Morse Code Translator Libraries

EE 2361 Final Project

by

Carson Dock, Henry Hein, Omar Sleiman, Nathan Hafey

Table of Contents

Presentation link:
https://docs.google.com/presentation/d/16cw---DpRb21tfV-e4JUhr7Lib5tPvJCltmI1f_P04k/edit?usp=sharing

Introduction:

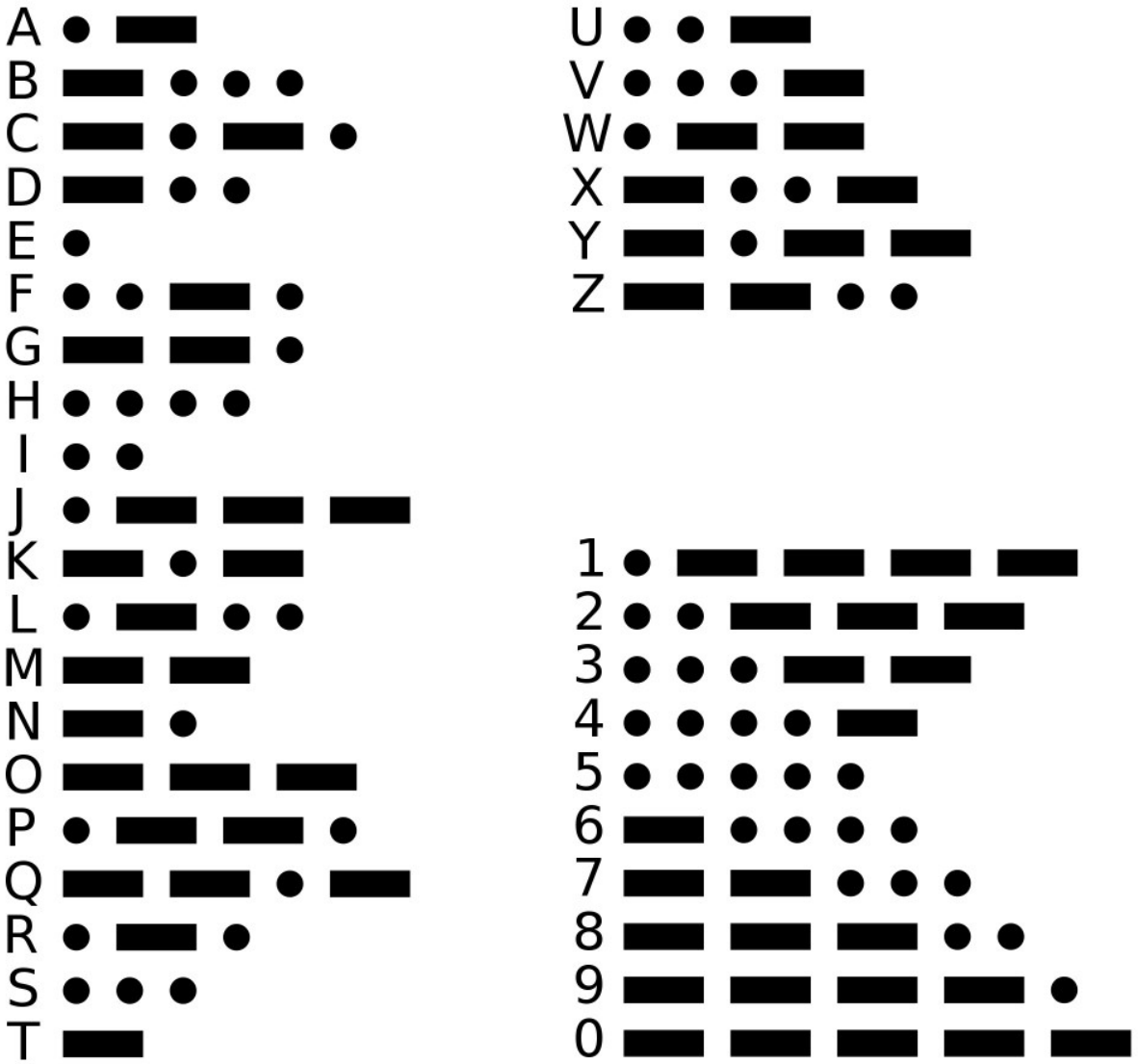
The goal of this project was to assemble a functioning Morse Code Translator that is based off of the international translation of Morse Code. This Library is based on the PIC24 microcontroller architecture and utilizes an ST7066 LCD display to display the Morse code. You can also use an HD44780 Display as their communication methods are cross compatible.

