THOMPSON RIVERS UNIVERSITY

CENG 3010 – Digital Systems Design

Digital Controller for a Programmable Locking Safe

Andrei Vivar (T00668428)

Raiden Yamaoka (T00661480)

Jack Ukitetu (T00647975)

Date submitted December 2, 2022

Table of Contents

List of Figures

List of Tables

1 Introduction

Many homeowners around the world are oftentimes unequipped for a break-in, resulting in a loss of money or valuables. In a normal scenario where someone is looking to prevent break-ins the first place they go is to a locksmith for doors and window locks, this costs an average of $300 and can be up to $800 **[1]** depending on the number of locks. However, there is a simple and cheaper method than replacing your locks; A Programmable Locking Safe. Globally the most common form of locking safe is a combination lock **[3]**, this can be difficult for some people to operate. Oftentimes a good value safe that is small enough to fit in an average household can be hard to find and when choosing a safe you want it to be the most reliable. In most cases, this will either be more in cost or more time invested searching. With our programmable locking safe, you could purchase a cheaper, more reliable and simple-to-unlock safe that can fit into any average home.

Our system aims to give people that may not have the chance to pay for the best locksmith or safe a helping hand to hopefully find them safe that will fit their budget and where they can feel confident with their purchase. It helps to know you can easily find a reliable and easy-to-use safe for your belongings. Many locking safes suffer from being too large for an average home **[4]**, too expensive or having a difficult-to-use locking mechanism. We aim to protect both users and their belongings by implementing a programmable lock so that the password can be changed by the user but not by anyone that doesn't know the current password; this means that although hired employees can interact with the password if something goes wrong they will not be able to change the user's password unless requested. Our programmable locking safe seeks to provide a viable household appliance that will give them the extra closure that they want, while also providing safety and peace of mind to let them have the time to also get on with their own life. This will all be achieved while delivering it in a cheap-to-manufacture, simple-to-use, and long-lasting format because we know everyone deserves to have a cheap and readily available way to keep their belongings safe. Our main goal is to relieve stress for anyone that has any concerns about break-ins or robberies.

2 Design Problem

This section has the following two subsections:

2.1 Problem Definition

Due to the rise of cases of home break-ins and thefts, the demand for better home security has been increased. We needed to come up with a solution that will help fight against this problem but will take as few resources as possible. An affordable yet secure way to combat this issue is to develop a programmable lock safe. For this project, we were tasked to design a prototype for a Programmable Lock Safe which will make it simpler and easier for customers to eliminate the need to go to a locksmith and pay a high price to get a customized lock for their house and to ensure the safety of the customer's belongings.

2.2 Design Requirements

This section has the following three subsections:

2.2.1 Functions

- The function of this Programmable Lock Safe is to allow the user to be able to input a personalized passcode on a lock safe to be able to store their personal belongings in the safe. It acts as storage that will safely store the customer's belongings with the reliance on the personalized password that was set up initially.
- **Goal of the function:** The main goal is to design a programmable lock safe with the use of an FPGA board and a keypad that serves as the lock system for the programmable lock safe.
- **Action:** The user input a personalized password for the locked system.
- **Functions:** The main function of the lock safe are:
    - It gives the client storage for their personal belongings.
    - It gives the customer privilege to input their own password
    - It displays the inputted code on the seven-segment display
- **Behaviours:** A passcode is entered by the user and then the system checks if it matches the current set passcode inside the code. If it matches the passcode with the current set code, it opens the lock safe for a certain amount of time and displays a notification, which is by turning on a light in our prototype. If it does not match the code, it notifies the user with turning on another light, and the safe will remain locked.
- **Structure:** The programmable lock safe is made up of various parts: the body of the safe, an FPGA board which acts as the interface for the lock system, a keypad that acts as a medium to communicate with the FPGA, and a VHDL code which is the language used.

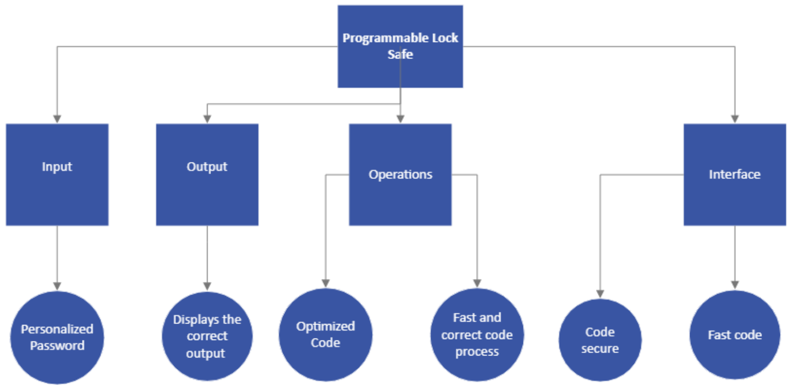


Figure 1. Function Tree

2.2.2 Objectives

Our objective is to develop a design that serves as a secure storage that is able to accommodate the customer's personal belongings safely with the reliance on the personalized passcode system that is provided during the initial setup of the programmable lock safe. For our objectives, we broke down major ideas into individual sections: safety, cost, and efficiency. For safety, our objective is to have a program with a non-breachable code, so not any random person can access and modify the password that is set up in the FPGA board. For the cost, our objective for our lock safe is to be cheap and easily accessible to the public. For efficiency, our objective for our lock safe is to have fast code and be user-friendly, resulting in our app being easy to use. Below is a visual representation of our objective tree:

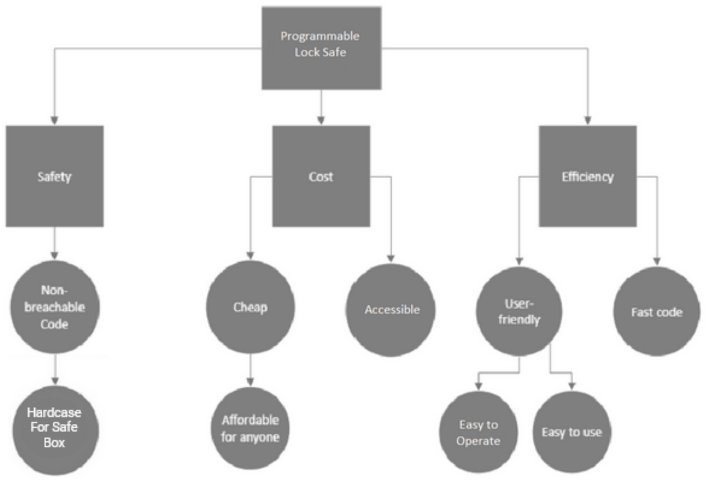


Figure 2. Objective Tree

2.2.3 Constraints

In this project, we are given four major constraints by our professor:

1. The safe has multiple compartments so that various categories of goods of value can easily be accommodated.
2. The safe is automatically locked after a certain time frame.
3. The design project should not have unnecessary digital elements.
4. The design must have at least one sequential machine.

3 Stakeholders

We Identified 5 key stakeholders as listed below in the table, this includes their name, type, priority, power and needs.

Name	Type	Priority	Power	Needs
Homeowners	External	Primary	Direct	A working programmable safe.
Investors	External	Primary	Direct	Project be completed on time and no investment money is done to waste
Employees	Internal	Primary	Direct	To continue to develop an updated version of the safe to maintain a job.
Suppliers	External	Secondary	Indirect	To continue to use their product to manufacture our safes
Media	External	Secondary	Indirect	Reliable information to keep the public up to date on the progress of the project

Table 1. Stakeholders and needs

4 Scheduling and budgeting

- **Green** indicates that the team has fully met the criteria for the deliverable
- **Amber/Yellow** indicates that the criteria have not been fully met, but the team has a work plan to achieve the criteria
- **Blue** indicates that the team has not fully met the criteria and has no plan to meet the criteria (requiring help from management)

