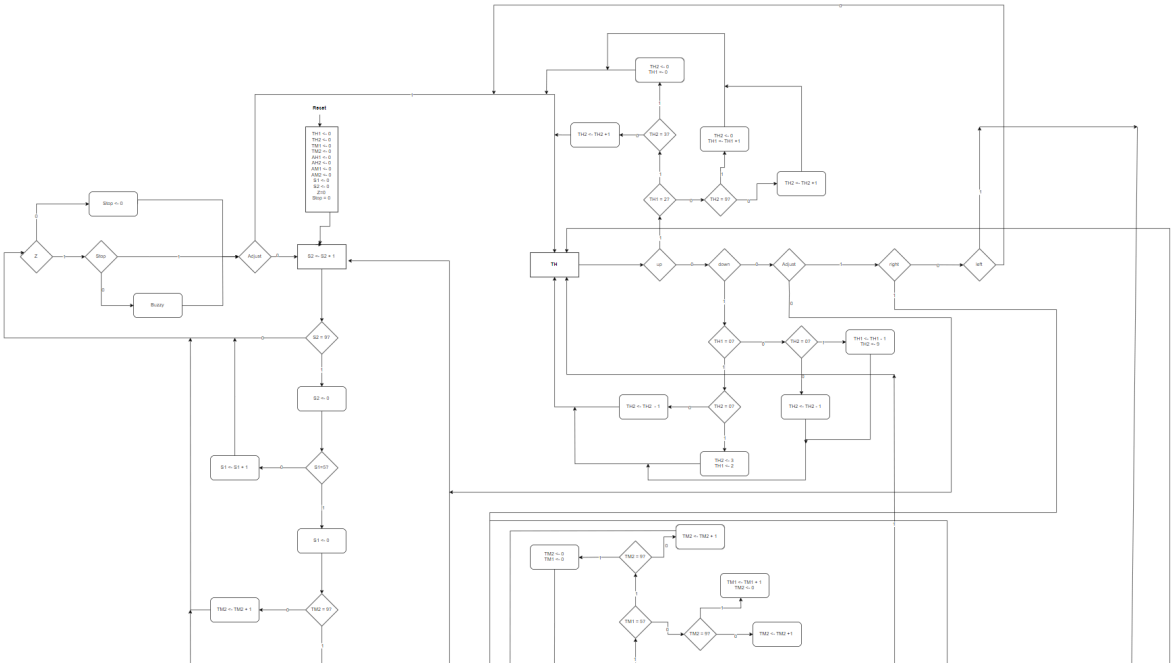
Appendix D



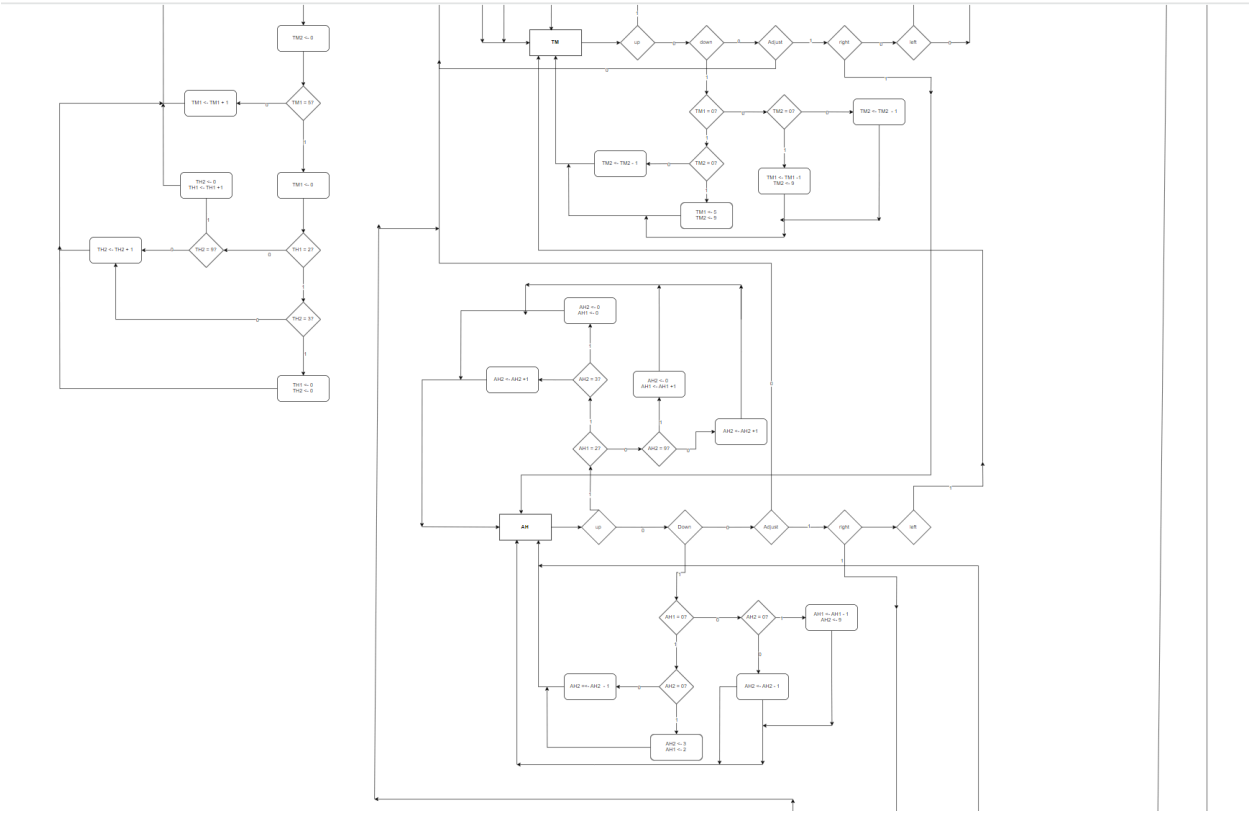
Appendix E



