FAU

COLLEGE OF ENGINEERING
& COMPUTER SCIENCE
Florida Atlantic University

Digital Lock System

Department of Electrical Engineering
Introduction to Microprocessor System – CDA 3331C

Daniel Leach

Saturday, April 20, 2019

Table of Contents

Abstract

The objective of this design project is to design and implement a digital lock system using a seven-segment display. This project uses a common cathode 4-digit seven segment display. The onboard potentiometer, a tactile switch, and 4 LEDs of the MPLAB Xpress Evaluation board.

Project Specification Changes

The original project called for the use of timers and interrupts to input the combination of the digital lock system. I decided that using the tactile switch would be a better solution. Having a timed input could potentially lead to user error. Other changes included how a new combination would be set and what was to be displayed on the seven-segment display. The original project wanted a long press while the lock is open to indicate a new combination is to be set. I decided that using the potentiometers position and the tactile switch will indicate if there is a new combination to be set. The display changes include the use of scrolling text. Instead of displaying "NEW" I decided it should scroll text displaying "New Combo" as well as indicating if a combo has been set with "Combo Set" followed by the onboard LED's blinking 3 times.

Design Methodology or Theory

Using the Structured Design Method, we must break the project down into small parts that can easily be completed to achieve our main goal of creating this digital lock system.

First is the seven-segment display, we will use this to display our locking systems current number. We will also use this to send messages to the user to announce if the combo that is entered is correct, denied, if the lock is opened, closed, and if a new combo is to be set.

The next thing that needs to be accounted for is the potentiometer. This is located on the Xpress board and will be used to control the scrolling of numbers and determine if the lock needs to be reset or have a new combination entered.

The third item on the list is the tactile switch also located on the Xpress board. This button will be used to confirm the sequence of numbers entered. It will be used along with the potentiometer to help confirm if a new combination is to be entered. This will also be used to close the lock if the door is opened.

The onboard LEDs are the last components to this digital lock system. These will be used to keep track of the current combination that is entered. As a digit is entered the light will indicate if it was accepted. The LEDs will also be used as indication that a new combo is being entered as well if the new combo is set.

Implementation

First, we should look at the seven-segment display and how it works with the MPLAB Xpress Evaluation board. The project uses a common cathode seven-segment display. The one provided to us was created by Perry Weinthal, FAU's College of Engineering and Computer Science laboratory manager. This seven-segment display use Ports B and C of the PIC microcontroller. Referring to the diagram below you can see that all LEDs on the display are tied to each other. Each digit is controlled by a single pin, since there are four digits then there a 4 control pins. These pins on our board will be controlled by RB5, RC4, RC5 and RC7. To display any character, we simply just need to toggle each control pin while changing what is to be displayed to get the desired result.

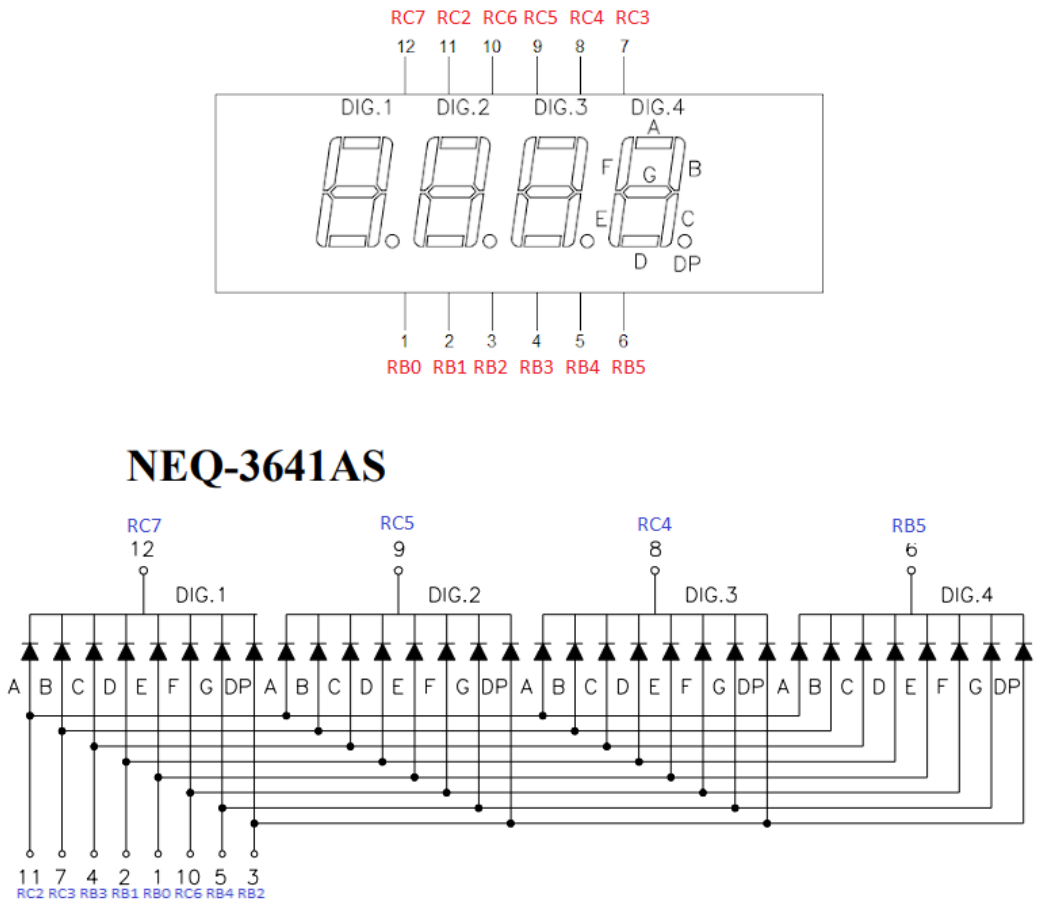


Figure 1: Seven Segment Display Wire Diagram

The pins RB0 – RB4 and RC2, RC3 and RC6 are connected to the LEDs on the seven-segment display. setting these High or Low will determine what number or letter will be displayed.