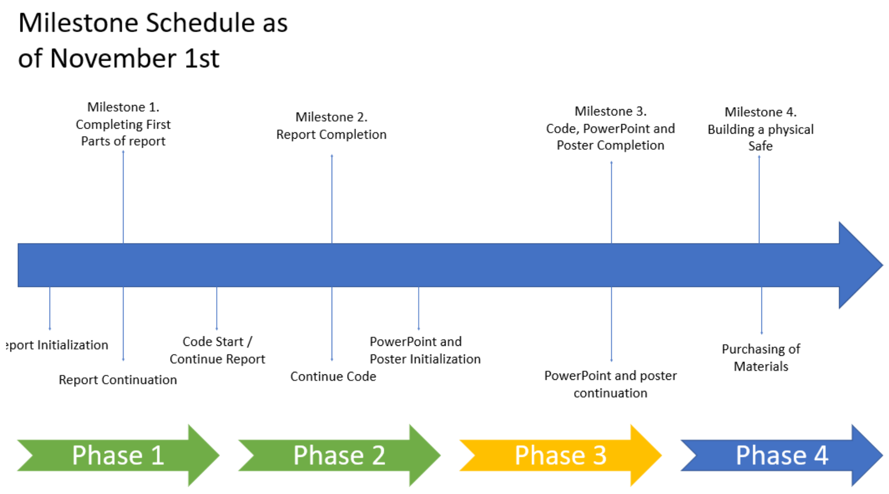
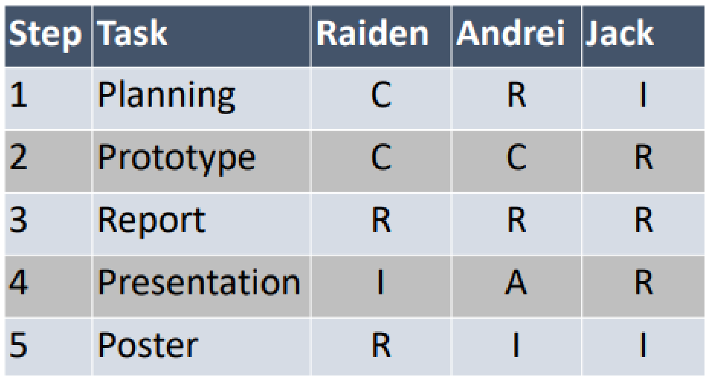

Milestone Schedule as of November 1st
Milestone 1. Completing First Parts of report
Milestone 2. Report Completion
Milestone 3. Code, PowerPoint and Poster Completion
Milestone 4. Building a physical Safe
port Initialization
Report Continuation
Code Start / Continue Report
Continue Code
PowerPoint and Poster Initialization
PowerPoint and poster continuation
Purchasing of Materials
Phase 1
Phase 2
Phase 3
Phase 4

Figure 3. Milestone Schedule



Step	Task	Raiden	Andrei	Jack
1	Planning	C	R	I
2	Prototype	C	C	R
3	Report	R	R	R
4	Presentation	I	A	R
5	Poster	R	I	I

Figure 4. Responsibility Matrix

Cost estimates for Manufacturing

The largest customer for smaller-sized safes are hotels so for a cost estimate we will base it on one hotel. An average hotel has around 315 rooms. Given that each room has a safe we will need to produce 315 safes. The price of the Basys 3 Artix-7 FPGA board of $149.99, the keypad at a price of $14.99 and the actual safe at approximately $100 of iron-carbon alloy. Using the current taxation rate in BC of 12% buying a safe comes out to a total of approximately $267 per safe. 315 safes * $267 per safe = $84,105 of manufactured goods per safe. The average cost for a burglary safe is around 750$ so if we were to retail our safe for around $500 we would profit around $200 and maintain market superiority due to our cheaper price.

5 Solution

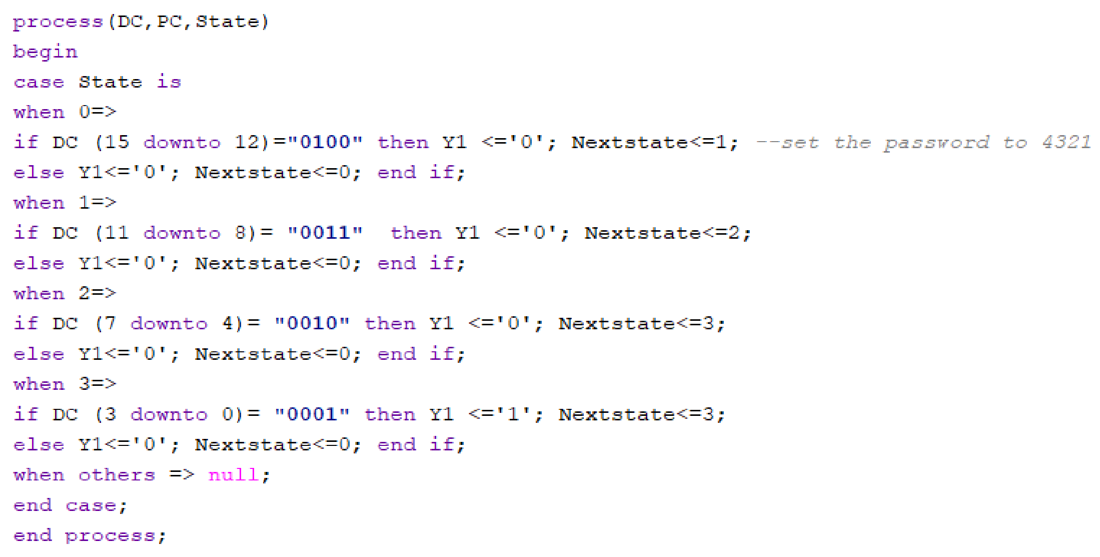
5.1 Solution 1

For this solution, we will be using the FPGA board and the PmodKypd for our design solution. Based on a Moore sequential machine. The switches on the FPGA will be used to select a specific anode of the 4 seven segment display. Therefore if sw[0] is turned on the first of the four seven-segment displays will turn on. With this, a number can be entered into the specific anode and the next switch. For the password, a sequential detector is made to detect the sequence of the button pressed on the keypad and compare it to the store password which is “432”.

If the password is gotten an led named “Y1” as seen below will light open showing the safe is open. If it fails it goes back to state 0 where you can try again

Reason for not using

- The password can not be changed as it is stored within the code
- When typing a combination of numbers for the password it does not automatically go to the next seven segment display and must turn on the switch to move the seven segment display.
- When two of the same number are entered e.g “2256”. it will take it as “256” which will make the sequence detector fail.
- Once open it does not lock itself after a certain time.



```
process(DC,PC,State)
begin
case State is
when 0=>
if DC (15 downto 12)="0100" then Y1 <='0'; Nextstate<=1; --set the password to 4321
else Y1<='0'; Nextstate<=0; end if;
when 1=>
if DC (11 downto 8)= "0011"  then Y1 <='0'; Nextstate<=2;
else Y1<='0'; Nextstate<=0; end if;
when 2=>
if DC (7 downto 4)= "0010" then Y1 <='0'; Nextstate<=3;
else Y1<='0'; Nextstate<=0; end if;
when 3=>
if DC (3 downto 0)= "0001" then Y1 <='1'; Nextstate<=3;
else Y1<='0'; Nextstate<=0; end if;
when others => null;
end case;
end process;
```

Figure 5. Sequence detector code for solution 1

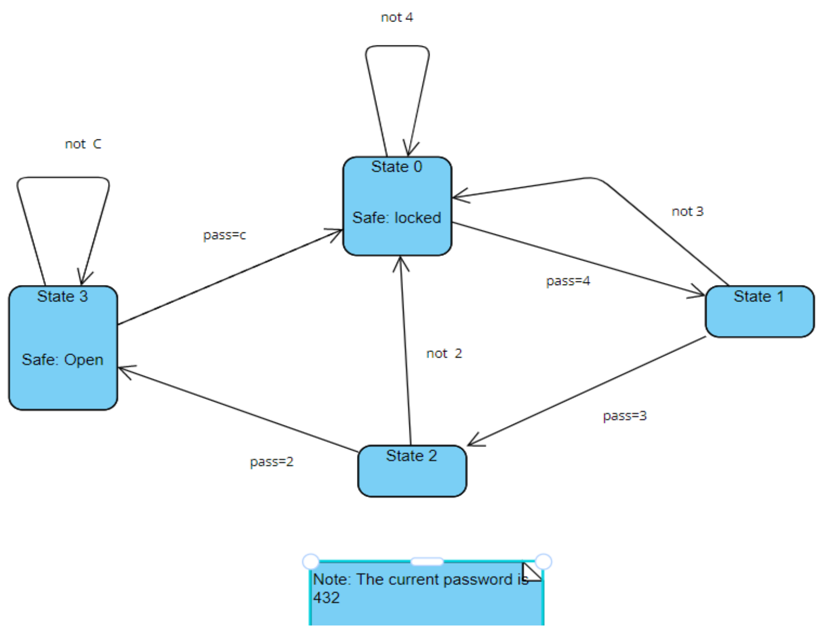


Figure 6. State diagram for solution 1.

5.2 Solution 2

For this solution, an improvement of the previous solution was done. The sequential detector using Moore was kept and expanded allowing the user to change the password. It also included the delay setting it to 2 min so letting the safe lock itself after that time period. It also allows the user to enter a combination without the use of the switch as seen in solution 1. There displaying all 4 numbers of the combination on the 4 seven segment display. It also fixes the error of the combination of two or more sequences of the same number like “2222” as it checks with the clock to check the time these numbers were pressed.

Features added:

- Pressing A on the keypad will serve as an enter key.
- Pressing C will clear all digits on the seven-segment display.
- Pressing B will allow changing the password as long as the former password was entered first.
- The safe will stay open for approx 2 minutes and then lock itself.