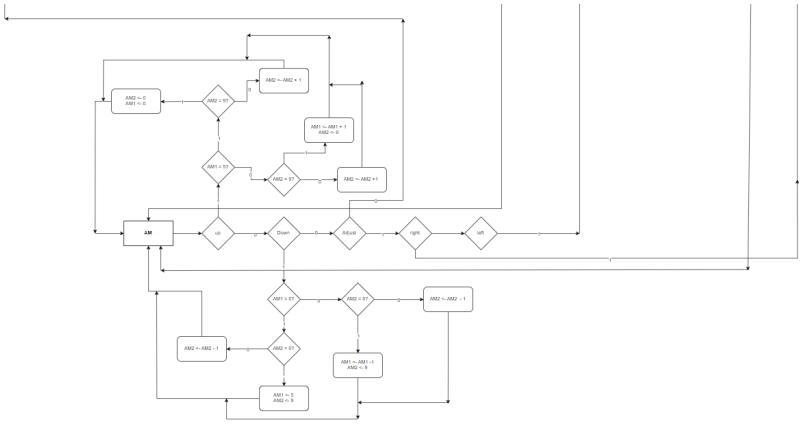
Appendix F



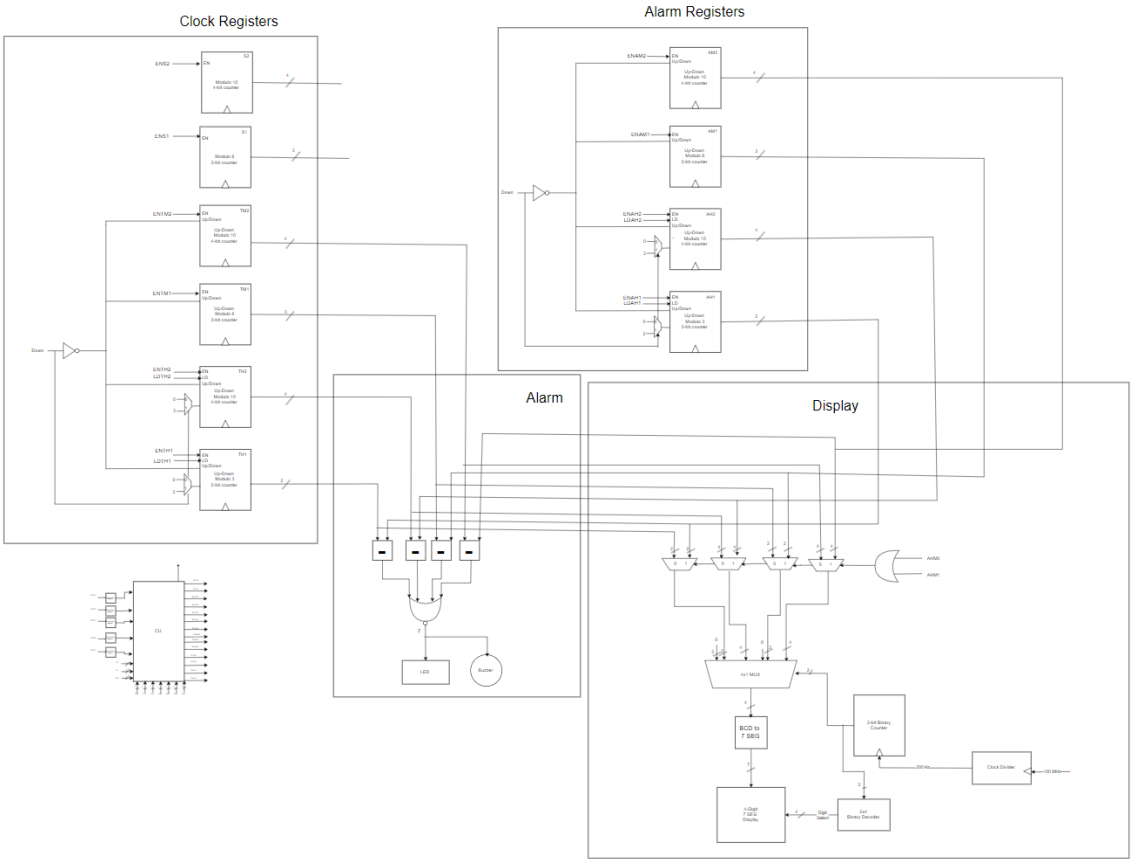
Appendix G