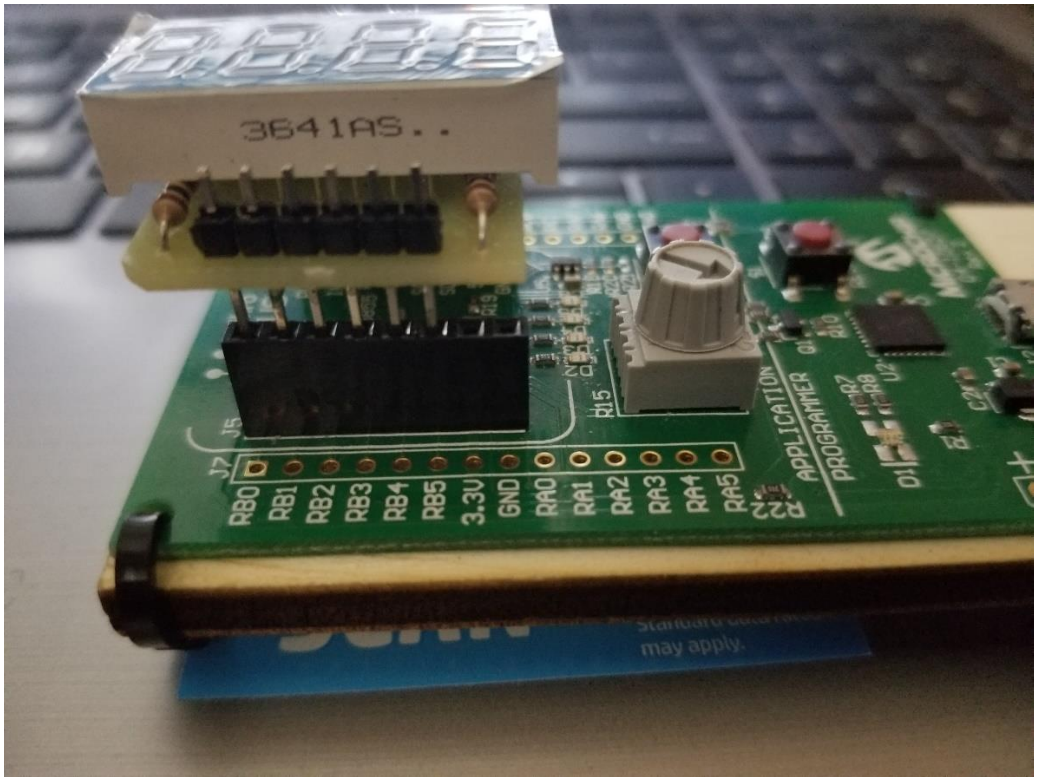


Figure 2: Port B on Xpress Board

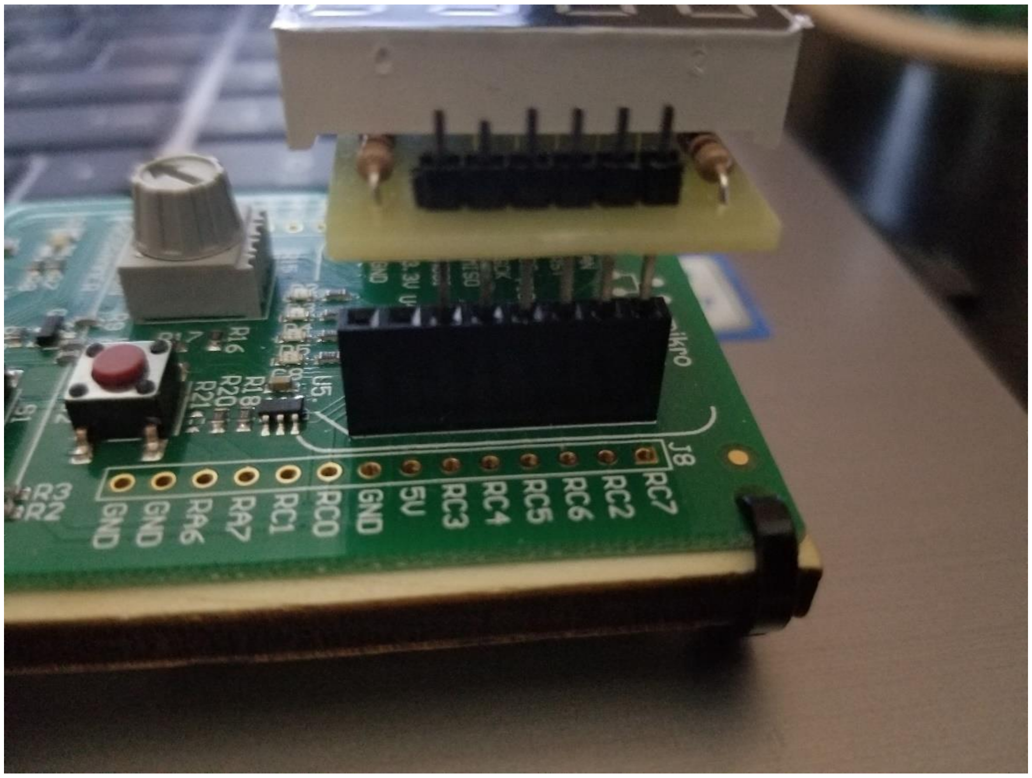


Figure 3: Port C on Xpress Board

Next, we should look at the Potentiometer on the Xpress board to find out how we will be using this to control the numbers of the seven-segment display. Potentiometer is an analog peripheral that we will use as input. The potentiometer is connected to pin RA4 on the microcontroller. For us to be able to read this input we will need to convert this to a digital signal. For this we will use the ADCC to get a signal that the microprocessor will be able to understand. The ADCC has a 10 bit register this allow us to get a value between 0 and 1023. For this project we will be reducing this value to 63. This will be done in code by using logical bitwise operators in C and a little math to get the desired value.

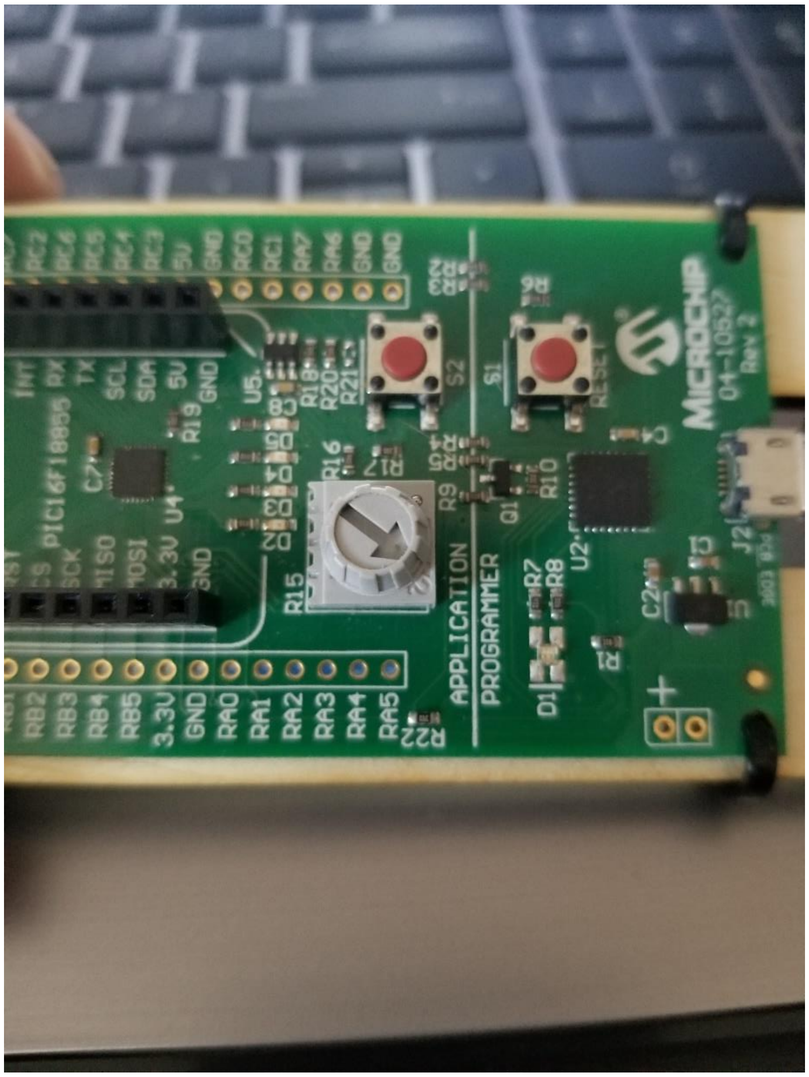


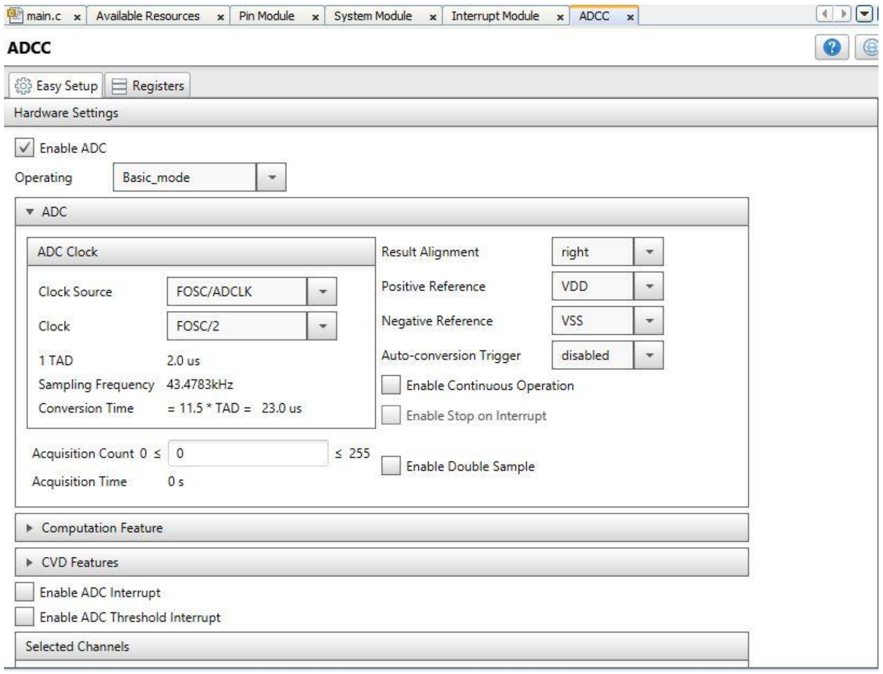
Figure 4: Xpress board Potentiometer & Button

The tactile switch is located on pin RA5 of the Xpress board. In the Pin Manager set this as an input Pin Module. For debouncing we will use C code to solve the problem.

