Digital Lock System

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

Dalton McClain

Supervised by
Dr. Shankar
Douglas Athenosy
Caner Mutlu

Sunday, April 26, 2019

Table of Contents

Abstract

The objective of this design project was to use the course given evaluation board with a few of its peripherals and a common cathode 7 segment display. Using these components we needed to create a 4 digit digital locking system.

Project Specification Changes

The original specifications for this project wanted us to have the users choice for a digit, based on the potentiometer position, be selected by leaving it alone for a certain amount of time. The amount of time was decided by us and we didn't care for this style of selection. We decided to select the digits by having the user scroll through the possible values with the potentiometer and once they found the digit they wanted to use all they would have to do is press the button. Once the button was pressed the digit would then be "selected" and the user would then repeat the process for the remaining digits.

Another significant change that we made to the project would be the way the user decides to set a new combination for the lock. The original specifications wanted us to make it where the user would press the button a certain amount of times or hold it for one long press, again we did not like this. So to make the program more reliable we decided to allow the user to change the pin once the correct pin was entered. Once the correct pin was entered the user would need to turn the potentiometer all the way to zero (all the way counter-clockwise) and then press the button. This would initiate the "new pin" function of the program.

The last change we made was more of a minor change, but yet another change we wanted to implement for the reliability of the program. The original specifications wanted us to have the messages on the screen change when the potentiometer slightly moved or changed its current position, we wanted something a little more secure, in case of user error. So we decided to have the user press the button when coming to a message on screen, so we know the user is ready to continue without a doubt, because it is more difficult to "accidently" press the button than it is to "accidently" wiggle the potentiometer.

Design Methodology or Theory

In the designing phase of this project we had to figure out a way to balance the recommended specifications and the overall simplicity, reliability, and user friendliness. The specifications of the project that were provided gave us a heading to follow and a few goals to accomplish. The best way to utilize the hardware we have is to use an infinite loop, while controlling the outcome and speed at which it iterates. With this need for control we had to utilize conditional statements, accumulators, and delays, many delays. The conditional statements allowed for flow control and testing of the users inputs, the accumulators allowed us to check and control where the user was in the program and keep track of their position, and the delays allowed us to manage the output to the display, keeping glitches and hiccups out of the way. The way we wanted to design the processing of user information was to first get what the user wanted the initial pin to be, once we had that we were able to move on to testing that pin with what the user guessed. This required two sets of variables, one for getting and storing the initial pin and another for getting the guessed pin. After we had both sets of variables filled with

the users data, we needed to test the two sets against one another to see whether they matched or not and based on their parity we would implement the desired outcome.

Implementation

Implementation of this board starts with fixing the 7 segment display to the board and positioning the board in the correct manner so as to see the display as it is intended, you can see the 7 segment display connected to the board in figure 1 below.

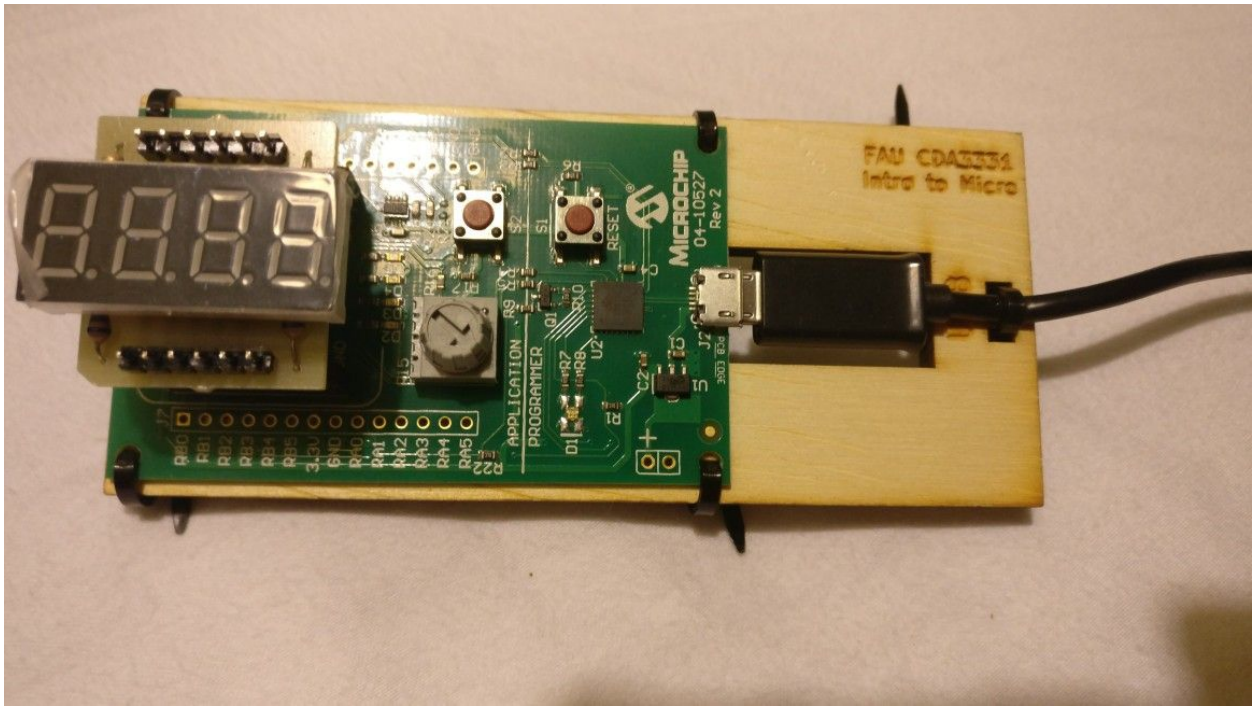

Figure 1: Starting Board

After getting the 7 segment display connected to the board properly we need to connect the board to the computer and start up our X IDE. Once the X IDE is up and the board is connected we can create a new project by using the the shortcut *CTRL+SHIFT+N*. After that we need to open MCC to generate the starting files necessary for this project. So with MCC open we are going to need the ADCC peripheral, the [PIC10 / PIC12 / PIC16] one will work best for us as we are using a PIC16 evaluation board see figure 2.