Sequential code

```
process(clk, current_state, Key_Pressed_mp, Cancel, TC, RP, UL, delay_TC)
begin
next_state <= current_state;
CE <= '0';
count_CLR <= '0';
shift_en <= '0';
sr_CLR <= '0';
disp_CLR <= '0';
load_C <= '0';
load_M <= '0';
m_CLR <= '0';
LOCK <= '1';
UNLOCK <= '0';
FAIL <= '0';
REPROG <= '0';
delay_CLR <= '0';
delay_CE <= '0';
tc_VAL <= "1111111111111111";
case current_state is
when idle =>
sr_CLR <= '1';
disp_CLR <= '1';
count_CLR <= '1';
delay_CE <= '0';
delay_CLR <= '1';
if Key_Pressed_mp = '1' then
```

```
next_state <= ty_take_sample;
end if;
when ty_take_sample =>
shift_en <= '1';
CE <= '1';
if Cancel = '1' then
next_state <= idle;
else
next_state <= typing;
end if;
when typing =>
if Cancel = '1' then
next_state <= idle;
elsif TC = '1' then
next_state <= load_M_reg;
elsif Key_Pressed_mp = '1' then
next_state <= ty_take_sample;
end if;
when load_M_reg =>
load_M <= '1';
delay_CLR <= '0';
delay_CE <= '1';
tc_VAL <= "0000000000001111"; -- delay timer: .016 sec
if delay_TC = '1' then
```

```
next_state <= compare;
end if;
when compare =>
delay_CE <= '0';
delay_CLR <= '1';
sr_CLR <= '1';
count_CLR <= '1';
disp_CLR <= '1';
if RP = '1' then
next_state <= reprogram;
elsif UL = '1' then
next_state <= unlocked;
else
next_state <= failure;
end if;
when unlocked =>
m_CLR <= '1';
UNLOCK <= '1';
LOCK <= '0';
delay_CE <= '1';
delay_CLR <= '0';
if delay_TC = '1' then -- hold unlock for roughly 2 sec
next_state <= idle;
end if;
```

```
when failure =>
m_CLR <= '1';
FAIL <= '1';
delay_CE <= '1';
delay_CLR <= '0';
if delay_TC = '1' then -- hold failure f
next_state <= idle;
end if;
when reprogram =>
m_CLR <= '1';
REPROG <= '1';
if Cancel = '1' then
next_state <= idle;
elsif TC = '1' then
next_state <= load_C_reg;
elsif Key_Pressed_mp = '1' then
next_state <= rp_take_sample;
end if;
when rp_take_sample =>
shift_en <= '1';
CE <= '1';
if Cancel = '1' then
next_state <= idle;
else

next_state <= reprogram;
end if;
when load_C_reg =>
load_C <= '1';
next_state <= idle;
end case;
end process COMBONATIONAL_LOGIC;
```

Figure 7. Sequence code for the Moore machines.

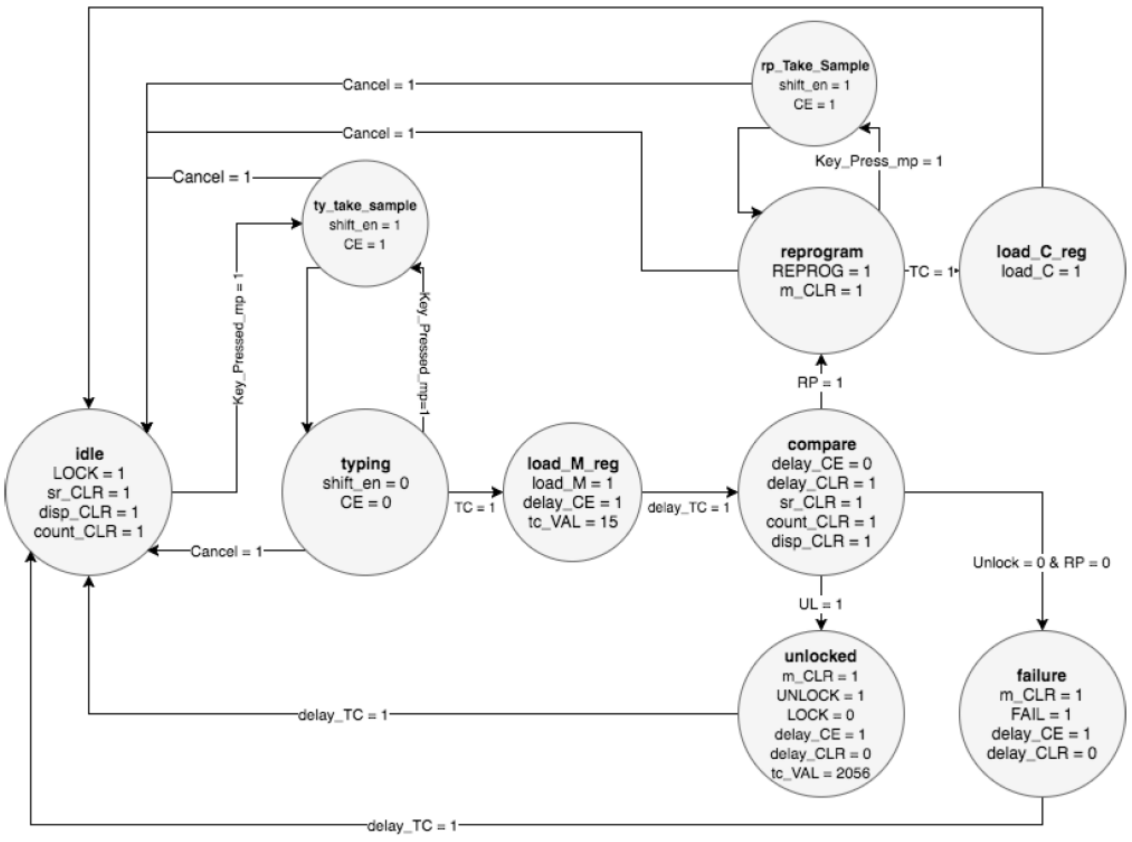


Figure 8. State diagram for the final solution.

5.3 Final Solution

The Final is an iteration of solution 2. Making the delay on the unlock closer to 2min. We also added a lock so if the password is gotten wrong you get locked out for 2 minutes. Increasing the security of the lock. A debounce was also added to deal with when two numbers were pressed at the same time[6].

To stop an operation at any time, press "C." By doing this, you can reset all processes and return the Keypad to lock mode.

Enter the master code "1234" or reprogrammable code (default: "1234") to unlock the door. To enter the code, press "A."

The keypad now unlocks. Only the LED marked "Unlock" will be lit. If the Failure LED is lit, the instructions were not correctly followed or the right code was not input.

The keypad will go back into Lock mode after two minutes, with the Lock LED being the only one on.

Enter the master or reprogrammable code to begin reprogramming. To enter reprogram mode, press "B." Then Enter the new passcode and then press A.

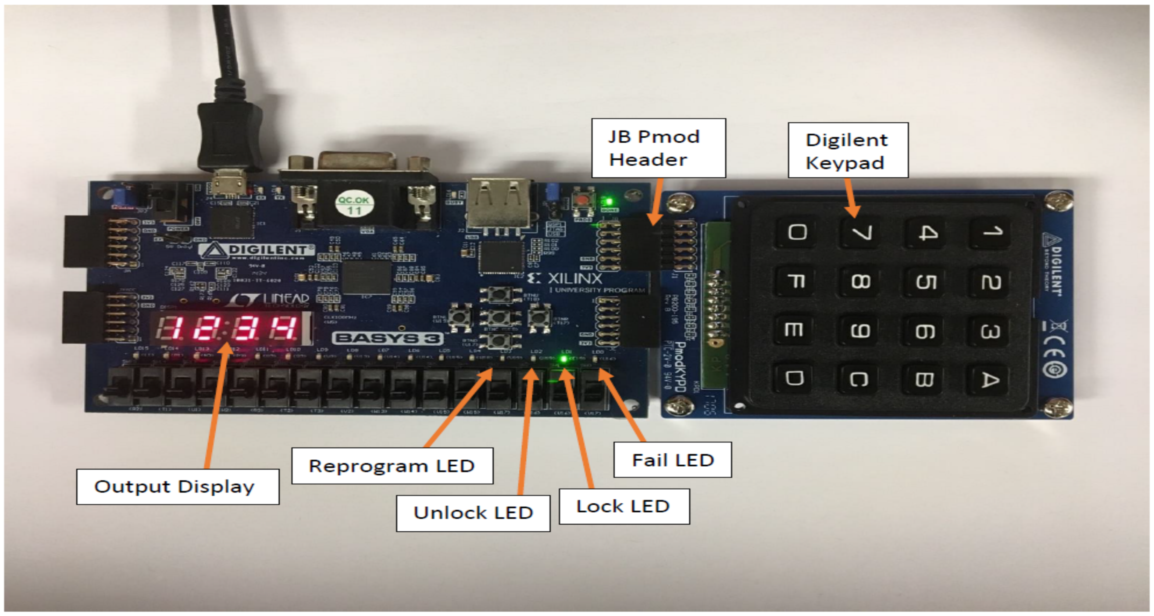


Figure 9. Layout picture of the FPGA and Pmod Keypad

Why it is the better solution

The safe is automatically locked after a certain time frame.