The LEDs on the Xpress board are located just left of the potentiometer and tactile switch. The LEDs are connected to pins RA0 – RA3.

Now that the seven-segment display is connected, and we have an idea of how each peripheral works on the Xpress board we can start by creating a project in MPLAB X IDE. Once the project is created you will want to open the MCC. Add the ADCC that is in the Device Resource list to the Project Resource list by double clicking it. It will automatically be listed under Peripherals in the Project Resource list.

The ADCC also known as the Analog-to-Digital Converter with Computation will be the first thing that needs to be setup in the MCC. Click the ADCC listed under Peripherals in the Project Resource list. A window will pop up that will have many options for the ADCC. For this project you will want to make sure that the ADC is enabled and running in Basic_Mode. Use the default clock settings and have the Result Alignment set to Right. Refer to Figure 5 for help.



ADCC

Hardware Settings

☑ Enable ADC

Operating: Basic_mode

ADC

ADC Clock

Clock Source: FOSC/ADCLK

Clock: FOSC/2

1 TAD 2.0 us

Sampling Frequency 43.4783kHz

Conversion Time = 11.5 * TAD = 23.0 us

Result Alignment: right

Positive Reference: VDD

Negative Reference: VSS

Auto-conversion Trigger: disabled

☐ Enable Continuous Operation

☐ Enable Stop on Interrupt

Acquisition Count 0 ≤ 0 ≤ 255

Acquisition Time 0 s

☐ Enable Double Sample

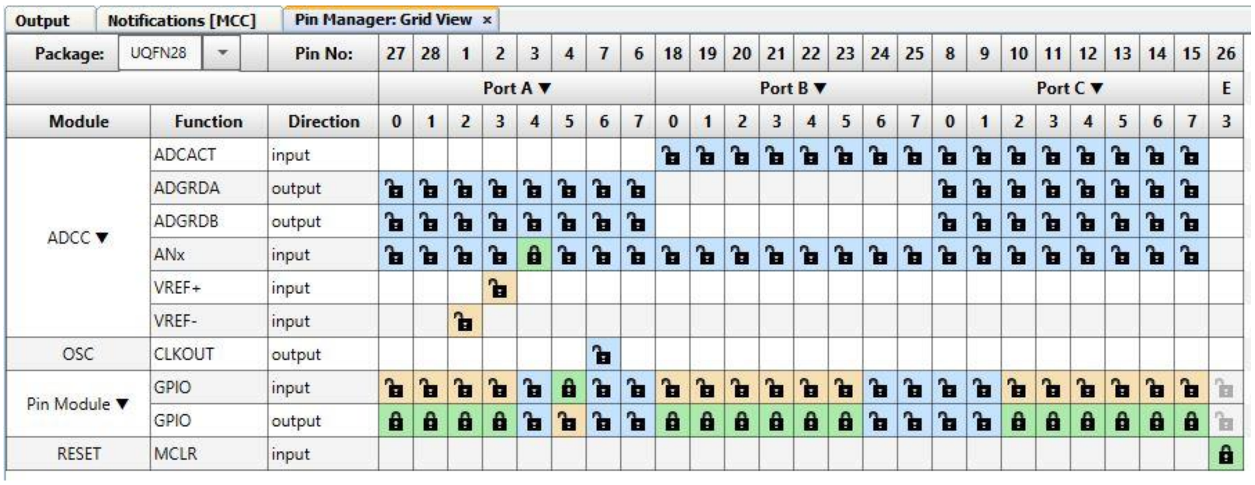
☐ Enable ADC Interrupt

☐ Enable ADC Threshold Interrupt

Figure 5: ADCC Module

The next thing to setup in the MCC is the Pin Manger. This is where we decide what pins need to be assigned for this project. for the seven-segment display we are using pins RB0-Rb5, RC2-RC7. The control pins for the seven-segment display are RB5, RC4, RC5 and RC7. The potentiometer is connected to the RA4. The tactile switch is connected to RA5 and the LEDs are connected to RA0- RA3.

When looking at the Pin Manager that is in the MCC there you can see all the appropriate pins that have to be selected for this project in the figure below. Each pin has been referenced prior to this point.

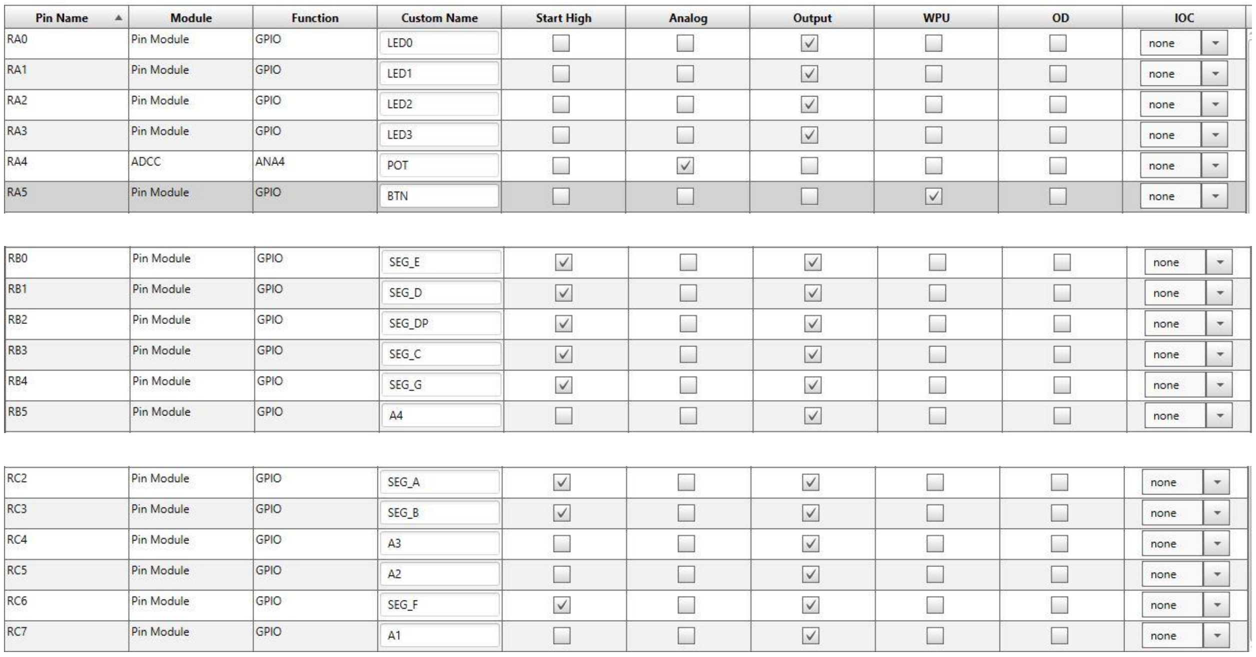


Output | Notifications [MCC] | Pin Manager: Grid View ×

Package:	UQFN28	Pin No:	27	28	1	2	3	4	7	6	18	19	20	21	22	23	24	25	8	9	10	11	12	13	14	15	26
			Port A ▼								Port B ▼								Port C ▼								E
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	3
ADCC ▼	ADCACT	input									🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	
	ADGRDA	output	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓									🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	
	ADGRDB	output	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓									🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	
	ANx	input	🔓	🔓	🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	
	VREF+	input				🔓																					
	VREF-	input			🔓																						
OSC	CLKOUT	output							🔓																		
Pin Module ▼	GPIO	input	🔓	🔓	🔓	🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓
	GPIO	output	🔒	🔒	🔒	🔒	🔓	🔓	🔓	🔓	🔒	🔒	🔒	🔒	🔒	🔒	🔓	🔓	🔓	🔓	🔒	🔒	🔒	🔒	🔒	🔒	🔓
RESET	MCLR	input																									🔒

Figure 6: Pin Manager

The next thing to do is set up variable that will be used to reference these pins. I chose to go with simple variable that resemble what they control.