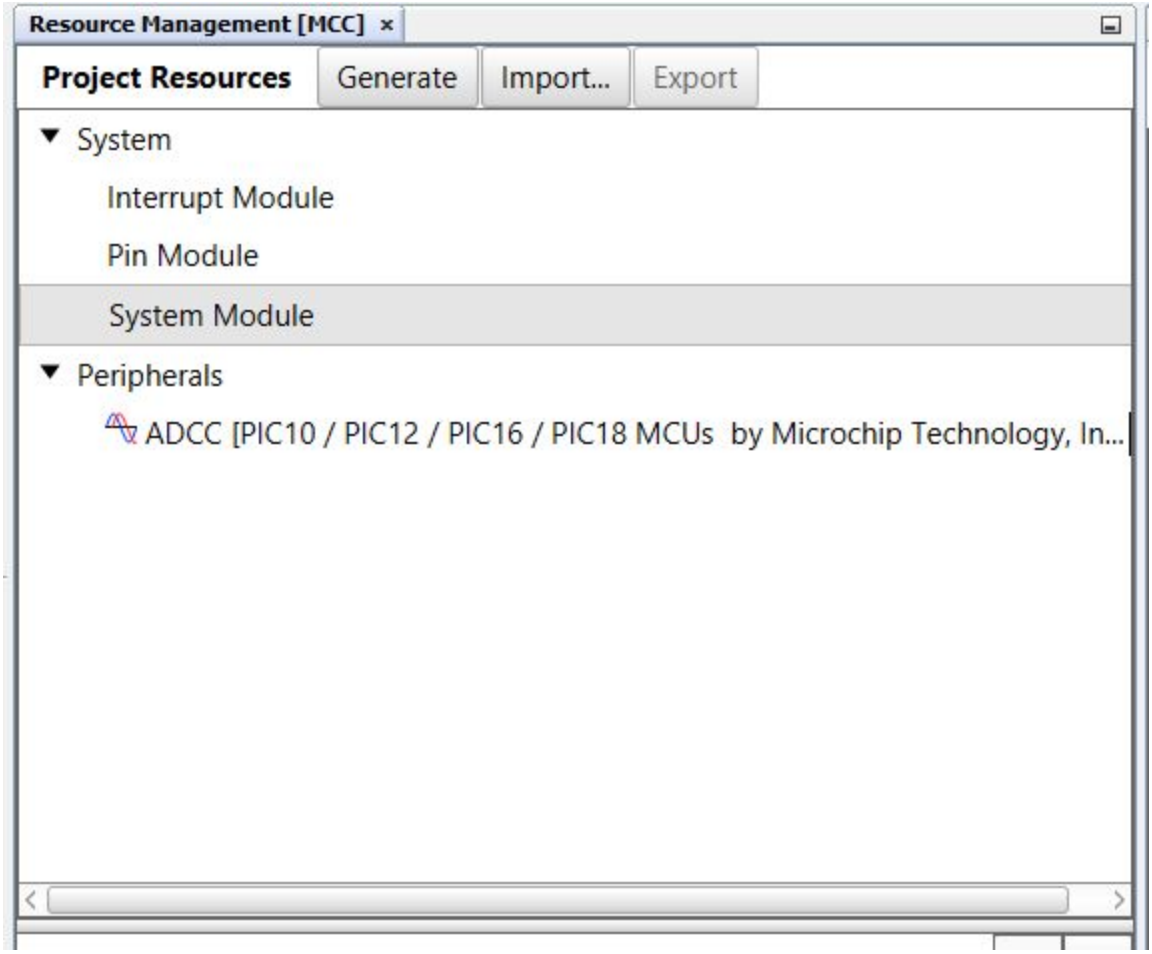

Figure 2: ADCC Peripheral

Now that we have added the ADCC peripheral to our program we need to make sure that it is set up according to the necessary configuration. Figure 3 shows exactly what should be where, we see that we need to be operating the ADCC in basic_mode with the clock source set to FOSC/ADCLK and the clock set to FOSC/2. There should be no need to change anything other than what is in figure 3.