The design project displays all the necessary information needed by the user.

The design has one sequential Moore machine which is used to detect the sequence of the entered password.

Design Criteria	Accuracy	Reliability	Cost	Performance	User-friendly	Programmability	Sum
Weighing Factor	0.19	0.17	0.13	0.17	0.15	0.19	1.0
Solution 1	6 / 1.14	6 / 1.02	9 / 1.17	8 / 1.36	5 / 0.75	5 / 0.95	6.39
Solution 2	8 / 1.52	9 / 1.53	9 / 1.17	7 / 1.19	8 / 1.20	9 / 1.71	8.32
Final Solution	9 / 1.71	9 / 1.53	10 / 1.30	8 / 1.36	9 / 1.35	10 / 1.90	9.15

Table 2. Decision Matrix.

5.3.1 Features

Features	Approach to enable
Display Number entered	When a number is entered it takes the position of the row and column of the number. Which is then sent to a decoder which converts that to a seven-segment out bit_vector.
Detected Passcode	Using a Moore sequential machine. We can compare the password entered to the saved password. In order to determine if the safe should be unlocked or not.
Change the passcode	When the safe is unlocked and B is pressed on the Keypad. You can then enter a new password which is saved into a signal in a code.
Clear seven-segment display	When C is pressed it clears all saved digits in their position on each seven-segment display to 0.
Delay for the unlock safe to lock	We used a clock counter to determine the time needed for the safe to stay open for 2 minutes and then lock itself.

Table 3. Feature table.

5.3.2 Environmental, Societal, Safety, and Economic Considerations

Our team took into consideration societal and cost the most, as we want this to be a low-cost product for everyone to use. Having a cheap product means spending as little as possible, but also making it available to the vast majority of the population which is one of our main goals. Societally, we wanted to consider everyone and their struggles when it comes to using technology so our interface would be simple and output and input as little as possible to ensure everyone has no issues. Environmental considerations were discussed but not deemed important enough as this product will be entirely digital having no harm to the environment. Safety was another point we wanted to consider as the safety of the user is the main function of our product. We would ensure no data from the user is taken other than the location to provide accurate weather conditions and have a non-hackable code with no possible breaches. Ensure the safe will stay locked after multiple failed attempts.

5.3.3 Limitations

- If numbers are entered too fast they might not be registered.
- We did not get to implement a servo due to us not having it.
- Entering the passcode through Bluetooth was considered but could not be implemented on time.
- The user would have to wait for 2 minutes if a wrong passcode is entered.

6 Life-Long Learning

When working on this project we learnt a lot about how this be done here is a list:

- How to integrate the Pmod Keypad with the FPGA.
- Spent a total of 4 hours working on the code to include the lock delay.
- Implementation of code from a complex state graph to a case state in Vhdl.
- Learned how to refresh the clock in order to display all digits on the seven segments display.
- Researching how to integrate the Bluetooth module into the FPGA. (This could not be done due to a lack of resources and time limits.)
- Researching how to implement the servo with the FPGA**[5]**.

7 Technical Standard

Electronic Safe Lock Ratings

In the United States UL 2058 is the primary standard for electronic safe locks. Locks which are certified to meet the criteria of UL 2058 are often listed as "UL Type 1". Like mechanical locks, electronic locks are required to offer a minimum of 1,000,000 possible combinations; however, there are several unique requirements for electronic locks. To aid reliability and prevent accidental lockouts batteries must be stored in the keypad on the outside of the door so that they can be replaced even when the container is locked and combinations must be stored in non-volatile memory so that codes will not be lost when the lock loses power. To prevent decoding or manipulation of the lock all storage, and processing of combinations must occur in electronics located on the secure side of the door (ie inside the container).

UL 2058 also lays out a variety of durability requirements intended to ensure the lock will continue to function even with a certain amount of rough handling or in less-than-ideal environments. Unlike UL 768's multiple grades, UL 2058 currently only offers one grade which makes it essentially a pass/fail system. Using an electronic lock that is not rated as UL Type 1 is not recommended on any burglary-rated safe**[2]**.

8 Team Work

8.1 Meeting 1

Time: October 5, 2022, 2:00 pm to 3:30 pm

Agenda: Brainstorming about the project design

Team Member	Previous Task	Completion State	Next Task
Andrei Vivar	N/A	N/A	Report Introduction
Raiden Yamaoka	N/A	N/A	Poster Concept/Template
Jack Ukitetu	N/A	N/A	Initial VHDL code

Table 4. Meeting 1

8.2 Meeting 2

Time: October 18, 2022, 2:30 pm to 4:00 pm

Agenda: Starting the reports and research

Team Member	Previous Task	Completion State	Next Task
Andrei Vivar	Report Introduction	100%	Report design problems
Raiden Yamaoka	Poster Concept/Template	100%	Report stakeholders and budgeting
Jack Ukitetu	Initial VHDL code	20%	Continuing VHDL code

Table 5. Meeting 2

8.3 Meeting 3

Time: October 25, 2022, 6:00 pm to 8:15 pm

Agenda: Working on the reports and code

Team Member	Previous Task	Completion State	Next Task
Andrei Vivar	Report design problems	100%	Report Lifelong Learning and Technical Standard
Raiden Yamaoka	Report stakeholders and budgeting	100%	Finishing Poster
Jack Ukitetu	Continuing VHDL code	60%	Continuing VHDL code

Table 6. Meeting 3

8.4 Meeting 4

Time: November 1, 2022, 4:00 pm to 5:50 pm

Agenda: Working on the reports and code

Team Member	Previous Task	Completion State	Next Task
Andrei Vivar	Report Lifelong Learning and Technical Standard	100%	Creating Presentation
Raiden Yamaoka	Finishing Poster	100%	Report Conclusion and Future work
Jack Ukitetu	Continuing VHDL code	100%	Report Solutions

Table 7. Meeting 4

8.5 Meeting 5

Time: November 8, 2022, 3:00 pm to 6:00 pm

Agenda: Finishing the reports and code

Team Member	Previous Task	Completion State	Next Task
Andrei Vivar	Creating Presentation	100%	Finalizing Presentation
Raiden Yamaoka	Report Conclusion and Future Work	100%	Finalizing Report/Poster
Jack Ukitetu	Report Solutions	100%	Finalizing Report

Table 8. Meeting 5

8.6 Meeting 6

Time: November 30, 2022, 3:00 pm to 6:00 pm

Agenda: Completing and cleaning up final tasks

Team Member	Previous Task	Completion State	Next Task
Andrei Vivar	Finalizing Presentation	100%	N/A
Raiden Yamaoka	Finalizing Report/Poster	100%	N/A
Jack Ukitetu	Finalizing Report	100%	N/A

Table 9. Meeting 6

9 Conclusion and Future Work

- As a group, we achieved multiple big milestones. We coded all the desired requirements for a programmable locking safe working with VHDL. Our safe has many features including a fully functional password locking system, the ability to change the four-digit password as well as lockout when the password is incorrect. We learnt many cooperative methods in how to work well as a group while both programming together and writing this report. Our final design has many functions that we wanted to include and is not missing one key feature, a physical safe. As this design is just a technical prototype we felt that it was not necessary to include the box and only test the application of the password system. The features that we have included for the safe are; Programmable password, a seven-segment display for the password and a lockout system when the password is incorrect. The objectives we achieved are based on cost, safety and efficiency. For the cost, we achieved our only goal in regards to cost and made our product cheap to manufacture, however, this is only the locking mechanism and is not considering the physical box, this maintains access for many people to use, all while doing it at low project-related costs. Secondly, for safety, the main goals we accomplished were that our code is non-breachable and provides a safe and secure password system both of which were achieved. Finally, our code is only missing none of our efficiency goals and they were all accomplished as the code runs at amazing speeds with few halts and has a very basic interface so that anyone can use it without any issues.

- Our final design is intended to be used as a general-use safe for an average household, therefore more planning and research would have to be done to find the best and cheapest materials for a safe. Improvements to the code for next time would be adding a more user-friendly lock-out system as currently it is one wrong attempt and it locks you out. Secondly, if a password is entered too quickly sometimes numbers might not be registered. Upgrading the hardware to work faster and possibly switching to a more efficient keypad is also something that we would like to do. As well as possibly programming a notification system to send a notice to a mobile device based on when the safe is opened.

10 References

[1] D. Weinberger L. Pelchen, “How Much Does A Locksmith Cost?”, 8 June 2022. [Online]. Available:

https://www.forbes.com/home-improvement/home-security/cost-to-hire-a-locksmith/

(Accessed 5 October 2022)

[2] W. Wayne, “Ratings for Safe Locks,” 18 October 2018. [Online]. Available:

https://lockreference.com/ratings-for-safe-locks/

(Accessed 28 November 2022)

[3] R. Valdes, “How Safecracking Works”, 6 April 2004. [Online]. Available:

https://home.howstuffworks.com/home-improvement/household-safety/safecracking1.htm

(Accessed 6 October 2022)

[4] M. Borchert, “Buying a Safe: A Complete Guide”, 29 September 2022. [Online]. Available:

https://www.familyhandyman.com/article/home-safe-buying-guide/

(Accessed 6 October 2022)

[5] C. Ramos, “Servomotor Control with PWM and VHDL,” 20 December 2012. [Online]. Available:
https://www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL
(Accessed 28 November 2022).