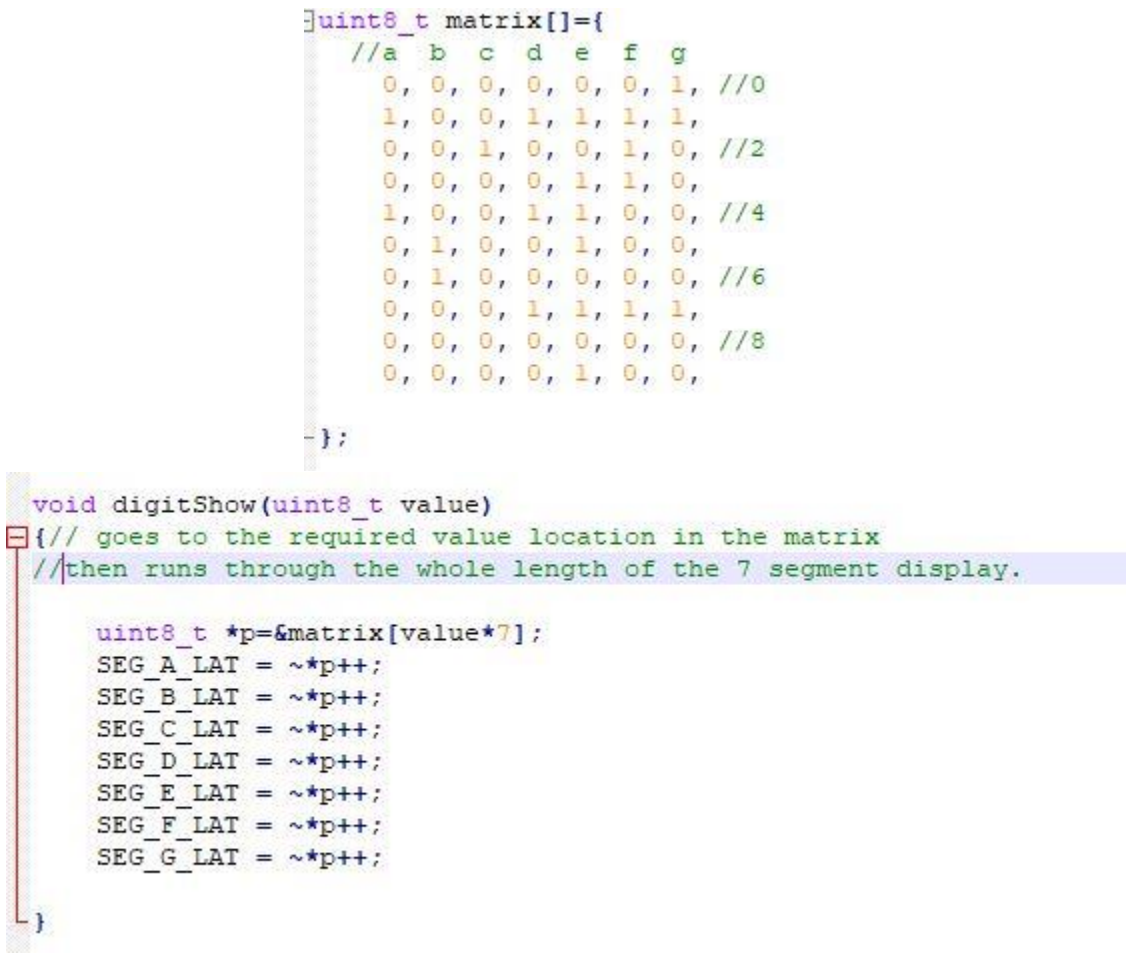
Pin Name ▲	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA0	Pin Module	GPIO	LED0			✓			none
RA1	Pin Module	GPIO	LED1			✓			none
RA2	Pin Module	GPIO	LED2			✓			none
RA3	Pin Module	GPIO	LED3			✓			none
RA4	ADCC	ANA4	POT		✓				none
RA5	Pin Module	GPIO	BTN				✓		none
RB0	Pin Module	GPIO	SEG_E	✓		✓			none
RB1	Pin Module	GPIO	SEG_D	✓		✓			none
RB2	Pin Module	GPIO	SEG_DP	✓		✓			none
RB3	Pin Module	GPIO	SEG_C	✓		✓			none
RB4	Pin Module	GPIO	SEG_G	✓		✓			none
RB5	Pin Module	GPIO	A4			✓			none
RC2	Pin Module	GPIO	SEG_A	✓		✓			none
RC3	Pin Module	GPIO	SEG_B	✓		✓			none
RC4	Pin Module	GPIO	A3			✓			none
RC5	Pin Module	GPIO	A2			✓			none
RC6	Pin Module	GPIO	SEG_F	✓		✓			none
RC7	Pin Module	GPIO	A1			✓			none

Figure 7: Pin Module

Now that all the ADCC, Pin Manager and Pin Module have been set up all that is left to do is to click that Generate button located in the Project Resource list. Once the MCC has finished generating the appropriate files for this project we will need to open the main.c file located in the projects folder under Source Files. This is where will write our main program for this project.

Good coding practices are very important when writing large programs. Code can get confusing very fast if you don't take precautions prior to starting. So, to help us out what you want to do is get rid of all the pre-generated comments that where left by the MCC in the main.c file. This will help reduce clutter in our code. We only want to see our comments, the ones that make sense to the program that is being written. Once all the generated comments are deleted. Its time to finally write the code. I won't be going over every aspect of the code, but I will be focusing on some very important topics that will help you understand how this program works.

Using the example code that is given in Di Jasio's *"In 10 Lines of Code"* you can see a global unsigned 8-bit integer array called matrix. This array holds the values of 0 and 1. In a void function called digitShow the matrix array is set to a pointer called p. Each 0 and 1 is designated to the LEDs on the seven-segment display. unlike the code in the book we are using a common cathode display and the 1's and 0's need to be inverted. You can do this in two different ways, one is by changing the 1's to 0 and the 0's to 1 in the matrix array or you can remove the tilde bitwise operator in the digitShow function to get the same result.



```
uint8_t matrix[]={
  //a  b  c  d  e  f  g
    0, 0, 0, 0, 0, 0, 1, //0
    1, 0, 0, 1, 1, 1, 1,
    0, 0, 1, 0, 0, 1, 0, //2
    0, 0, 0, 0, 1, 1, 0,
    1, 0, 0, 1, 1, 0, 0, //4
    0, 1, 0, 0, 1, 0, 0,
    0, 1, 0, 0, 0, 0, 0, //6
    0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 0, 0, 0, //8
    0, 0, 0, 0, 1, 0, 0,

};
```

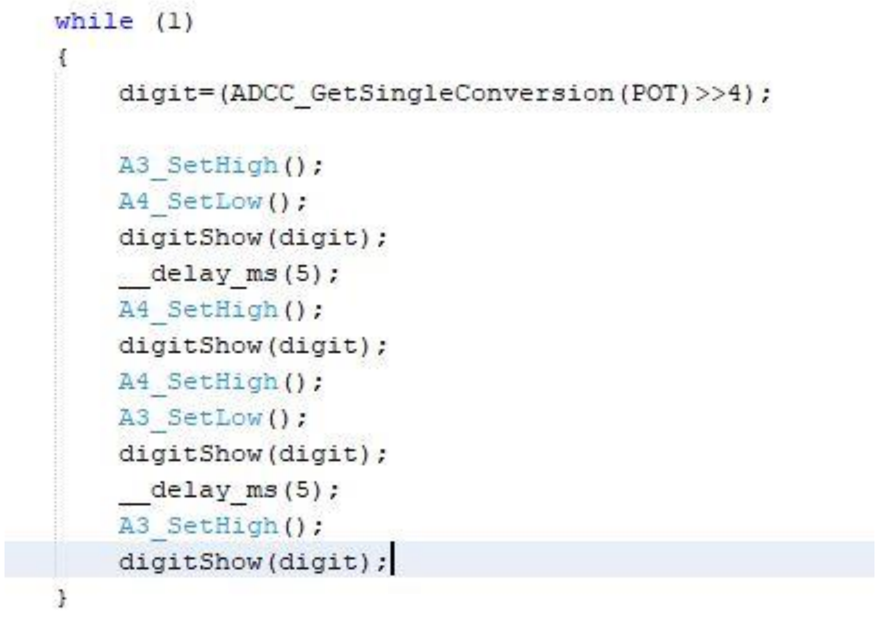
```
void digitShow(uint8_t value)
{// goes to the required value location in the matrix
//then runs through the whole length of the 7 segment display.

    uint8_t *p=&matrix[value*7];
    SEG_A_LAT = ~*p++;
    SEG_B_LAT = ~*p++;
    SEG_C_LAT = ~*p++;
    SEG_D_LAT = ~*p++;
    SEG_E_LAT = ~*p++;
    SEG_F_LAT = ~*p++;
    SEG_G_LAT = ~*p++;

}
```