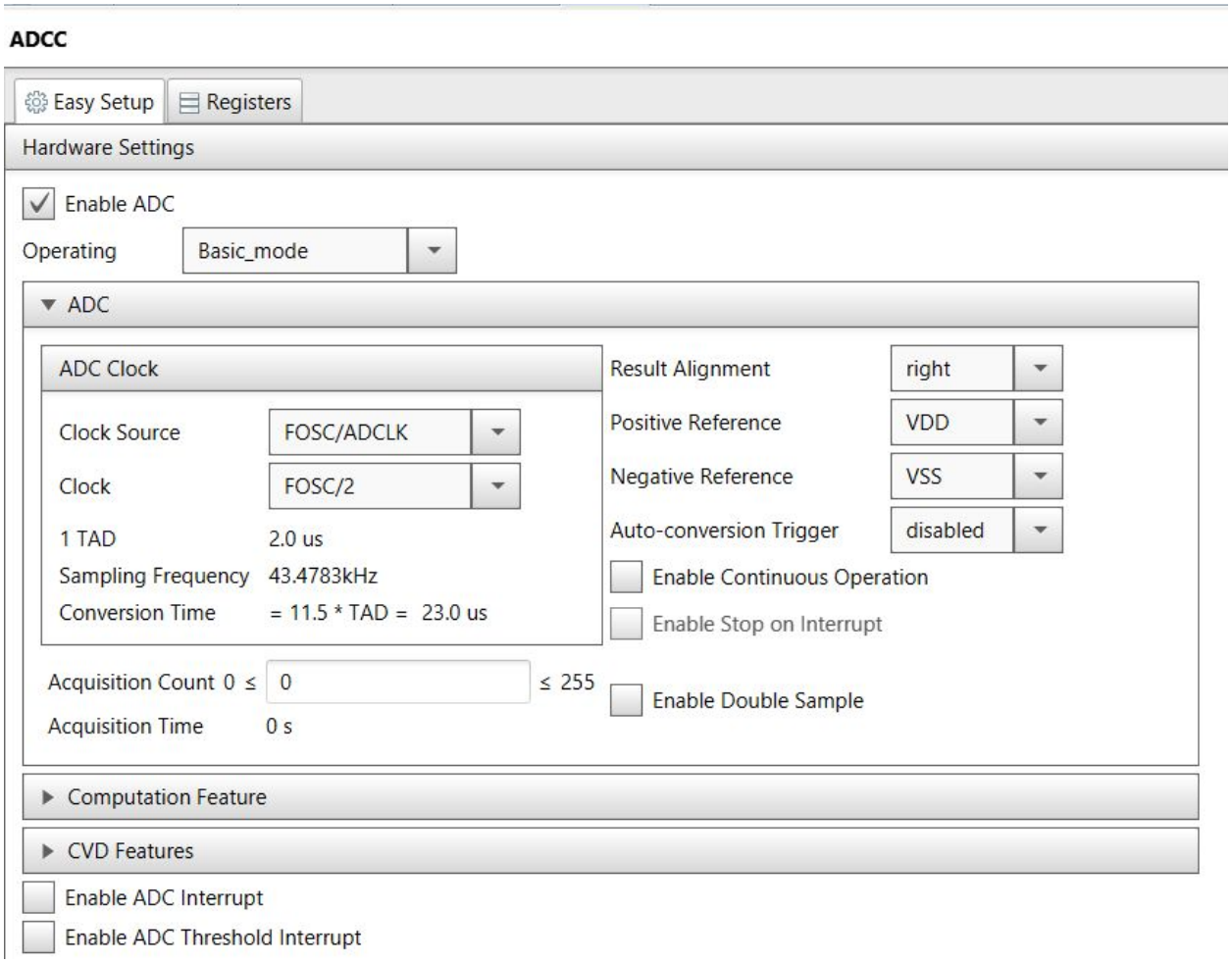

Figure 3: ADCC Configuration

Once we have configured the ADCC to our necessary specifications we then need to set up the hardware pins that we will be using. We will be using the *Pin Manager: Grid View* to set up our pins see figure 4 to follow along with which pins need to be set up. Right off the bat we need to set up our onboard LED pins, we will do this by locking GPIO output pins A0, A1, A2, and A3. Next we are going to want to grab our potentiometer by locking Anx input pin A4, then we are going to get the button and to do that we will lock GPIO input pin A5. After getting the potentiometer and button squared away we need to set up the 7 segment display pins by locking GPIO output pins B0, B1, B2, B3, B4, B5 and C2, C3, C4, C5, C6, C7.

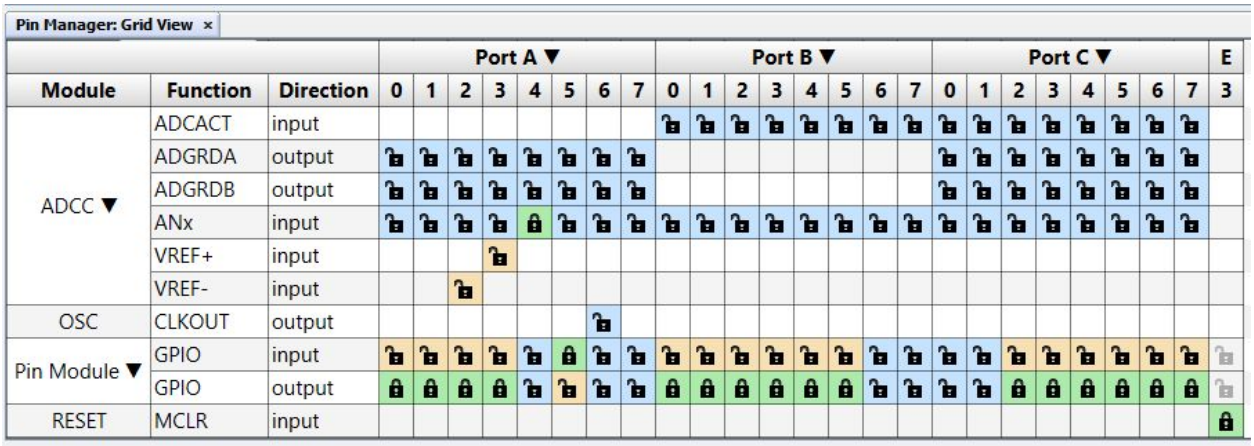

Figure 4: Pin Setup

Moving forward we need to give our pins, that we just setup, custom names for easier use on the software side of our project. We will go through each name for each pin one after the other, in order see figure 5. RA0 = LED1, RA1 = LED2, RA2 = LED3, RA3 = LED4, RA4 = POT, RA5 = BTN, RB0 = SEG_E, RB1 = SEG_D, RB2 = SEG_DP, RB3 = SEG_C, RB4 = SEG_G, RB5 = RC4, RC2 = SEG_A, RC3 = SEG_B, RC4 = C3, RC5 = C2, RC6 = SEG_F, and RC7 = C1.

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA0	Pin Module	GPIO	LED1	☐	☑	☑	☐	☐	none
RA1	Pin Module	GPIO	LED2	☐	☑	☑	☐	☐	none
RA2	Pin Module	GPIO	LED3	☐	☑	☑	☐	☐	none
RA3	Pin Module	GPIO	LED4	☐	☑	☑	☐	☐	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	C4	☐	☑	☑	☐	☐	none
RC2	Pin Module	GPIO	SEG_A	☐	☑	☑	☐	☐	none
RC3	Pin Module	GPIO	SEG_B	☐	☑	☑	☐	☐	none
RC4	Pin Module	GPIO	C3	☐	☑	☑	☐	☐	none
RC5	Pin Module	GPIO	C2	☐	☑	☑	☐	☐	none
RC6	Pin Module	GPIO	SEG_F	☐	☑	☑	☐	☐	none
RC7	Pin Module	GPIO	C1	☐	☑	☑	☐	☐	none

Figure 5: Custom Naming Pins

Now that the setup of all of the hardware is complete we will move on to the coding aspect of the project. The full .c file is below see Listing 2. The set up for this project requires an array that stores all of the letters for the words that we will need to show on the display. The matrix_word array holds those letters. The digitShow function updates and flashes the 7

segment display with the passed value, in our case the value is basically a coordinate for the letter we want to show to the display. We create a few arrays to hold the letters for the words needed to display “set”, “entr”, “open”, “deny”, “lock__”, and “new”.

The first as prgm_count == 1 and btn_count == 0 the first if statement goes through and we get our first digit entered by the user, then the second, third, and fourth and a message on the display says “set”(see figure 6) indicating that you will set the pin. After the fourth button press the prgm_count increases to 1 and the btn_count resets to 0. With that the second set of conditionals goes through getting the users guess digits for the pin and a message on the display says “entr”(see figure 7) indicating that you will enter the digits you selected, once the btn_count passes 3 the prgm_count increases to 2 and the main conditional statement goes through, the one that handles whether the pin entered was correct or incorrect. If the user entered the incorrect pin the lock displays “deny”(see figure 8) to the screen because it isn’t the right pin, if it is the correct pin the lock displays “open”(see figure 9). If the user wants to change the pin combination they are able to do so by setting the potentiometer to 0 (all the way counter-clockwise,) in doing this a message on the display says “new”(see figure 10) indicating that the program has restarted and is going to take a new pin. If you open the lock and do not want to change the pin, then keep the potentiometer off of 0 when pressing the button when the display says “open”, when you do this the display will say “lock”(see figure 11) indicating the the lock is still locked and the correct pin needs to be entered to reopen it.