[6] Scott, “7-Segment Display Driver for Multiple Digits (VHDL),” March 2021. [Online]. Available:
https://forum.digikey.com/t/7-segment-display-driver-for-multiple-digits-vhdl/12526
(Accessed 25 November 2022).

Appendix

Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;

entity compare_tb is -- Port ( ); end compare_tb;

architecture Behavioral of compare_tb is

component compare is Port( clk : in STD_LOGIC; serial_input : in STD_LOGIC_VECTOR(3
downto 0); shift_en : in STD_LOGIC; sr_CLR : in STD_LOGIC; load_C : in STD_LOGIC; load_M
: in STD_LOGIC; M_CLR : in STD_LOGIC; delay_CLR : in std_logic; tc_VAL: in
std_logic_vector(10 downto 0); delay_CE : in std_logic;

disp_output : out STD_LOGIC_VECTOR(15 downto 0); RP : out STD_LOGIC; UL : out
STD_LOGIC; delay_TC : out std_logic); end component compare;

constant clk_period: time := 100ns;

signal clk : STD_LOGIC := '0'; signal serial_input : STD_LOGIC_VECTOR(3 downto 0) :=
"0000"; signal shift_en, sr_CLR, load_C, load_M, M_CLR, RP, UL, delay_CLR, delay_CE,
delay_TC : STD_LOGIC := '0'; signal disp_output : STD_LOGIC_VECTOR(15 downto 0) :=
(others => '0'); signal tc_VAL : STD_LOGIC_VECTOR(10 downto 0) := "00000000111";

begin dut : compare port map ( clk => clk, serial_input => serial_input, shift_en => shift_en,
sr_CLR => sr_CLR, load_C => load_C, load_M => load_M, M_CLR => M_CLR, disp_output =>
disp_output, RP => RP, UL => UL, delay_CLR => delay_CLR, tc_VAL => tc_VAL, delay_CE =>
delay_CE, delay_TC => delay_TC );

CLK_PROCESS: process begin clk <= not(clk); wait for clk_period/2; end process clk_process;

STIMULUS_PROCESS: process begin --BEGIN UNLOCK SIMULATION USING MASTER CODE--
serial_input <= "0001"; shift_en <= '1'; wait for clk_period; shift_en <= '0';

wait for clk_period*3;

serial_input <= "0010"; shift_en <= '1'; wait for clk_period;

serial_input <= "0011"; wait for clk_period;

serial_input <= "0100"; wait for clk_period;

serial_input <= "1010"; wait for clk_period;

shift_en <= '0'; wait for clk_period;

load_M <= '1'; serial_input <= "0000"; wait for clk_period;

load_M <= '0'; sr_CLR <= '1'; wait for clk_period;
```

sr_CLR <= '0'; M_CLR <= '1'; wait for clk_period*3;

M_CLR <= '0'; wait for clk_period*3; --END UNLOCK SIMULATION--

--BEGIN REPROGRAM SIMULATION USING MASTER CODE-- serial_input <= "0001"; shift_en <= '1'; wait for clk_period; shift_en <= '0'; wait for clk_period*3;

serial_input <= "0010"; shift_en <= '1'; wait for clk_period;

serial_input <= "0011"; wait for clk_period;

serial_input <= "0100"; wait for clk_period;

serial_input <= "1011"; wait for clk_period;

shift_en <= '0'; --Load_M_Reg State load_M <= '1'; serial_input <= "0000"; wait for clk_period;

load_M <= '0'; --Compare State sr_CLR <= '1'; wait for clk_period;

sr_CLR <= '0'; M_CLR <= '1'; --Reprogram State wait for clk_period*3; --ENTERING NEW CODE-- serial_input <= "0101"; shift_en <= '1'; wait for clk_period;

serial_input <= "0110"; wait for clk_period;

serial_input <= "0111"; wait for clk_period;

serial_input <= "1000"; wait for clk_period;

shift_en <= '0'; load_C <= '1'; --Load C_Reg State serial_input <= "0000"; wait for clk_period;

load_C <= '0'; --Idle State sr_CLR <= '1'; wait for clk_period; --BEGIN REPROGRAM SIMULATION USING MASTER CODE--

--BEGIN UNLOCK SIMULATION USING TEMP CODE-- sr_CLR <= '0'; serial_input <= "0101"; shift_en <= '1'; wait for clk_period;

serial_input <= "0110"; wait for clk_period;

serial_input <= "0111"; wait for clk_period;

serial_input <= "1000"; wait for clk_period;

shift_en <= '0'; load_M <= '1'; serial_input <= "0000"; wait for clk_period;

load_M <= '0'; sr_CLR <= '1'; wait for clk_period;

sr_CLR <= '0'; M_CLR <= '1'; wait for clk_period*3;

M_CLR <= '0'; wait for clk_period*3; --END UNLOCK SIMULATION--

```
wait; end process stimulus_process; end Behavioral;
```

Link to code: https://github.com/NduonyiJackUkitetu/Programmable-Locking-Safe

Vhd codes.

toplevel.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL; -- needed for arithmetic
use ieee.math_real.ALL; -- needed for automatic register sizing
library UNISIM; -- needed for the BUFG component
use UNISIM.Vcomponents.ALL;
entity toplevel is
port (mclk : in std_logic; -- FPGA board master clock (100 MHz)
-- interface to keypad
rows : in std_logic_vector(3 downto 0);
cols : out std_logic_vector(3 downto 0);
-- multiplexed seven segment display
segs : out std_logic_vector(0 to 6);
ans : out std_logic_vector(3 downto 0);
LOCK_LED : out std_logic;
UNLOCK_LED : out std_logic;
FAIL_LED : out std_logic;
REPRO_LED : out std_logic );
end toplevel;
architecture Behavioral of toplevel is
-- COMPONENT DECLARATIONS
component compare is
Port ( clk : in STD_LOGIC;
serial_input : in STD_LOGIC_VECTOR(3 downto 0);
shift_en : in STD_LOGIC;
sr_CLR : in STD_LOGIC;
load_C : in STD_LOGIC;
load_M : in STD_LOGIC;
m_CLR : in STD_LOGIC;
disp_output : out STD_LOGIC_VECTOR(15 downto 0);
RP : out STD_LOGIC;
UL : out STD_LOGIC;
delay_CE : in std_logic;
delay_TC : out std_logic; -- temp
delay_CLR : in std_logic;
tc_VAL : in std_logic_vector(16 downto 0));
end component compare;
component key_counter is
Port( clk : in STD_LOGIC;
CE : in STD_LOGIC;
count_CLR : in STD_LOGIC;
TC : out STD_LOGIC;
disp_Update : out STD_LOGIC );
end component key_counter;
component mux7seg is
Port( clk : in STD_LOGIC;
in0 : in std_logic_vector(3 downto 0);
in1 : in std_logic_vector(3 downto 0);
in2 : in std_logic_vector(3 downto 0);
in3 : in std_logic_vector(3 downto 0);
```

```
disp_load : in STD_LOGIC;
disp_CLR : in STD_LOGIC;
seg : out STD_LOGIC_VECTOR(0 to 6);
an : out STD_LOGIC_VECTOR( 3 downto 0) );
end component mux7seg;
component keypad_decoder is
Port( clk : in STD_LOGIC;
row : in STD_LOGIC_VECTOR(3 downto 0);
col : out STD_LOGIC_VECTOR(3 downto 0);
digit_out : out STD_LOGIC_VECTOR(3 downto 0);
Key_Pressed_mp : out STD_LOGIC;
Cancel : out STD_LOGIC );
end component keypad_decoder;
component controller is
Port( clk : in STD_LOGIC;
Key_Pressed_mp : in STD_LOGIC;
Cancel : in STD_LOGIC;
TC : in STD_LOGIC;
RP : in STD_LOGIC;
UL : in STD_LOGIC;
CE : out STD_LOGIC;
count_CLR : out STD_LOGIC;
shift_en : out STD_LOGIC;
sr_CLR : out STD_LOGIC;
disp_CLR : out STD_LOGIC;
load_C : out STD_LOGIC;
load_M : out STD_LOGIC;
```

```
m_CLR : out STD_LOGIC;
LOCK : out STD_LOGIC;
UNLOCK : out STD_LOGIC;
FAIL : out STD_LOGIC;
REPROG : out STD_LOGIC;
delay_CE : out std_logic;
delay_TC : in std_logic; -- temp
delay_CLR : out std_logic;
tc_VAL : out std_logic_vector(16 downto 0));
end component controller;
-- System Clock Divider Signals:
constant SCLK_DIVIDER_VALUE: integer := 50000; -- 1MHz/50000 = 2 kHz, divided again by
flip flop to 1 kHz
constant COUNT_LEN: integer := integer(ceil( log2( real(SCLK_DIVIDER_VALUE) ) ));
signal sclkdiv: unsigned(COUNT_LEN-1 downto 0) := (others => '0'); -- clock divider counter
signal sclk_unbuf: std_logic := '0'; -- unbuffered serial clock
signal sclk: std_logic := '0'; -- internal serial clock
signal disp_en, disp_clear, count_en, terminal_count, count_clear, shift_enable, sr_clear,
L_C, L_M, M_clear, RPro, ULo, Cncl, KPmp, del_CE, del_TC, del_CLR : std_logic := '0';
signal serial : std_logic_vector(3 downto 0);
signal disp_vector : std_logic_vector(15 downto 0);
signal delay_value : std_logic_vector(16 downto 0);
begin
-- Clock buffer for sclk
-- The BUFG component puts the signal onto the FPGA clocking network
Slow_clock_buffer: BUFG
port map (I => sclk_unbuf,
O => sclk );
```

```
-- Divide the 100 MHz clock down to 2 kHz, then toggling a flip flop gives the final
-- 1 kHz system clock
Serial_clock_divider: process(mclk, sclk)
begin
if rising_edge(mclk) then
if sclkdiv = SCLK_DIVIDER_VALUE-1 then
sclkdiv <= (others => '0');
sclk_unbuf <= NOT(sclk_unbuf);
else
sclkdiv <= sclkdiv + 1;
end if;
end if;
end process Serial_clock_divider;
display: mux7seg port map(
clk => sclk,
in0 => disp_vector(15 downto 12),
in1 => disp_vector(11 downto 8),
in2 => disp_vector(7 downto 4),
in3 => disp_vector(3 downto 0),
disp_load => disp_en,
disp_CLR => disp_clear,
seg => segs,
an => ans );
counter: key_counter port map(
clk => sclk,
CE => count_en,
TC => terminal_count,
count_CLR => count_clear,
disp_Update => disp_en );
compares : compare port map(
clk => sclk,
serial_input => serial,
shift_en => shift_enable,
sr_CLR => sr_clear,
load_C => L_C,
load_M => L_M,
m_CLR => M_clear,
disp_output => disp_vector,
RP => RPro,
UL => ULo,
delay_CE => del_CE,
delay_TC => del_TC,
delay_CLR => del_CLR,
tc_VAL => delay_value);
kp_decoder : keypad_decoder port map(
clk => sclk,
row => rows,
col => cols,
digit_out => serial,
Key_Pressed_mp => KPmp,
Cancel => Cncl );
controllers : controller port map(
clk => sclk,
Key_Pressed_mp => KPmp,
Cancel => Cncl,
TC => terminal_count,
RP => RPro,
UL => ULo,
CE => count_en,
count_CLR => count_clear,
shift_en => shift_enable,
sr_CLR => sr_clear,
disp_CLR => disp_clear,
load_C => L_C,
load_M => L_M,
m_CLR => M_clear,
LOCK => LOCK_LED,
UNLOCK => UNLOCK_LED,
FAIL => FAIL_LED,
REPROG => REPRO_LED,
delay_CE => del_CE,
delay_TC => del_TC,
delay_CLR => del_CLR,
tc_VAL => delay_value);
end Behavioral;
```