Figure 8: Matrix array & digitShow Function

To get the values that will be passed through the function called digitShow we have to look at the ADCC. We are using the ADCC's function called ADCC_GetSingleCoversion and passing the potentiometer variable that is named "POT" in to this function. Since we know that the ADCC holds a max value of 1024 in a 10 bit register the next logical thing to do is to reduce this value to 64. The way to do this in code is to use a bitwise shift operator and shift the bits by 4 this way we only look at the bits needed. Assign this to a variable and pass it through the function digitShow.



```
while (1)
{
    digit=(ADCC_GetSingleConversion(POT)>>4);

    A3_SetHigh();
    A4_SetLow();
    digitShow(digit);
    __delay_ms(5);
    A4_SetHigh();
    digitShow(digit);
    A4_SetHigh();
    A3_SetLow();
    digitShow(digit);
    __delay_ms(5);
    A3_SetHigh();
    digitShow(digit);
}
```

Figure 9: Display Digit Code

If you were to display this on the seven-segment display you would notice that the value will count to 9 and then you would start seeing garbage characters displayed on the screen until the potentiometer gets to the end. The ADCC thinks there is 63 lines in your matrix array when in fact there are only 10 lines. Each line corresponding to a digit that would be displayed on the seven-segment display.

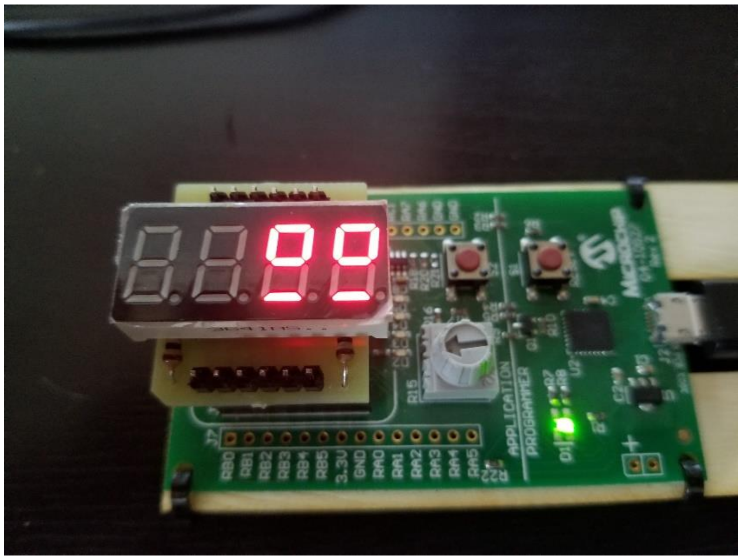


Figure 10: Garbage Displayed

You probably notice that the values that are being displayed is on all the screens. With this solution we will solve both problems. The values need to be split so that tens digit will increment with every 10 steps of the ones digit. This is were the little math comes into play. Looking at the number 64 or in this case 63, since we start our count at 0, you can use division and the modulus operator to find the values we need. For the ones place we need to take the variable that stores the ADCC value and use the modulus operator or take the remainder from dividing by 10. The mathematical equation is 63 / 10 = 6.3 this means with every step we will get a value that counts from 0 to 9 repeating till it ends on 3. This will get rid of the garbage characters that were being displayed when turning the potentiometer passed the 9th digit.

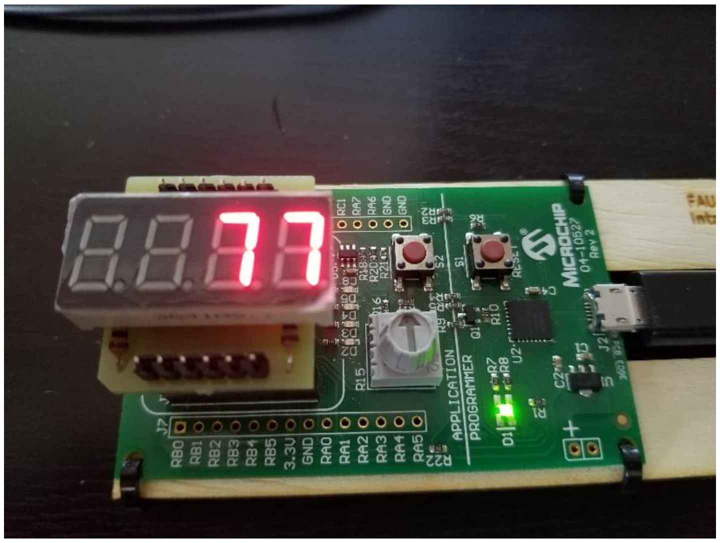
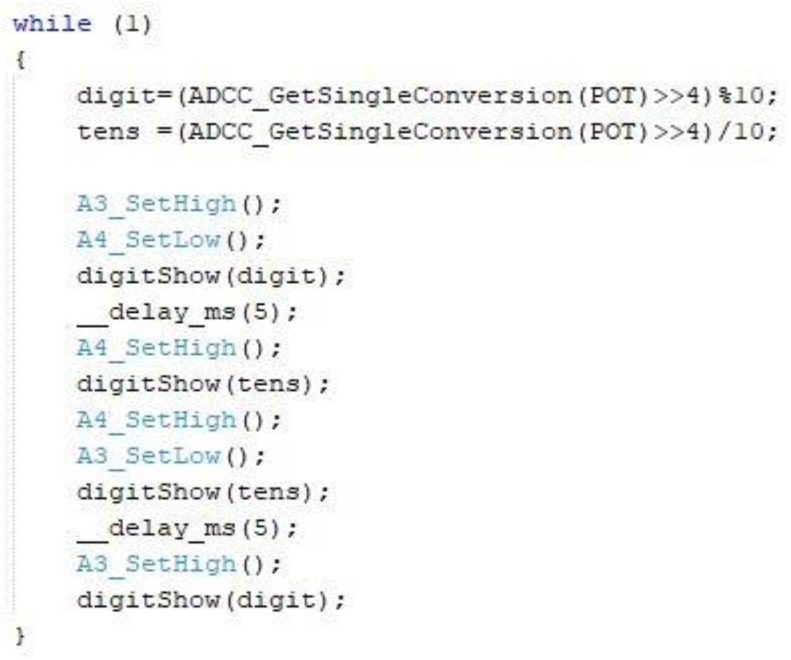


Figure 11: Modulus Division Result

*Notice the potentiometer in Figure 11 is almost set to the max value.