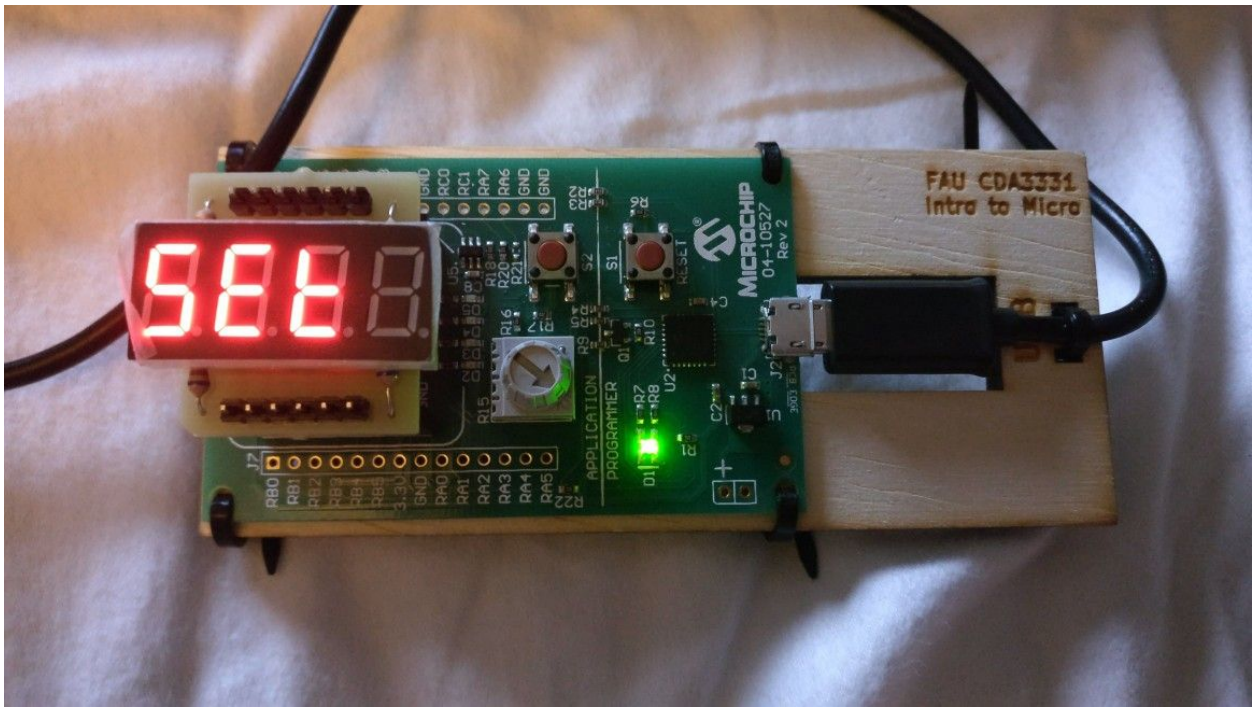

Figure 6: Set

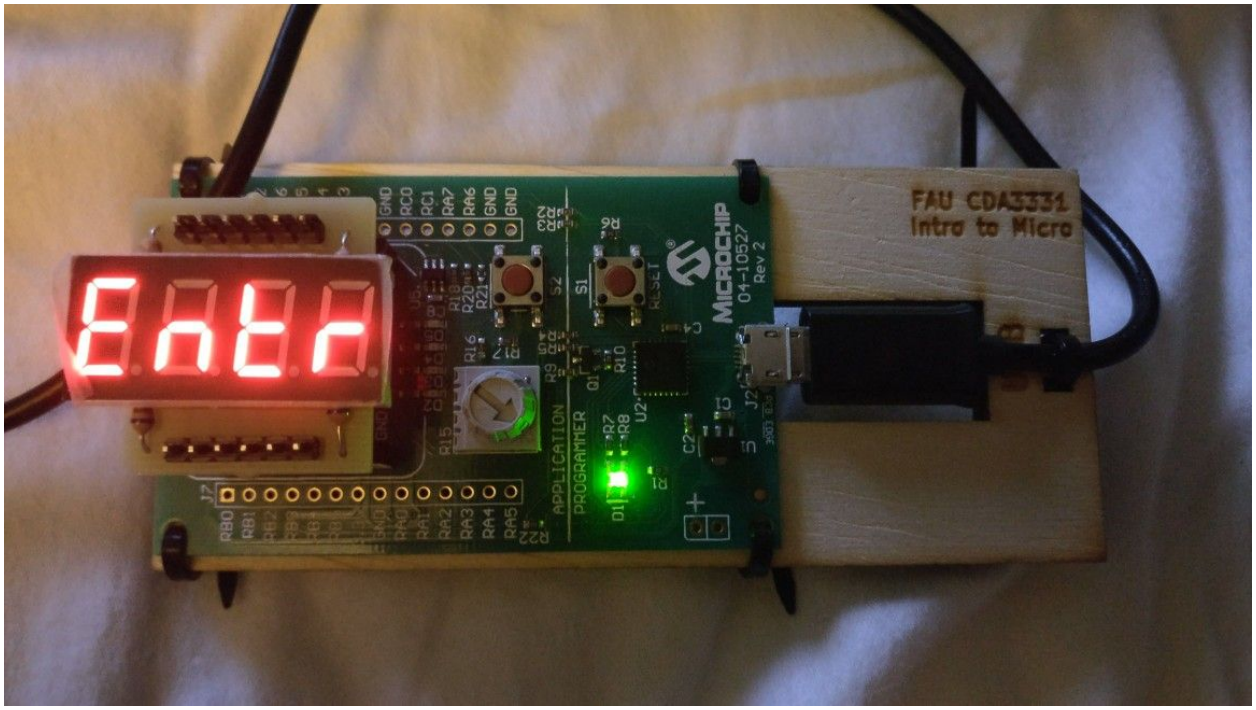

Figure 7: Entr

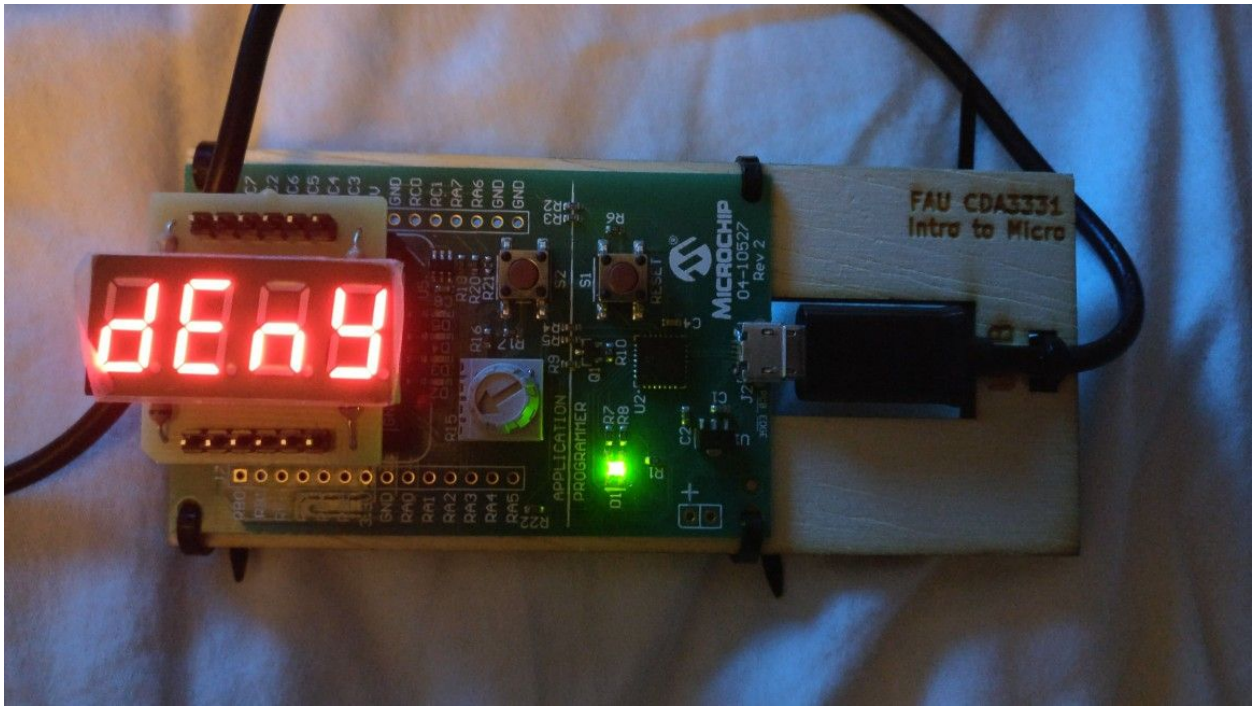

Figure 8: Deny

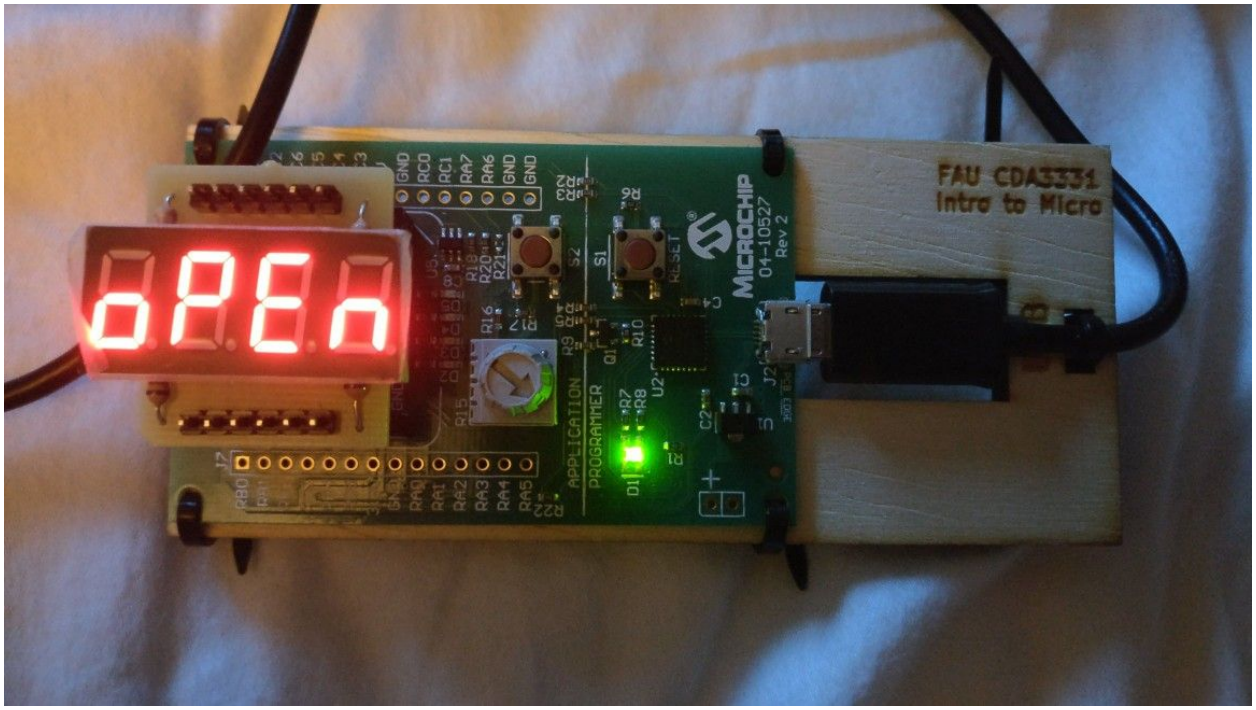

Figure 9: Open

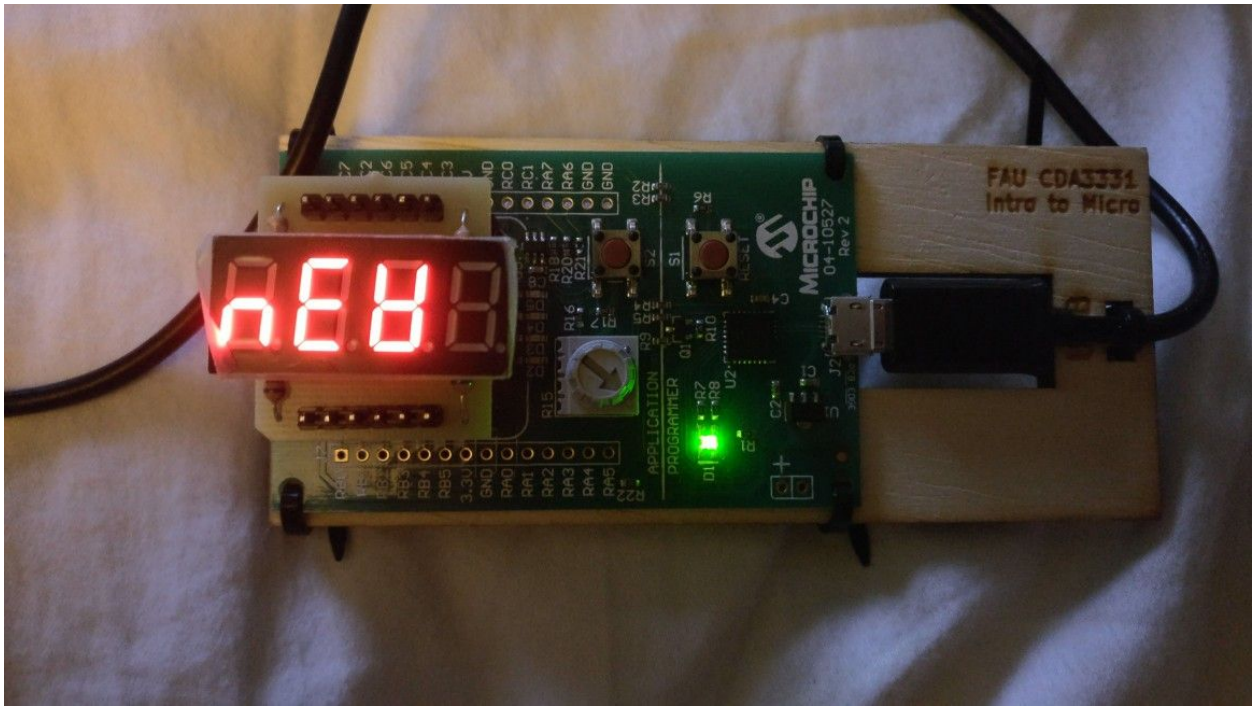

Figure 10: New

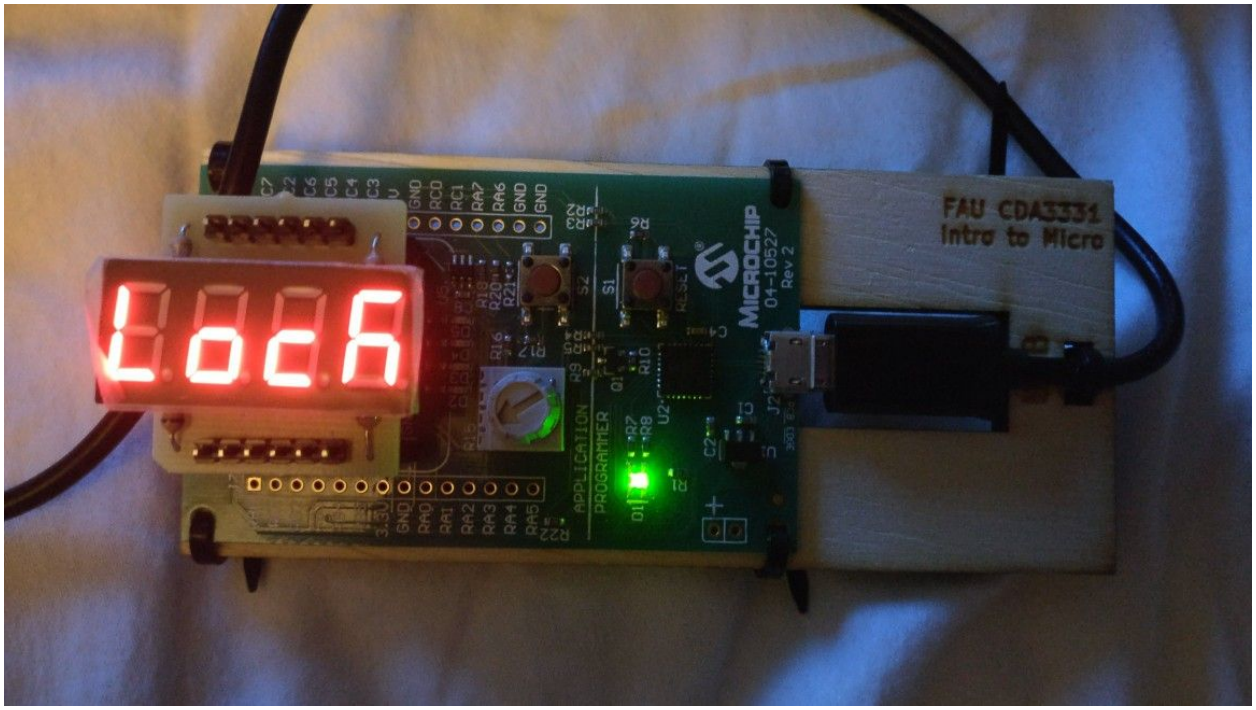

Figure 11: Lock

Results

I am very happy with how to project turned out, it surpassed my expectations. I enjoyed debugging and working out the kinks of this program as I actually saw differences in the way it performed as I cleaned it up. Listing 1 is a link to a youtube video that I put together showing off my project. In this video I go over everything that I talked about in this report, so not only do you get to see what it looks like when it is all said and done, but you know how the entire project works behind the scenes as well!

Discussion

The project took a long time to debug, I had to use very exact delay times in very specific places to ensure that the flow of the display was interrupted and even then I still was not able to get it perfect. I spent hours fiddling with the delays and I will say that it looks far better now than it did when I first "finished" it. Other then the messing around with the delays the project was fun, figuring out how to control the users experience and the flow of the program based on the users input was a challenge, but it was a very good learning experience for me.

Conclusion

In conclusion, this project taught me more about how to control hardware through the use of software than countless other projects have. I heavily enjoyed working on this project, on every aspect from implementing the software that controls the hardware to setting up the board for the use of the project. Learning how to set the pins up for use and how to rename the pins came in handy for the readability of the program. I am not as skilled in C as I am in other languages but the skills that I have learned throughout the years from coding countless

programs and projects ranging from robots and apps, to cars and websites, transferred over quite nicely for my first real in depth experience into the language of C.