compare.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity compare is
```

```
Port ( clk : in STD_LOGIC;

serial_input : in STD_LOGIC_VECTOR(3 downto 0);

shift_en : in STD_LOGIC;

sr_CLR : in STD_LOGIC;

load_C : in STD_LOGIC;

load_M : in STD_LOGIC;

M_CLR : in STD_LOGIC;

delay_CLR : in std_logic;

tc_VAL: in std_logic_vector(16 downto 0);

delay_CE : in std_logic;

disp_output : out STD_LOGIC_VECTOR(15 downto 0);

RP : out STD_LOGIC;

UL : out STD_LOGIC;

delay_TC : out std_logic);

end compare;

architecture Behavioral of compare is

constant master_code : std_logic_vector(15 downto 0) := "0001001000110100"; --Master
Code: 1234

constant unlock_key : std_logic_vector(3 downto 0) := "1010"; --Unlock key is A

constant reprogram_key : std_logic_vector(3 downto 0) := "1011";--Reprogram key is B

signal shift_output : std_logic_vector(19 downto 0) := "00000000000000000000";

signal temp_code : std_logic_vector(15 downto 0) := "0001000100100011";
--Reprogramable Code is defaulted to 1234

signal main_reg : std_logic_vector(19 downto 0) := "00000000000000000000";

signal delay : unsigned(16 downto 0) := "00000000000000000"; -- delay timer count

signal terminal_value : unsigned(16 downto 0); -- delay timer terminal value, adjusted via
input tc_VAL

begin
```

```
disp_output <= shift_output(15 downto 0);
--SHIFTS BIT INTO REGISTER WHEN shift_en IS ENABLED--
SHIFT_PROCESS: process(clk, shift_en, serial_input)
begin
if rising_edge(clk) then
if shift_en = '1' then
shift_output <= shift_output(15 downto 0) & serial_input;
end if;
if sr_CLR = '1' then
shift_output <= (others => '0');
end if;
end if;
end process SHIFT_PROCESS;
--LOADS C REGISTER WITH THE CURRENT CODE IN SHIFT REGISTER WHEN load_C IS
ENABLED--
NEW_COMBO: process(clk)
begin
if rising_edge(clk) then
if load_C = '1' then
temp_code <= shift_output(19 downto 4);
end if;
end if;
end process NEW_COMBO;
--LOADS/CLEARS MAIN REGISTER WHEN load_M/M_CLR IS ENABLED--
LOAD_PROC: process(clk, load_M, M_CLR)
begin
if rising_edge(clk) then
if load_M = '1' then

main_reg <= shift_output;

elsif M_CLR = '1' then

main_reg <= (others => '0');

end if;

end if;

end process LOAD_PROC;

-- ADJUSTABLE TIMER USED TO DELAY TRANSITION INTO COMPARE STATE, AVOID TIMING
CONFLICT --

delay_timer: process(clk, delay_CE, delay_CLR, tc_VAL, terminal_value, delay)

begin

terminal_value <= unsigned(tc_VAL);

if rising_edge(clk) then

if delay_CLR = '1' then

delay <= "00000000000000000";

else

if delay_CE = '1' then

if delay = terminal_value then

delay <= terminal_value;

else

delay <= delay + 1;

end if;

end if;

end if;

end if;

if delay = terminal_value then delay_TC <= '1'; else delay_TC <= '0';

end if;
end process delay_timer;

--COMPARES THE VALUE OF THE CODE LOADED INTO THE MAIN REGISTER AND THE
MASTER/TEMP

--CODE THEN IF CORRECT CODE IF A OR B IS PRESSED SENDS UNLOCK OR REPROGRAM.--

COMPARE: process(clk, main_reg)

begin

if rising_edge(clk) then

if (main_reg(19 downto 4) = master_code) or (main_reg(19 downto 4) = temp_code) then

if main_reg(3 downto 0) = unlock_key then

UL <= '1';

elsif main_reg(3 downto 0) = reprogram_key then

RP <= '1';

end if;

else

UL <= '0';

RP <= '0';

end if;

end if;

end process COMPARE;