For the tens place we need to store the value of the ADCC in a new variable called "tens" and just simply divide it by 10. This will give us a value from 0 to 6. This will allow us to keep track of two digits with every rotation of the potentiometer. Your code should look like this.



```
while (1)
{
    digit=(ADCC_GetSingleConversion(POT)>>4)%10;
    tens =(ADCC_GetSingleConversion(POT)>>4)/10;

    A3_SetHigh();
    A4_SetLow();
    digitShow(digit);
    __delay_ms(5);
    A4_SetHigh();
    digitShow(tens);
    A4_SetHigh();
    A3_SetLow();
    digitShow(tens);
    __delay_ms(5);
    A3_SetHigh();
    digitShow(digit);
}
```

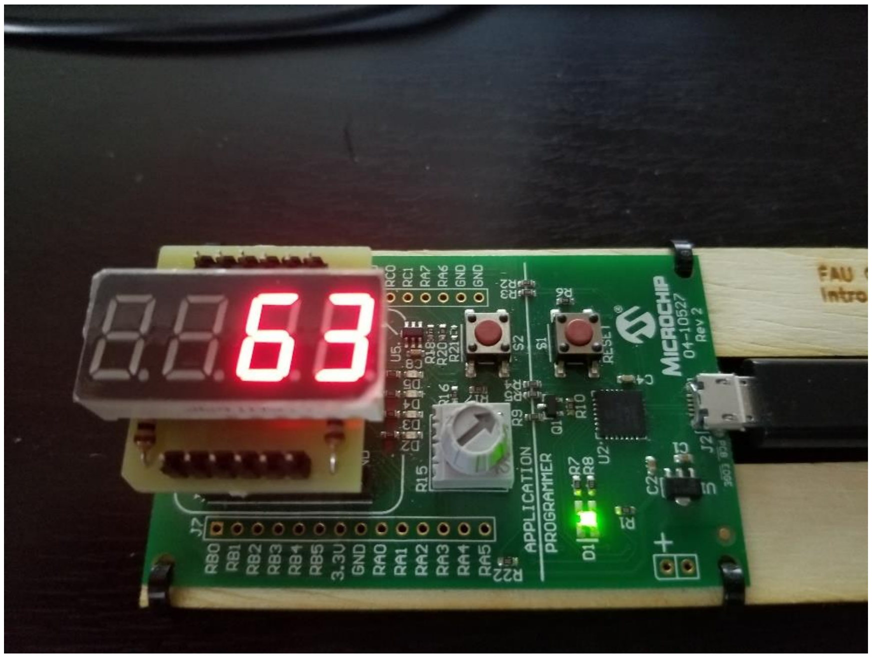
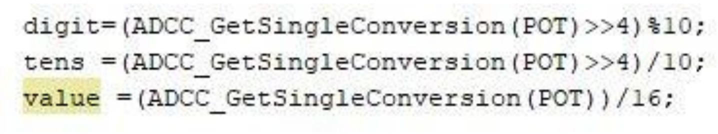


Figure 12: Tens Division Result

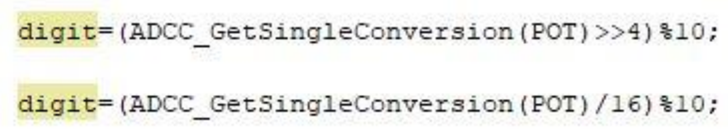
The last thing that needs to be done is to store the whole value of the ADCC into a variable this way the numbers that are entered match up with the numbers that are being displayed. I stored this in a variable called "value".



```
digit=(ADCC_GetSingleConversion(POT)>>4)%10;
tens =(ADCC_GetSingleConversion(POT)>>4)/10;
value =(ADCC_GetSingleConversion(POT))/16;
```

Figure 13: Value Variable

Note that in my code I used two different methods to get the value needed for the display and the value needed to compare with the combination being set. one uses bitwise logic operators and division while the other just uses division by 16 that's because 1024/ 16 = 64. You can choose whatever you would like. I only did this for teaching purposes.



```
digit=(ADCC_GetSingleConversion(POT)>>4)%10;

digit=(ADCC_GetSingleConversion(POT)/16)%10;
```

Figure 14: More Than One Way...

The rest of the code uses a combination of while loops, function calls and if statement to build the final solution.

Results

Overall the project came out better then expected. The use of the Seven-Segment display in combination with the onboard peripherals make this project possible. I programmed the microcontroller to display the numbers 0 through 63. I used the tactile switch to set the current combination and to lock the system. I used the LEDs to display when a combination has been set. The display scrolls the text just like I wanted, and the blinking LEDs definitely helps with indicating certain operations that the board is doing. Working Examples can be seen in Figures 15,16 and 17.

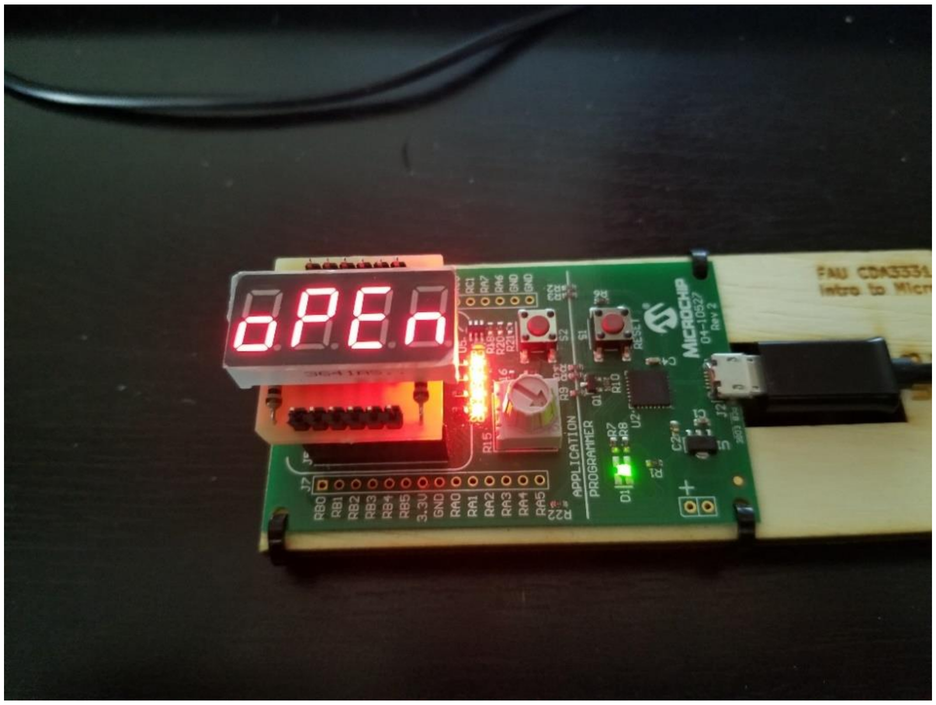


Figure 15: Display Open

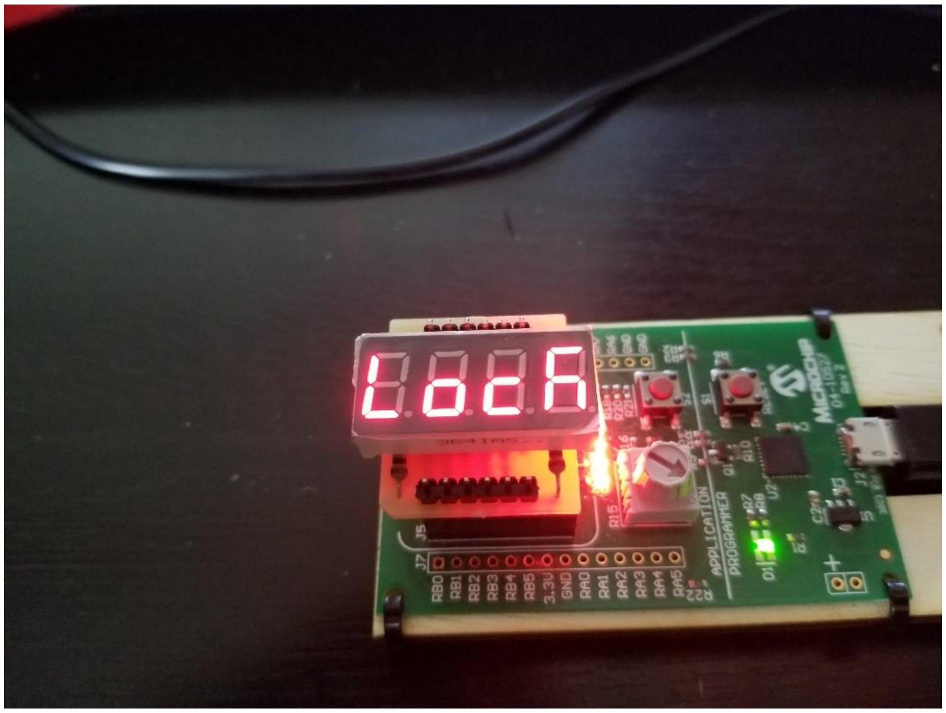


Figure 16: Display Lock

Working Example Link:

https://youtu.be/s3pvt_3-JYw

Discussion

When creating this project, I believe the hardest part was trying to figure out how to display the appropriate numbers. The challenge of getting them to change seamlessly with the potentiometer in a way to make user experience feel more like a padlock was the hardest part of this project. After figuring out how to display the number correctly the rest of the project went smoothly. I didn't run into any problems other than my inability to make a decision. Whether it had to do with what messages it should display or if there should be blinking lights in the beginning and end of entering a new combination. One thing that I had investigated while developing this project was if there was room in the GPIO ports to connect a servo motor. This would have been the physical locking mechanism for this project. unfortunately, I could not accomplish this because I would have to solder new header pins and did not want to risk damaging the board.