References

[1] L. Di Jasio, *In 10 Lines of Code,* Lulu Enterprises, Inc., 2016, p.65-72

Appendix Page

Listing 1: Video

[Design Project Report Video]
https://youtu.be/0xWywiTK9fM

Listing 2: Main.c

```
#include "mcc_generated_files/mcc.h"

#define PRESSED     0


//Matrix for word gathering

uint8_t matrix_word[] = {

    //a  b  c  d  e  f  g

   1, 1, 1, 1, 1, 1, 0, //0

   0, 1, 1, 0, 0, 0, 0,//1

   1, 1, 0, 1, 1, 0, 1, //2

   1, 1, 1, 1, 0, 0, 1, //3

   0, 1, 1, 0, 0, 1, 1, //4

   1, 0, 1, 1, 0, 1, 1, //5

   1, 0, 1, 1, 1, 1, 1, //6

   1, 1, 1, 0, 0, 0, 0, //7

   1, 1, 1, 1, 1, 1, 1, //8

   1, 1, 1, 1, 0, 1, 1, //9

   1, 1, 1, 0, 1, 1, 1, //a10

   0, 0, 1, 1, 1, 1, 1, //b11

   0, 0, 0, 1, 1, 0, 1, //c12

   0, 1, 1, 1, 1, 0, 1, //d13

   1, 0, 0, 1, 1, 1, 1, //e14
```

```
    1, 0, 0, 0, 1, 1, 1, //f15
    1, 0, 1, 1, 1, 1, 0, //g16
    0, 1, 1, 0, 1, 1, 1, //h17
    0, 0, 0, 0, 1, 1, 0, //i18
    0, 1, 1, 1, 1, 0, 0, //j19
    1, 0, 1, 0, 1, 1, 1, //K20
    0, 0, 0, 1, 1, 1, 0, //l21
    1, 1, 0, 1, 0, 1, 0, //m22
    0, 0, 1, 0, 1, 0, 1, //n23
    0, 0, 1, 1, 1, 0, 1, //o24
    1, 1, 0, 0, 1, 1, 1, //p25
    1, 1, 1, 0, 0, 1, 1, //q26
    0, 0, 0, 0, 1, 0, 1, //r27
    1, 0, 1, 1, 0, 1, 1, //s28
    0, 0, 0, 1, 1, 1, 1, //t29
    0, 0, 1, 1, 1, 0, 0, //u30
    0, 1, 0, 1, 0, 1, 0, //v31
    0, 1, 1, 1, 1, 1, 1, //w32
    1, 0, 0, 1, 0, 0, 1, //x33
    0, 1, 1, 1, 0, 1, 1, //y34
    1, 1, 0, 1, 1, 0, 1, //z35
    0, 0, 0, 0, 0, 0, 0  //SPACE36
};



//Matrices for words that need to be displayed
uint8_t set[] = {28, 14, 29, 36};
uint8_t entr[] = {14, 23, 29, 27};
```

```
uint8_t open[] = {24, 25, 14, 23};

uint8_t deny[] = {13, 14, 23, 34};

uint8_t lock__[] = {21, 24, 12, 20};

uint8_t new[] = {23, 14, 32, 36};


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.

   //NOTE this code has been changed for common cathode the original code was ~*p++;

   //the '~' is bitwise NOT in this instance it made all highs we had in the matrix low and the lows high this allowed for common anode to work

   uint8_t *p=&matrix_word[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 main(void)

{

   uint8_t digit__, last__, zero = 0;

   // initialize the device

   SYSTEM_Initialize();


   //Control accumulators
```

```
int btn_count = 0, prgm_count = 0;
int set_btn = 1, entr_btn = 1, open_btn = 1, deny_btn = 1, lock = 1, new_display = 0, lock_display = 0;
int digit[4];
int digit_tmp[4];

//Set digits high
C1_SetHigh();
C2_SetHigh();
C3_SetHigh();
C4_SetHigh();
SEG_DP_LAT = 0;

//Program loop
while (1)
{
   //loop to display "new"

   while(set_btn == 1 && new_display == 1){

    C1_SetHigh();
    digitShow(new[3]);
    __delay_ms(1);
    C4_SetLow();
    __delay_ms(5);

    C4_SetHigh();
    digitShow(new[2]);
    __delay_ms(1);
```

```
C3_SetLow();
__delay_ms(5);

C3_SetHigh();
digitShow(new[1]);
__delay_ms(1);
C2_SetLow();
__delay_ms(5);

C2_SetHigh();
digitShow(new[0]);
__delay_ms(1);
C1_SetLow();
__delay_ms(5);

if(BTN_GetValue() == PRESSED){
    new_display = 0;
    __delay_ms(350);
}
}

//Clearing the display
C1_SetHigh();
C2_SetHigh();
C3_SetHigh();
C4_SetHigh();

//These conditionals go through the first stage of the lock, getting the initial code and keeping it set.
```

```
if(BTN_GetValue() == PRESSED && btn_count == 0 && prgm_count == 0 && lock == 1){
   digit[0] = (ADCC_GetSingleConversion(POT)>>4) % 10;
   digit_tmp[0] = digit[0];
   btn_count++;
   LED1_SetHigh();
   __delay_ms(300);
}
if(BTN_GetValue() == PRESSED && btn_count == 1 && prgm_count == 0 && lock == 1){
   digit[1] = (ADCC_GetSingleConversion(POT)>>4) % 10;
   digit_tmp[1] = digit[1];
   btn_count++;
   LED2_SetHigh();
   __delay_ms(300);
}
if(BTN_GetValue() == PRESSED && btn_count == 2 && prgm_count == 0 && lock == 1){
   digit[2] = (ADCC_GetSingleConversion(POT)>>4) % 10;
   digit_tmp[2] = digit[2];
   btn_count++;
   LED3_SetHigh();
   __delay_ms(300);
}
if(BTN_GetValue() == PRESSED && btn_count == 3 && prgm_count == 0 && lock == 1){
   digit[3] = (ADCC_GetSingleConversion(POT)>>4) % 10;
   digit_tmp[3] = digit[3];
   btn_count++;
   LED4_SetHigh();
   __delay_ms(50);
}
```

```
//lopp to display "set"

while(set_btn == 1 && prgm_count == 1){

 C1_SetHigh();
 digitShow(set[3]);
 __delay_ms(1);
 C4_SetLow();
 __delay_ms(5);

 C4_SetHigh();
 digitShow(set[2]);
 __delay_ms(1);
 C3_SetLow();
 __delay_ms(5);

 C3_SetHigh();
 digitShow(set[1]);
 __delay_ms(1);
 C2_SetLow();
 __delay_ms(5);

 C2_SetHigh();
 digitShow(set[0]);
 __delay_ms(1);
 C1_SetLow();
 __delay_ms(5);
```

```
 if(BTN_GetValue() == PRESSED){
    set_btn = 0;
    __delay_ms(350);
 }
}

//Clearing the display
C1_SetHigh();
C2_SetHigh();
C3_SetHigh();
C4_SetHigh();


//This conditional goes through the second part of the lock, which is the part where the user input the code,
//whether it's wrong or right has no effect on the program as of yet
if(BTN_GetValue() == PRESSED && btn_count == 0 && prgm_count == 1){
   digit[0] = (ADCC_GetSingleConversion(POT)>>4) % 10;
   btn_count++;
   LED1_SetHigh();
   __delay_ms(300);
}
if(BTN_GetValue() == PRESSED && btn_count == 1 && prgm_count == 1){
   digit[1] = (ADCC_GetSingleConversion(POT)>>4) % 10;
   btn_count++;
   LED2_SetHigh();
   __delay_ms(300);
}
if(BTN_GetValue() == PRESSED && btn_count == 2 && prgm_count == 1){
```

```
    digit[2] = (ADCC_GetSingleConversion(POT)>>4) % 10;
    btn_count++;
    LED3_SetHigh();
    __delay_ms(300);
}
if(BTN_GetValue() == PRESSED && btn_count == 3 && prgm_count == 1){
    digit[3] = (ADCC_GetSingleConversion(POT)>>4) % 10;
    btn_count++;
    LED4_SetHigh();
    __delay_ms(50);
}

//loop to display "lock"

while(lock_display == 1){

 C1_SetHigh();
 digitShow(lock__[3]);
 __delay_ms(1);
 C4_SetLow();
 __delay_ms(5);

 C4_SetHigh();
 digitShow(lock__[2]);
 __delay_ms(1);
 C3_SetLow();
 __delay_ms(5);
```

```
C3_SetHigh();
digitShow(lock__[1]);
__delay_ms(1);
C2_SetLow();
__delay_ms(5);

C2_SetHigh();
digitShow(lock__[0]);
__delay_ms(1);
C1_SetLow();
__delay_ms(5);

if(BTN_GetValue() == PRESSED){