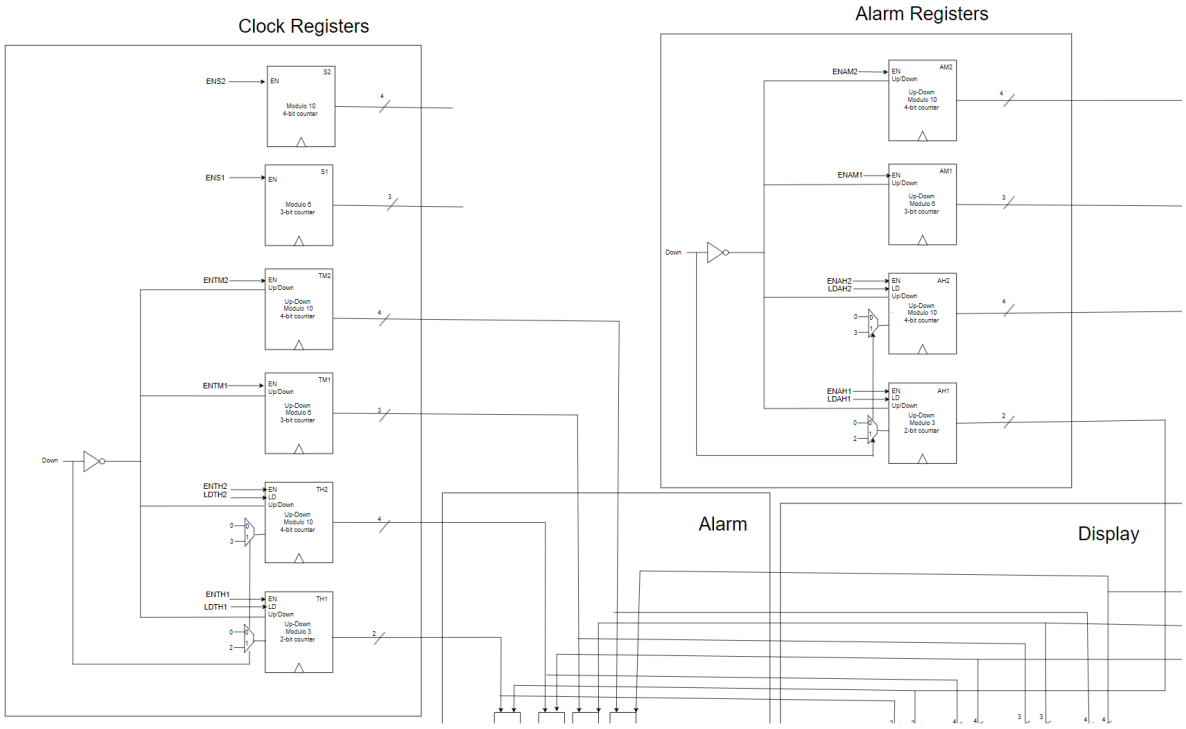
Appendix H



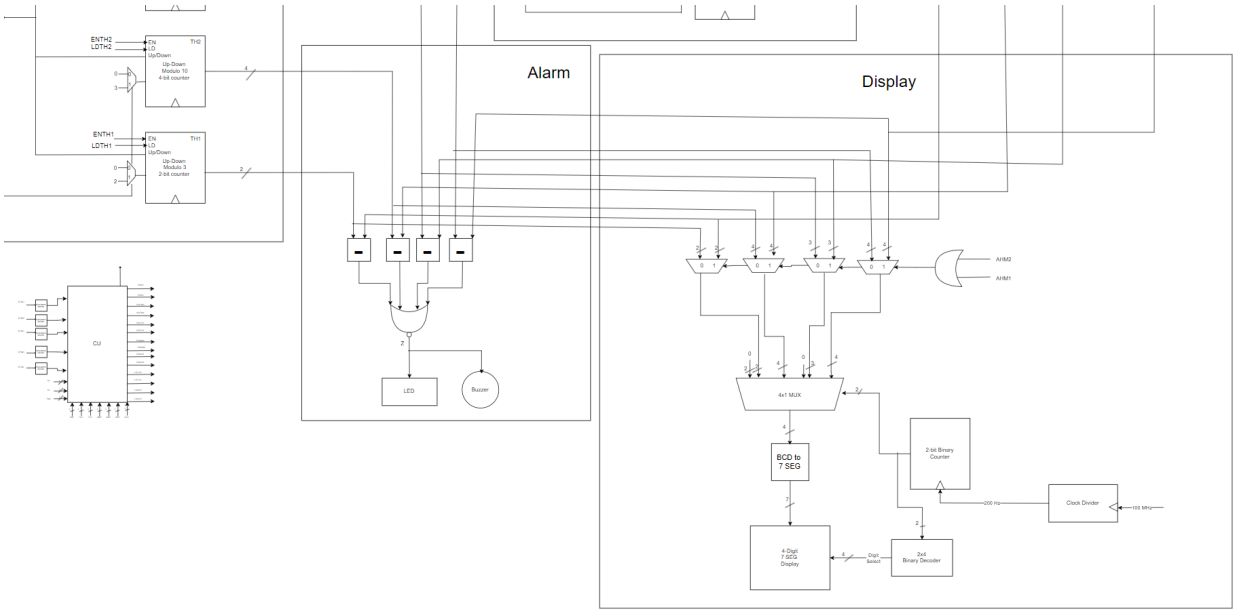
Appendix I