Conclusion

In conclusion, this project has taught me a lot. when I started, I did not know the C programming language but with my understanding of how all programming languages work and my knowledge of C++ the transition was not hard. I know that the code could be optimized by someone with a better understanding of C. This is not to discredit my own work but to acknowledge that there is room for improvement. I hope in the near future I will be recreating this project to work with a real locking mechanism to further develop my understanding of this subject.

References

[1] L. Di Jasio, *In 10 Lines of Code,* Lulu Enterprises, Inc., 2016, p.55-63

Appendix Page

Listing 1: Main.c

```
/*
created by Daniel Leach
4/16/19
CDA3331C
*/
#include "mcc_generated_files/mcc.h"
// global variable arrays
uint8_t matrix[]={
  //a  b  c  d  e  f  g
    0, 0, 0, 0, 0, 0, 1, //0
    1, 0, 0, 1, 1, 1, 1,
    0, 0, 1, 0, 0, 1, 0, //2
    0, 0, 0, 0, 1, 1, 0,
    1, 0, 0, 1, 1, 0, 0, //4
    0, 1, 0, 0, 1, 0, 0,
    0, 1, 0, 0, 0, 0, 0, //6
    0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 0, 0, 0, //8
    0, 0, 0, 0, 1, 0, 0,
};
uint8_t Lock[]={
   0, 0, 0, 1, 1, 1, 0,  // L
   0, 0, 1, 1, 1, 0, 1,  //o
   0, 0, 0, 1, 1, 0, 1,  //c
   1, 0, 1, 0, 1, 1, 1,  //K
};
uint8_t Open[]={
   0, 0, 1, 1, 1, 0, 1, //o
   1, 1, 0, 0, 1, 1, 1, //p
   1, 0, 0, 1, 1, 1, 1, //e
   0, 0, 1, 0, 1, 0, 1, //n
};
uint8_t Deny[]={
   0, 1, 1, 1, 1, 0, 1,  //d
   1, 0, 0, 1, 1, 1, 1, //e
   0, 0, 1, 0, 1, 0, 1, //n
   0, 1, 1, 1, 0, 1, 1, //y
};
uint8_t New[]={
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 1, 0, 1, 0, 1, //n
   1, 0, 0, 1, 1, 1, 1, //e
   0, 1, 1, 1, 1, 1, 1, //w
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 1, 1, 0, 1, //c
   0, 0, 1, 1, 1, 0, 1, //o
   1, 1, 0, 1, 0, 1, 0, //m
   0, 0, 1, 1, 1, 1, 1, //b
   0, 0, 1, 1, 1, 0, 1, //o
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
};
uint8_t Set[] ={
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 1, 1, 0, 1, //c
   0, 0, 1, 1, 1, 0, 1, //o
   1, 1, 0, 1, 0, 1, 0, //m
   0, 0, 1, 1, 1, 1, 1, //b
   0, 0, 1, 1, 1, 0, 1, //o
   0, 0, 0, 0, 0, 0, 0,
   1, 0, 1, 1, 0, 1, 1, //s
   1, 0, 0, 1, 1, 1, 1, //e
   0, 0, 0, 1, 1, 1, 1, //t
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
};
// function prototypes
void OpenShow(uint8_t value);
void LockShow(uint8_t value);
void DenyShow(uint8_t value);
void NewShow(uint8_t value);
void digitShow(uint8_t value);
void SetShow(uint8_t value);
void DisplayOpen();
void DisplayLock();
void DisplayDeny();
void RunningADCC(uint8_t digit,uint8_t tens, int *combo1,int *combo2,int
*combo3,int *combo4,int count,int value,bool running);
void RunningNew(uint8_t digit,uint8_t tens,int *num1,int *num2,int *num3,int
*num4,int *value,bool working);
void Scrollnew();
void ScrollSet();
void Reset();
void main(void){
    // initialize the device
    SYSTEM_Initialize();
    int combo1;
    int combo2;
    int combo3;
    int combo4;
    int num1 = 10;
    int num2 = 20;
    int num3 = 30;
    int num4 = 40;
    // counter for switch statement in main
    int count=0;
    // used to store ADCC Values
    uint8_t digit;
    uint8_t tens;
    int value;
    // used for running while loop
      bool running = true;
    bool working = true;
      // turns off all displays before starting while loop.
    A1_SetHigh();
    A2_SetHigh();
    A3_SetHigh();
    A4_SetLow();
    SEG_DP_SetLow();
 RunningADCC(digit,tens,&combo1,&combo2,&combo3,&combo4,count,value,running);
    // main while loop
    while(1){
            digit=(ADCC_GetSingleConversion(POT)>>4)%10;
            tens =(ADCC_GetSingleConversion(POT)>>4)/10;
            value =(ADCC_GetSingleConversion(POT))/16;
            ////// breaks loop and checks if combo is right ////////////////
            if(combo1 == num1 && combo2 == num2 && combo3 == num3 && combo4
== num4){
            DisplayOpen();
            if(value == 63){
                if(BTN_GetValue() == 0){
                    __delay_ms(200);
                    Scrollnew();

RunningNew(digit,tens,&num1,&num2,&num3,&num4,value,working);
                    count = 0;
                    Reset();

RunningADCC(digit,tens,&combo1,&combo2,&combo3,&combo4,count,value,running);
                }
            }else{
                if(BTN_GetValue() == 0){
                    DisplayLock();
                    count = 0;
                    Reset();

RunningADCC(digit,tens,&combo1,&combo2,&combo3,&combo4,count,value,running);
                    }
            }
            }else{
            DisplayDeny();
                  if(digit == 0 && tens == 0 && value ==0){
                        count = 0;
                        Reset();

RunningADCC(digit,tens,&combo1,&combo2,&combo3,&combo4,count,value,running);
                  }
            }
    }
}
// reset function
void Reset(){
    A1_SetHigh();
    A2_SetHigh();
    A3_SetHigh();
    A4_SetHigh();
    LED0_SetLow();
    LED1_SetLow();
    LED2_SetLow();
    LED3_SetLow();
}
```

```
// function for assigning array values to
// segment LEDs
void OpenShow(uint8_t value){
    uint8_t *p=&Open[value*7];
    SEG_A_LAT = *p++;
    SEG_B_LAT = *p++;
    SEG_C_LAT = *p++;
    SEG_D_LAT = *p++;
    SEG_E_LAT = *p++;
    SEG_F_LAT = *p++;
    SEG_G_LAT = *p++;
}
void LockShow(uint8_t value){
    uint8_t *p=&Lock[value*7];
    SEG_A_LAT = *p++;
    SEG_B_LAT = *p++;
    SEG_C_LAT = *p++;
    SEG_D_LAT = *p++;
    SEG_E_LAT = *p++;
    SEG_F_LAT = *p++;
    SEG_G_LAT = *p++;
}
void DenyShow(uint8_t value){
    uint8_t *p=&Deny[value*7];
    SEG_A_LAT = *p++;
    SEG_B_LAT = *p++;
    SEG_C_LAT = *p++;
    SEG_D_LAT = *p++;
    SEG_E_LAT = *p++;
    SEG_F_LAT = *p++;
    SEG_G_LAT = *p++;
}
void NewShow(uint8_t value){
    uint8_t *p=&New[value*7];
    SEG_A_LAT = *p++;
    SEG_B_LAT = *p++;
    SEG_C_LAT = *p++;
    SEG_D_LAT = *p++;
    SEG_E_LAT = *p++;
    SEG_F_LAT = *p++;
    SEG_G_LAT = *p++;
}
void digitShow(uint8_t value){
    uint8_t *p=&matrix[value*7];
    SEG_A_LAT = ~*p++;
    SEG_B_LAT = ~*p++;
    SEG_C_LAT = ~*p++;
    SEG_D_LAT = ~*p++;
    SEG_E_LAT = ~*p++;
    SEG_F_LAT = ~*p++;
    SEG_G_LAT = ~*p++;
}
void SetShow(uint8_t value){
    uint8_t *p=&Set[value*7];
    SEG_A_LAT = *p++;
    SEG_B_LAT = *p++;
    SEG_C_LAT = *p++;
    SEG_D_LAT = *p++;
    SEG_E_LAT = *p++;
    SEG_F_LAT = *p++;
    SEG_G_LAT = *p++;
}
// function for getting combo from user
void RunningADCC(uint8_t digit,uint8_t tens,int *combo1,int *combo2,int
*combo3,int *combo4,int count,int value,bool running){
      while(running){
            digit=(ADCC_GetSingleConversion(POT)>>4)%10;
            tens =(ADCC_GetSingleConversion(POT)>>4)/10;