end Behavioral;
```

controller.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity controller is

Port ( clk : in STD_LOGIC;

Key_Pressed_mp : in STD_LOGIC;
Cancel : in STD_LOGIC;
TC : in STD_LOGIC;
RP : in STD_LOGIC;
UL : in STD_LOGIC;
delay_TC : in STD_LOGIC;
CE : out STD_LOGIC;
count_CLR : out STD_LOGIC;
shift_en : out STD_LOGIC;
sr_CLR : out STD_LOGIC;
disp_CLR : out STD_LOGIC;
load_C : out STD_LOGIC;
load_M : out STD_LOGIC;
m_CLR : out STD_LOGIC;
LOCK : out STD_LOGIC;
UNLOCK : out STD_LOGIC;
FAIL : out STD_LOGIC;
REPROG : out STD_LOGIC;
delay_CE : out STD_LOGIC;
delay_CLR : out STD_LOGIC;
tc_VAL : out STD_LOGIC_VECTOR(16 downto 0));
end controller;
architecture Behavioral of controller is
type state_type is (idle, typing, ty_take_sample, load_M_reg,
compare, unlocked, failure, reprogram,
rp_take_sample, load_C_reg);
signal current_state, next_state: state_type := idle;
begin
```

```
--NEXT STATE AND OUTPUT LOGIC--
COMBONATIONAL_LOGIC:
process(clk, current_state, Key_Pressed_mp, Cancel, TC, RP, UL, delay_TC)
begin
next_state <= current_state;
CE <= '0';
count_CLR <= '0';
shift_en <= '0';
sr_CLR <= '0';
disp_CLR <= '0';
load_C <= '0';
load_M <= '0';
m_CLR <= '0';
LOCK <= '1';
UNLOCK <= '0';
FAIL <= '0';
REPROG <= '0';
delay_CLR <= '0';
delay_CE <= '0';
tc_VAL <= "11111111111111111";
case current_state is
when idle =>
sr_CLR <= '1';
disp_CLR <= '1';
count_CLR <= '1';
delay_CE <= '0';
delay_CLR <= '1';
```

```
if Key_Pressed_mp = '1' then
next_state <= ty_take_sample;
end if;
when ty_take_sample =>
shift_en <= '1';
CE <= '1';
if Cancel = '1' then
next_state <= idle;
else
next_state <= typing;
end if;
when typing =>
if Cancel = '1' then
next_state <= idle;
elsif TC = '1' then
next_state <= load_M_reg;
elsif Key_Pressed_mp = '1' then
next_state <= ty_take_sample;
end if;
when load_M_reg =>
load_M <= '1';
delay_CLR <= '0';
delay_CE <= '1';
tc_VAL <= "0000000000001111"; -- delay timer: .016 sec
if delay_TC = '1' then
next_state <= compare;
end if;
```

```
when compare =>
delay_CE <= '0';
delay_CLR <= '1';
sr_CLR <= '1';
count_CLR <= '1';
disp_CLR <= '1';
if RP = '1' then
next_state <= reprogram;
elsif UL = '1' then
next_state <= unlocked;
else
next_state <= failure;
end if;
when unlocked =>
m_CLR <= '1';
UNLOCK <= '1';
LOCK <= '0';
delay_CE <= '1';
delay_CLR <= '0';
if delay_TC = '1' then -- hold unlock for roughly 2 min
next_state <= idle;
end if;
when failure =>
m_CLR <= '1';
FAIL <= '1';
delay_CE <= '1';
delay_CLR <= '0';
```

```
if delay_TC = '1' then -- hold failure for roughly 2 min
next_state <= idle;
end if;
when reprogram =>
m_CLR <= '1';
REPROG <= '1';
if Cancel = '1' then
next_state <= idle;
elsif TC = '1' then
next_state <= load_C_reg;
elsif Key_Pressed_mp = '1' then
next_state <= rp_take_sample;
end if;
when rp_take_sample =>
shift_en <= '1';
CE <= '1';
if Cancel = '1' then
next_state <= idle;
else
next_state <= reprogram;
end if;
when load_C_reg =>
load_C <= '1';
next_state <= idle;
end case;
end process COMBONATIONAL_LOGIC;
--UPDATES STATE TO NEXT_STATE--
```

```
UPDATE_STATE : process(clk)
begin
if rising_edge(clk) then
current_state <= next_state;
end if;
end process UPDATE_STATE;
end Behavioral;
```

debouncer.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity debouncer is
Port ( clk : in STD_LOGIC; -- assumed 1kHz
deb_in : in STD_LOGIC;
deb_out : out STD_LOGIC);
end debouncer;
architecture Behavioral of debouncer is
-- counter signals
signal count: unsigned(3 downto 0) := "0100";
signal CE: std_logic := '0';
signal reset: std_logic := '1';
signal tc: std_logic;
signal T: std_logic := '0';-- t-flop
begin
deb: process(clk, count, CE, deb_in, T)
begin
```

```
if rising_edge(clk) then -- counter mechanics
if reset = '1' then
count <= "0100";
else
if CE = '1' then
if count = "0000" then
count <= "0100";
else
count <= count - 1;
end if;
end if;
end if;
T <= T xor tc; -- t flip flop mechanics
end if;
if count = "0000" and CE = '1' then -- combinational logic
tc <= '1';
else
tc <= '0';
end if;
CE <= deb_in xor T;
reset <= not(deb_in xor T);
deb_out <= T;
end process deb;
end Behavioral;
```

key_counter.vhd

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity key_counter is
Port ( clk : in STD_LOGIC;
CE : in STD_LOGIC;
count_CLR : in STD_LOGIC;
TC : out STD_LOGIC;
disp_Update : out STD_LOGIC);
end key_counter;
architecture Behavioral of key_counter is
signal count: unsigned (2 downto 0) := "000";
begin
increment: process(clk, count)
begin
if rising_edge(clk) then
if count_CLR = '1' then -- reset counter
count <= "000";
else
if CE = '1' and count < "101" then -- if count enabled & count <5, increment
count <= count + 1;
end if;
end if;
end if;
if count = "101" then -- assert TC @ 4 (after 5 keypresses)
TC <= '1';
disp_Update <= '0';
else
```

```
TC <= '0';
disp_Update <= '1';
end if;
end process increment;
end Behavioral;
```

keypad_decoder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity keypad_decoder is
Port ( clk : in STD_LOGIC; -- assumed 1000 Hz
row : in STD_LOGIC_VECTOR (3 downto 0);
col : out STD_LOGIC_VECTOR (3 downto 0);
digit_out : out STD_LOGIC_VECTOR (3 downto 0);
Key_Pressed_mp : out STD_LOGIC;
Cancel : out STD_LOGIC);
end keypad_decoder;
architecture Behavioral of keypad_decoder is
component debouncer is
Port ( clk : in STD_LOGIC;
deb_in : in STD_LOGIC;
deb_out : out STD_LOGIC);
end component;
signal div_count: unsigned(3 downto 0) := "1001"; -- clk divider count
signal CE: STD_LOGIC; -- divided clk signal
signal dec_count: unsigned(3 downto 0) := "0000"; -- decoder counter
signal dec_count_en: STD_LOGIC := '1'; -- decoder counter enable

signal drive_col: unsigned(1 downto 0); -- column driven addresss

signal read_row: unsigned(1 downto 0); -- row to read from address

signal row_status: STD_LOGIC; -- signal held on row being read

signal digit_decode: unsigned(3 downto 0); -- decoded output of dec_count

signal ss_sync: STD_LOGIC_VECTOR(1 downto 0) := "00"; -- monopulse logic signals

signal button_held: STD_LOGIC := '0';

signal ss: STD_LOGIC;

begin

deb: debouncer port map(

clk=>clk, deb_in=>row_status, deb_out=>dec_count_en);

key_press: process(clk, row, ss_sync, dec_count_en, digit_decode, ss) -- adapted from Prof.
Hansen's lab4_shell

begin

if dec_count_en = '0' and digit_decode /= "0000" then button_held <= '1';

else button_held <= '0';

end if;

if rising_edge(clk) then -- monopulse send_scan when a key is pressed

ss_sync <= button_held & ss_sync(1);

end if;

ss <= ss_sync(1) and not(ss_sync(0));

Key_Pressed_mp <= ss;

end process key_press;