Appendix J



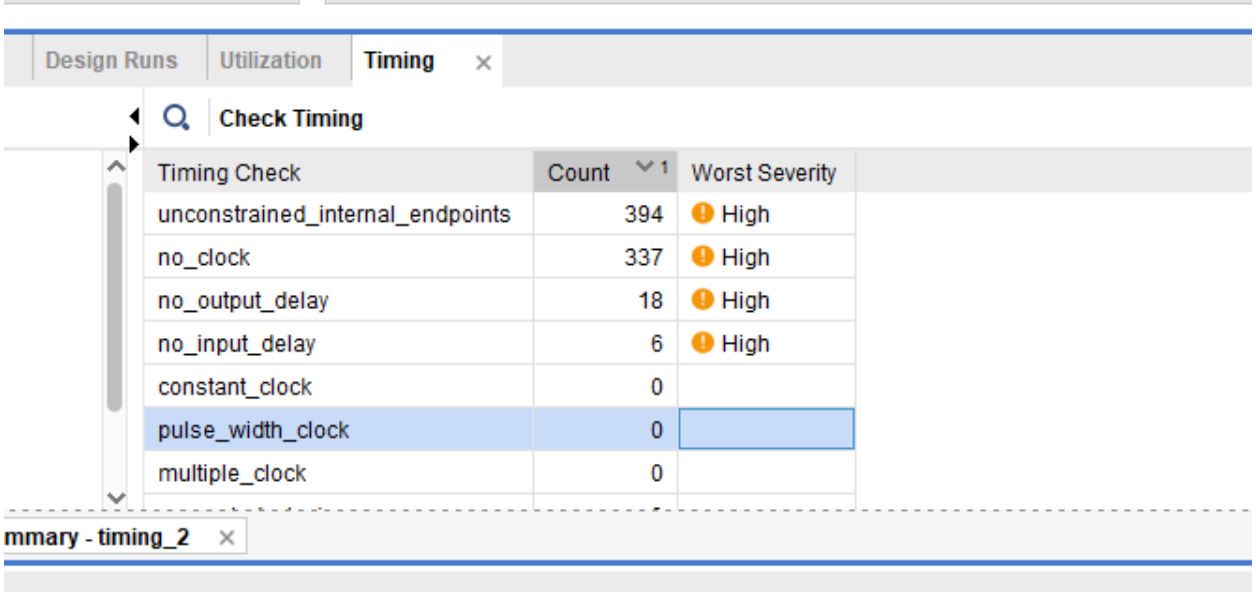
Appendix K



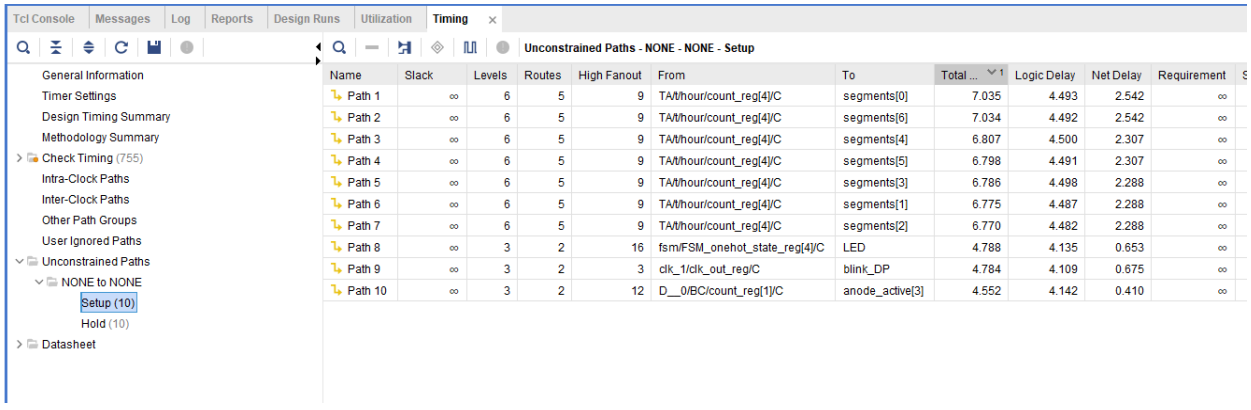
Appendix L

Resource	Utilization	Available	Utilization %
LUT	239	20800	1.15
FF	201	41600	0.48
IO	25	106	23.58