            value =(ADCC_GetSingleConversion(POT))/16;
            A3_SetHigh();
            A4_SetLow();
            digitShow(digit);
            __delay_ms(5);
            A4_SetHigh();
            digitShow(tens);
            A4_SetHigh();
            A3_SetLow();
            digitShow(tens);
            __delay_ms(5);
            A3_SetHigh();
            digitShow(digit);
            if (BTN_GetValue() == 0){
                  __delay_ms(200);
                  switch(count){
                        case 0:
                              *combo1 =  value;
                    LED0_SetHigh();
                              break;
                        case 1:
                            *combo2 =  value;
                    LED1_SetHigh();
                              break;
                        case 2:
                              *combo3 = value;
                    LED2_SetHigh();
                              break;
                        case 3:
                              *combo4 = value;
                    LED3_SetHigh();
                              running = false;
                    break;
                        default:
                    break;
                  };
                  count++;
            }
      }
}
// function for setting a new combo
void RunningNew(uint8_t digit,uint8_t tens,int *num1,int *num2,int *num3,int
*num4,int *value,bool working){
    int count = 0;
    Reset();
    while(working){
        digit=(ADCC_GetSingleConversion(POT)>>4)%10;
        tens =(ADCC_GetSingleConversion(POT)>>4)/10;
        value =(ADCC_GetSingleConversion(POT))/16;
        A3_SetHigh();
        A4_SetLow();
        digitShow(digit);
        __delay_ms(5);
        A4_SetHigh();
        digitShow(tens);
        A4_SetHigh();
        A3_SetLow();
        digitShow(tens);
        __delay_ms(5);
        A3_SetHigh();
        digitShow(digit);
        if(BTN_GetValue()==0){
            __delay_ms(200);
            switch(count){
                case 0:
                    *num1 =  value;
                    LED0_SetHigh();
                    break;
                case 1:
                    *num2 =  value;
                    LED1_SetHigh();
                    break;
                case 2:
                    *num3 = value;
                    LED2_SetHigh();
                    break;
                case 3:
                    *num4 = value;
                    LED3_SetHigh();
                    ScrollSet();
                    DisplayLock();
                    working= false;
                    break;
                default:
                    break;
            };
        count++;
        }
    }
}
// function for displaying messages
void DisplayOpen(){
        A1_SetHigh();
        A4_SetLow();
        OpenShow(3);
        __delay_ms(5);
        A4_SetHigh();
        OpenShow(2);
        A4_SetHigh();
        A3_SetLow();
        OpenShow(2);
        __delay_ms(5);
        A3_SetHigh();
        OpenShow(1);
        A3_SetHigh();
        A2_SetLow();
        OpenShow(1);
        __delay_ms(5);
        A2_SetHigh();
        OpenShow(0);
        A2_SetHigh();
        A1_SetLow();
        OpenShow(0);
        __delay_ms(5);
        A1_SetHigh();
        OpenShow(3);
}
void DisplayLock(){
    int count = 0;
    while(count < 100){
        A1_SetHigh();
        A4_SetLow();
        LockShow(3);
        __delay_ms(5);
        A4_SetHigh();
        LockShow(2);
        A4_SetHigh();
        A3_SetLow();
        LockShow(2);
        __delay_ms(5);
        A3_SetHigh();
        LockShow(1);
        A3_SetHigh();
        A2_SetLow();
        LockShow(1);
        __delay_ms(5);
        A2_SetHigh();
        LockShow(0);
        A2_SetHigh();
        A1_SetLow();
        LockShow(0);
        __delay_ms(5);
        A1_SetHigh();
        LockShow(3);
        count++;
    }
}
void DisplayDeny(){
    int count = 0;
    while(count < 100){
        A1_SetHigh();
        A4_SetLow();
        DenyShow(3);
        __delay_ms(5);
        A4_SetHigh();
        DenyShow(2);
        A4_SetHigh();
        A3_SetLow();
        DenyShow(2);
        __delay_ms(5);
        A3_SetHigh();
        DenyShow(1);
        A3_SetHigh();
        A2_SetLow();
        DenyShow(1);
        __delay_ms(5);
        A2_SetHigh();
        DenyShow(0);
        A2_SetHigh();
        A1_SetLow();
        DenyShow(0);
        __delay_ms(5);
        A1_SetHigh();
        DenyShow(3);
        count++;
    }
}
//function for scrolling messages
void Scrollnew(){
    int scrl_len = 0, scrl_speed = 0;
        int count = 0;
    while(count < 3){
        LED0_SetHigh();
        LED1_SetHigh();
        LED2_SetHigh();
        LED3_SetHigh();
        __delay_ms(200);
        LED0_SetLow();
        LED1_SetLow();
        LED2_SetLow();
        LED3_SetLow();
        __delay_ms(200);
        count++;
    }
    while (scrl_len<12){
        while (scrl_speed < 11){
            A4_SetLow();
            __delay_ms(3);
            A4_SetHigh();
            NewShow(scrl_len+2);
            __delay_ms(1);
            A3_SetLow();
            __delay_ms(3);
            A3_SetHigh();
            NewShow(scrl_len+1);
            __delay_ms(1);
            A2_SetLow();
            __delay_ms(3);
            A2_SetHigh();
            NewShow(scrl_len);
            __delay_ms(1);
            A1_SetLow();
            __delay_ms(3);
            A1_SetHigh();
            NewShow(scrl_len+3);
            __delay_ms(1);
            scrl_speed++;
        }
        scrl_speed=0;
        scrl_len++;
    }
A1_SetHigh();
A2_SetHigh();
A3_SetHigh();
A4_SetHigh();
}
void ScrollSet(){
    int scrl_len = 0, scrl_speed = 0;
    while (scrl_len<12){
        while (scrl_speed < 11){
            A4_SetLow();
            __delay_ms(3);
            A4_SetHigh();
            SetShow(scrl_len+2);
            __delay_ms(1);
            A3_SetLow();
            __delay_ms(3);
            A3_SetHigh();
            SetShow(scrl_len+1);
            __delay_ms(1);
            A2_SetLow();
            __delay_ms(3);
            A2_SetHigh();
            SetShow(scrl_len);
            __delay_ms(1);
            A1_SetLow();
            __delay_ms(3);
            A1_SetHigh();
            SetShow(scrl_len+3);
            __delay_ms(1);
            scrl_speed++;
        }
        scrl_speed=0;
        scrl_len++;
    }
    int count = 0;
while(count < 3){
    LED0_SetHigh();
    LED1_SetHigh();
    LED2_SetHigh();
    LED3_SetHigh();
    __delay_ms(200);
    LED0_SetLow();
    LED1_SetLow();
    LED2_SetLow();
    LED3_SetLow();
    __delay_ms(200);
    count++;
}
A1_SetHigh();
A2_SetHigh();
A3_SetHigh();
A4_SetHigh();
}
```