take_sample: process(clk, div_count) -- clk divider to drive at 100 Hz

begin

if rising_edge(clk) then

if div_count = "0000" then div_count <= "1001";
```

```
else div_count <= div_count - 1;
end if;
end if;
if div_count = "0000" then CE <= '1';
else CE <= '0';
end if;
end process take_sample;
dec_driver: process(clk, dec_count) -- decoder driver
begin
if rising_edge(clk) then
Cancel <= '0';
if CE = '1' then
if dec_count_en = '1' then
dec_count <= dec_count + 1;
digit_decode <= "0000";
else
if dec_count = "0000" then digit_decode <= "0001"; -- c0,r0: 1
elsif dec_count = "0001" then digit_decode <= "0100"; -- c0,r1: 4
elsif dec_count = "0010" then digit_decode <= "0111"; -- c0,r2: 7
elsif dec_count = "0011" then digit_decode <= "0000"; -- c0,r3: 0
elsif dec_count = "0100" then digit_decode <= "0010"; -- c1,r0: 2
elsif dec_count = "0101" then digit_decode <= "0101"; -- c1,r1: 5
elsif dec_count = "0110" then digit_decode <= "1000"; -- c1,r2: 8
elsif dec_count = "0111" then digit_decode <= "1111"; -- c1,r3: F
elsif dec_count = "1000" then digit_decode <= "0011"; -- c2,r0: 3
elsif dec_count = "1001" then digit_decode <= "0110"; -- c2,r1: 6
elsif dec_count = "1010" then digit_decode <= "1001"; -- c2,r2: 9
elsif dec_count = "1011" then digit_decode <= "1110"; -- c2,r3: E
elsif dec_count = "1100" then digit_decode <= "1010"; -- c3,r0: A
elsif dec_count = "1101" then digit_decode <= "1011"; -- c3,r1: B
elsif dec_count = "1110" then Cancel <= '1'; digit_decode <= "1100";-- c3,r2 -- C: cancel
else digit_decode <= "1101";
end if;
end if;
end if;
digit_out <= std_logic_vector(digit_decode);
end if;
drive_col <= dec_count(3 downto 2); -- rename highest and lowest order bits
read_row <= dec_count(1 downto 0);
end process dec_driver;
drive_mux: process(drive_col) -- changing dec_count's highest order bits to one cold
begin
if drive_col = "00" then col <= "0111";
elsif drive_col = "01" then col <= "1011";
elsif drive_col = "10" then col <= "1101";
elsif drive_col = "11" then col <= "1110";
else col <= "1110"; -- remove latch, could have eliminated last case
end if;
end process drive_mux;
read_dec: process(read_row, row) -- changing dec_count's lowest order bits to one cold
begin
if read_row = "00" then row_status <= row(3);
elsif read_row = "01" then row_status <= row(2);
elsif read_row = "10" then row_status <= row(1);
elsif read_row = "11" then row_status <= row(0);
else row_status <= row(0); -- remove latch, could have eliminated last case
end if;
end process read_dec;
end Behavioral;
```

mux7seg.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
entity mux7seg is
Port ( clk : in STD_LOGIC; -- runs on a 1kHz clock
in0, in1, in2, in3 : in STD_LOGIC_VECTOR (3 downto 0); -- digits
seg : out STD_LOGIC_VECTOR(0 to 6); -- segments (a...g)
an : out STD_LOGIC_VECTOR (3 downto 0); -- anodes
disp_load : in std_logic;
disp_CLR : in std_logic);
end mux7seg;
architecture Behavioral of mux7seg is
constant NCLKDIV: integer := 2; -- 1 kHz / 2^2 = 250 Hz
constant MAXCLKDIV: integer := 2**NCLKDIV-1; -- max count of clock divider
signal cdcount: unsigned(NCLKDIV-1 downto 0); -- clock divider counter register
signal CE : std_logic; -- clock enable
signal adcount : unsigned(1 downto 0) := "00"; -- anode / mux selector count
signal anb: std_logic_vector(3 downto 0);
signal muxy : std_logic_vector(3 downto 0); -- mux output
signal segh : std_logic_vector(0 to 6); -- segments (high true)
signal y0 : std_logic_vector(3 downto 0):= "0000";

signal y1 : std_logic_vector(3 downto 0):= "0000";

signal y2 : std_logic_vector(3 downto 0) := "0000";

signal y3 : std_logic_vector(3 downto 0) := "0000";

begin

-- Clock divider sets the rate at which the display hops from one digit to the next. A larger
value of

-- MAXCLKDIV results in a slower clock-enable (CE)

ClockDivider:

process(clk)

begin

if rising_edge(clk) then

if cdcount < MAXCLKDIV then

CE <= '0';

cdcount <= cdcount+1;

else CE <= '1';

cdcount <= (others => '0');

end if;

end if;

end process ClockDivider;

--LOAD OR CLEAR THE DISPLAY WITH VALUES FROM ins--

d_reg: process(clk, disp_load, disp_CLR)

begin

if rising_edge(clk) then

if disp_CLR = '1' then

y0 <= "0000";

y1 <= "0000";
```

```
y2 <= "0000";
y3 <= "0000";
else
if (disp_load) = '1' then
y0 <= in0;
y1 <= in1;
y2 <= in2;
y3 <= in3;
else
y0 <= y0;
y1 <= y1;
y2 <= y2;
y3 <= y3;
end if;
end if;
end if;
end process d_reg;
--SEQUENTIALLY DRIVE THE ANODES LOAD TO CYCLE THROUGH THEM--
AnodeDriver:
process(clk, adcount)
begin
if rising_edge(clk) then
if CE='1' then
adcount <= adcount + 1;
end if;
end if;
case adcount is
```

```
when "00" => anb <= "1110";
when "01" => anb <= "1101";
when "10" => anb <= "1011";
when "11" => anb <= "0111";
when others => anb <= "1111";
end case;
end process AnodeDriver;
an <= anb or "0000"; --- blank digit 3
--CHANGE SEGMENTS BASED ON WHICH ANODE IS BEING DRIVEN LOW--
Multiplexer:
process(adcount, y0, y1, y2, y3)
begin
case adcount is
when "00" => muxy <= y0;
when "01" => muxy <= y1;
when "10" => muxy <= y2;
when "11" => muxy <= y3;
when others => muxy <= x"0";
end case;
end process Multiplexer;
-- Seven segment decoder
with muxy select segh <=
"1111110" when x"0", -- active-high definitions
"0110000" when x"1",
"1101101" when x"2",
"1111001" when x"3",
"0110011" when x"4",
"1011011" when x"5",
"1011111" when x"6",
"1110000" when x"7",
"1111111" when x"8",
"1111011" when x"9",
"1110111" when x"a",
"0011111" when x"b",
"1001110" when x"c",
"0111101" when x"d",
"1001111" when x"e",
"1000111" when x"f",
"0000000" when others;
seg <= not(segh); -- Convert to active-low
end Behavioral;
```

xdc. file

Basys3_Master.xdc

```
set_property PACKAGE_PIN W5 [get_ports mclk]
set_property IOSTANDARD LVCMOS33 [get_ports mclk]
create_clock -period 20.000 -name sys_clk_pin -waveform {0.000 10.000} -add [get_ports
mclk]

set_property PACKAGE_PIN U16 [get_ports {FAIL_LED}]
set_property IOSTANDARD LVCMOS33 [get_ports {FAIL_LED}]
set_property PACKAGE_PIN E19 [get_ports {LOCK_LED}]
set_property IOSTANDARD LVCMOS33 [get_ports {LOCK_LED}]
set_property PACKAGE_PIN U19 [get_ports {UNLOCK_LED}]
set_property IOSTANDARD LVCMOS33 [get_ports {UNLOCK_LED}]
set_property PACKAGE_PIN V19 [get_ports {REPRO_LED}]
set_property IOSTANDARD LVCMOS33 [get_ports {REPRO_LED}]

set_property PACKAGE_PIN W7 [get_ports {segs[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segs[0]}]
#Bank = 34, Pin name = , Sch name = CB
set_property PACKAGE_PIN W6 [get_ports {segs[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segs[1]}]
#Bank = 34, Pin name = , Sch name = CC
set_property PACKAGE_PIN U8 [get_ports {segs[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segs[2]}]
#Bank = 34, Pin name = , Sch name = CD
set_property PACKAGE_PIN V8 [get_ports {segs[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segs[3]}]
#Bank = 34, Pin name = , Sch name = CE
set_property PACKAGE_PIN U5 [get_ports {segs[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segs[4]}]
#Bank = 34, Pin name = , Sch name = CF
set_property PACKAGE_PIN V5 [get_ports {segs[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segs[5]}]
#Bank = 34, Pin name = , Sch name = CG
set_property PACKAGE_PIN U7 [get_ports {segs[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segs[6]}]
#Bank = 34, Pin name = , Sch name = DP
#set_property PACKAGE_PIN V7 [get_ports {dp}]
#set_property IOSTANDARD LVCMOS33 [get_ports {dp}]
```

```
#Bank = 34, Pin name = , Sch name = AN0

set_property PACKAGE_PIN U2 [get_ports {ans[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {ans[3]}]

#Bank = 34, Pin name = , Sch name = AN1

set_property PACKAGE_PIN U4 [get_ports {ans[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {ans[2]}]

#Bank = 34, Pin name = , Sch name = AN2

set_property PACKAGE_PIN V4 [get_ports {ans[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {ans[1]}]

#Bank = 34, Pin name = , Sch name = AN3

set_property PACKAGE_PIN W4 [get_ports {ans[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {ans[0]}]


#Pmod Header JB

#Bank = 15, Pin name = IO_L15N_T2_DQS_ADV_B_15, Sch name = JB1

set_property PACKAGE_PIN A14 [get_ports {cols[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cols[0]}]

##Bank = 14, Pin name = IO_L13P_T2_MRCC_14, Sch name = JB2

set_property PACKAGE_PIN A16 [get_ports {cols[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cols[1]}]

##Bank = 14, Pin name = IO_L21N_T3_DQS_A06_D22_14, Sch name = JB3

set_property PACKAGE_PIN B15 [get_ports {cols[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cols[2]}]

##Bank = CONFIG, Pin name = IO_L16P_T2_CSI_B_14, Sch name = JB4

set_property PACKAGE_PIN B16 [get_ports {cols[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cols[3]}]

#Bank = 15, Pin name = IO_25_15, Sch name = JB7
```

```
set_property PACKAGE_PIN A15 [get_ports {rows[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rows[0]}]
#Bank = CONFIG, Pin name = IO_L15P_T2_DQS_RWR_B_14, Sch name = JB8
set_property PACKAGE_PIN A17 [get_ports {rows[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rows[1]}]
#Bank = 14, Pin name = IO_L24P_T3_A01_D17_14, Sch name = JB9
set_property PACKAGE_PIN C15 [get_ports {rows[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rows[2]}]
#Bank = 14, Pin name = IO_L19N_T3_A09_D25_VREF_14, Sch name = JB10
set_property PACKAGE_PIN C16 [get_ports {rows[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rows[3]}]
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```