   lock_display = 0;
   __delay_ms(350);
}
}

//Clearing the display
C1_SetHigh();
C2_SetHigh();
C3_SetHigh();
C4_SetHigh();

//loop to display "entr"
while(entr_btn == 1 && prgm_count == 2){

C1_SetHigh();
```

```
digitShow(entr[3]);
__delay_ms(1);
C4_SetLow();
__delay_ms(5);

C4_SetHigh();
digitShow(entr[2]);
__delay_ms(1);
C3_SetLow();
__delay_ms(5);

C3_SetHigh();
digitShow(entr[1]);
__delay_ms(1);
C2_SetLow();
__delay_ms(5);

C2_SetHigh();
digitShow(entr[0]);
__delay_ms(1);
C1_SetLow();
__delay_ms(5);

if(BTN_GetValue() == PRESSED){
   entr_btn = 0;
}
}
```

```
//Clearing the display
C1_SetHigh();
C2_SetHigh();
C3_SetHigh();
C4_SetHigh();

//This is the conditional statement that controls the flow of the program as well as the input handling of the users data.
if(BTN_GetValue() == PRESSED && btn_count == 0 && prgm_count == 2){
  if(digit[0] == digit_tmp[0] && digit[1] == digit_tmp[1] && digit[2] == digit_tmp[2] && digit[3] == digit_tmp[3]){
    __delay_ms(200);
    //loop to display "open"
    while(open_btn == 1){

      C1_SetHigh();
      digitShow(open[3]);
      __delay_ms(1);
      C4_SetLow();
      __delay_ms(5);

      C4_SetHigh();
      digitShow(open[2]);
      __delay_ms(1);
      C3_SetLow();
      __delay_ms(5);

      C3_SetHigh();
      digitShow(open[1]);
```

```
            __delay_ms(1);
            C2_SetLow();
            __delay_ms(5);

            C2_SetHigh();
            digitShow(open[0]);
            __delay_ms(1);
            C1_SetLow();
            __delay_ms(5);

            if(BTN_GetValue() == PRESSED){
               open_btn = 0;
               __delay_ms(25);
            }
           }

           //Clearing the display
           C1_SetHigh();
           C2_SetHigh();
           C3_SetHigh();
           C4_SetHigh();

           //This conditional is how we control the "reset" function of the program. If the user wants to reset the pin, all they have to do is set the potentiometer
           //to the 0 position (all the way counter clock wise) and press the button.
           if(((ADCC_GetSingleConversion(POT)>>4) % 10) == zero){
              //Completly reset everything to "restart" the program loop, whilst sending "new" to the display
               btn_count = 0;
```

```
                prgm_count = 0;
                set_btn = 1;
                entr_btn = 1;
                open_btn = 1;
                deny_btn = 1;
                lock = 1;
                new_display = 1;
                __delay_ms(200);
            }else{//If they don't press the button with the potentiometer at 0, it continues and sends "lock" to the display
                //Reset the relative values so as to maintain the fact that the user wants the lock to remain locked
                btn_count = 0;
                prgm_count = 1;
                set_btn = 2;
                entr_btn = 1;
                open_btn = 1;
                deny_btn = 1;
                lock = 0;
                lock_display = 1;
                __delay_ms(200);
            }


        }
        else{
            __delay_ms(200);
            //loop to display "deny"
            while(deny_btn == 1){
```

```
C1_SetHigh();
digitShow(deny[3]);
__delay_ms(1);
C4_SetLow();
__delay_ms(5);

C4_SetHigh();
digitShow(deny[2]);
__delay_ms(1);
C3_SetLow();
__delay_ms(5);

C3_SetHigh();
digitShow(deny[1]);
__delay_ms(1);
C2_SetLow();
__delay_ms(5);

C2_SetHigh();
digitShow(deny[0]);
__delay_ms(1);
C1_SetLow();
__delay_ms(5);

if(BTN_GetValue() == PRESSED){
   deny_btn = 0;
   __delay_ms(25);
```

```
            }
            }

            //Clearing the display
            C1_SetHigh();
            C2_SetHigh();
            C3_SetHigh();
            C4_SetHigh();

            //Resetting the relative values so as to maintain the fact tht the user still needs to enter the correct pin
            btn_count = 0;
            prgm_count = 1;
            set_btn = 2;
            entr_btn = 1;
            open_btn = 1;
            deny_btn = 1;
            lock = 0;
            __delay_ms(200);
        }
    }
    //This is the first part of the lock system, initializing the lock with the user entered pin
    else if(BTN_GetValue() == PRESSED && btn_count == 4 && prgm_count == 0){
        digit[0] = 0;
        digit[1] = 0;
        digit[2] = 0;
        digit[3] = 0;
        btn_count = 0;//increments the button press counter
        prgm_count++;//increments the program counter
```

```
        //This resets the LEDs
        LED1_SetLow();
        LED2_SetLow();
        LED3_SetLow();
        LED4_SetLow();
        __delay_ms(1000);
    }
    //This is the second part of the lock system, where the user enters a pin, whether it is the right or wrong pin doesn't matter here
    else if(BTN_GetValue() == PRESSED && btn_count == 4 && prgm_count == 1){
        btn_count = 0;//Reset the button count to 0
        prgm_count++;//increments the program counter
        //This resets the LEDs
        LED1_SetLow();
        LED2_SetLow();
        LED3_SetLow();
        LED4_SetLow();
        __delay_ms(1000);
    }

    //This is the logic behind getting the the potentiometer to select the values needed.
    //It works by taking advantage of the division and modulus operators, taking the ADCC potentiometer data bitshifted and then taking the modulus of it gives a remainder
    //and dividing it gives a rounded up value.
    digit__ = (ADCC_GetSingleConversion(POT)>>4) % 10;
    last__ = (ADCC_GetSingleConversion(POT)>>4) / 10;

    //This is just to control the flow of which digits on the display get lit up and which don't.
    //I chose the two digits on the right to keep things close (the display and the potentiometer)
```

```
        C3_SetHigh();
        C4_SetLow();
        digitShow(digit__);
        __delay_ms(2);
        C4_SetHigh();
        digitShow(last__);
        C4_SetHigh();
        C3_SetLow();
        digitShow(last__);
        __delay_ms(2);
        C3_SetHigh();
        digitShow(digit__);


    }
}
```