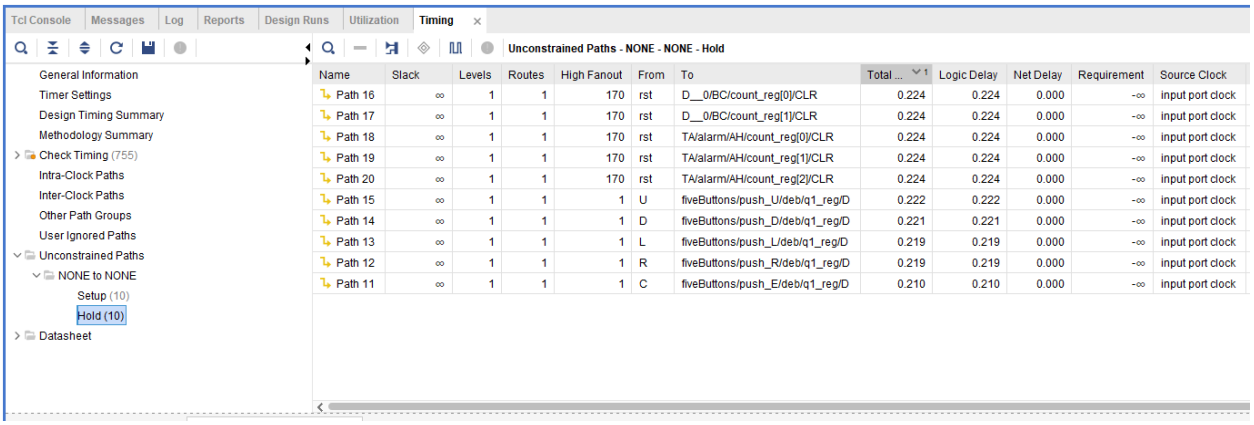
Appendix M



Appendix N



Appendix O


Tcl Console
Messages
Log
Reports
Design Runs
Utilization
Timing
General Information
Timer Settings
Design Timing Summary
Methodology Summary
Check Timing (755)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths
NONE to NONE
Setup (10)
Hold (10)
Datasheet
Unconstrained Paths - NONE - NONE - Hold
Name Slack Levels Routes High Fanout From To Total ... Logic Delay Net Delay Requirement Source Clock
Path 16 ∞ 1 1 170 rst D__0/BC/count_reg[0]/CLR 0.224 0.224 0.000 -∞ input port clock
Path 17 ∞ 1 1 170 rst D__0/BC/count_reg[1]/CLR 0.224 0.224 0.000 -∞ input port clock
Path 18 ∞ 1 1 170 rst TA/alarm/AH/count_reg[0]/CLR 0.224 0.224 0.000 -∞ input port clock
Path 19 ∞ 1 1 170 rst TA/alarm/AH/count_reg[1]/CLR 0.224 0.224 0.000 -∞ input port clock
Path 20 ∞ 1 1 170 rst TA/alarm/AH/count_reg[2]/CLR 0.224 0.224 0.000 -∞ input port clock
Path 15 ∞ 1 1 1 U fiveButtons/push_U/deb/q1_reg/D 0.222 0.222 0.000 -∞ input port clock
Path 14 ∞ 1 1 1 D fiveButtons/push_D/deb/q1_reg/D 0.221 0.221 0.000 -∞ input port clock
Path 13 ∞ 1 1 1 L fiveButtons/push_L/deb/q1_reg/D 0.219 0.219 0.000 -∞ input port clock
Path 12 ∞ 1 1 1 R fiveButtons/push_R/deb/q1_reg/D 0.219 0.219 0.000 -∞ input port clock
Path 11 ∞ 1 1 1 C fiveButtons/push_E/deb/q1_reg/D 0.210 0.210 0.000 -∞ input port clock