The library is broken into parts for ease of use. These parts being, Button input, Morse translation and LCD commands. If you plan on using a separate button, translation other than International morse code or a different communication protocol it would be easy to swap out one of these parts for the new desired section or modify code to make it usable.

The overall functionality of this library is targeted towards Morse Code translation but each individual part could work individually as variable length button press input reading, LCD display commands and a dictionary that can store various keys (Translation Method).

International Morse Code

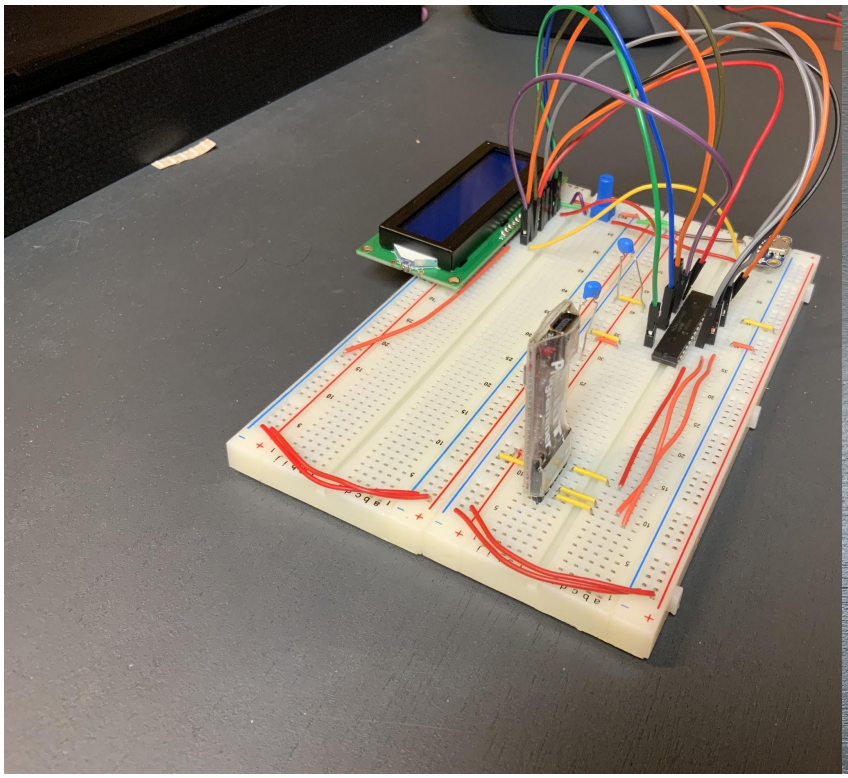
1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to seven dots.

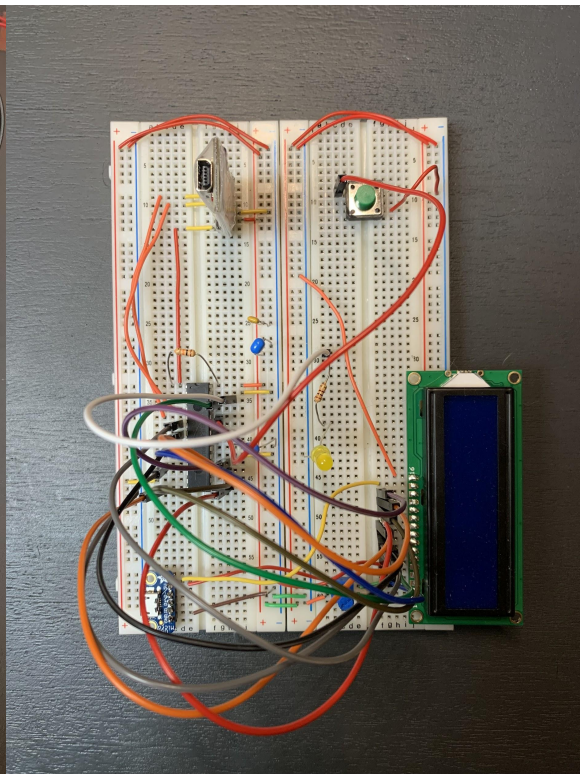


Hardware:

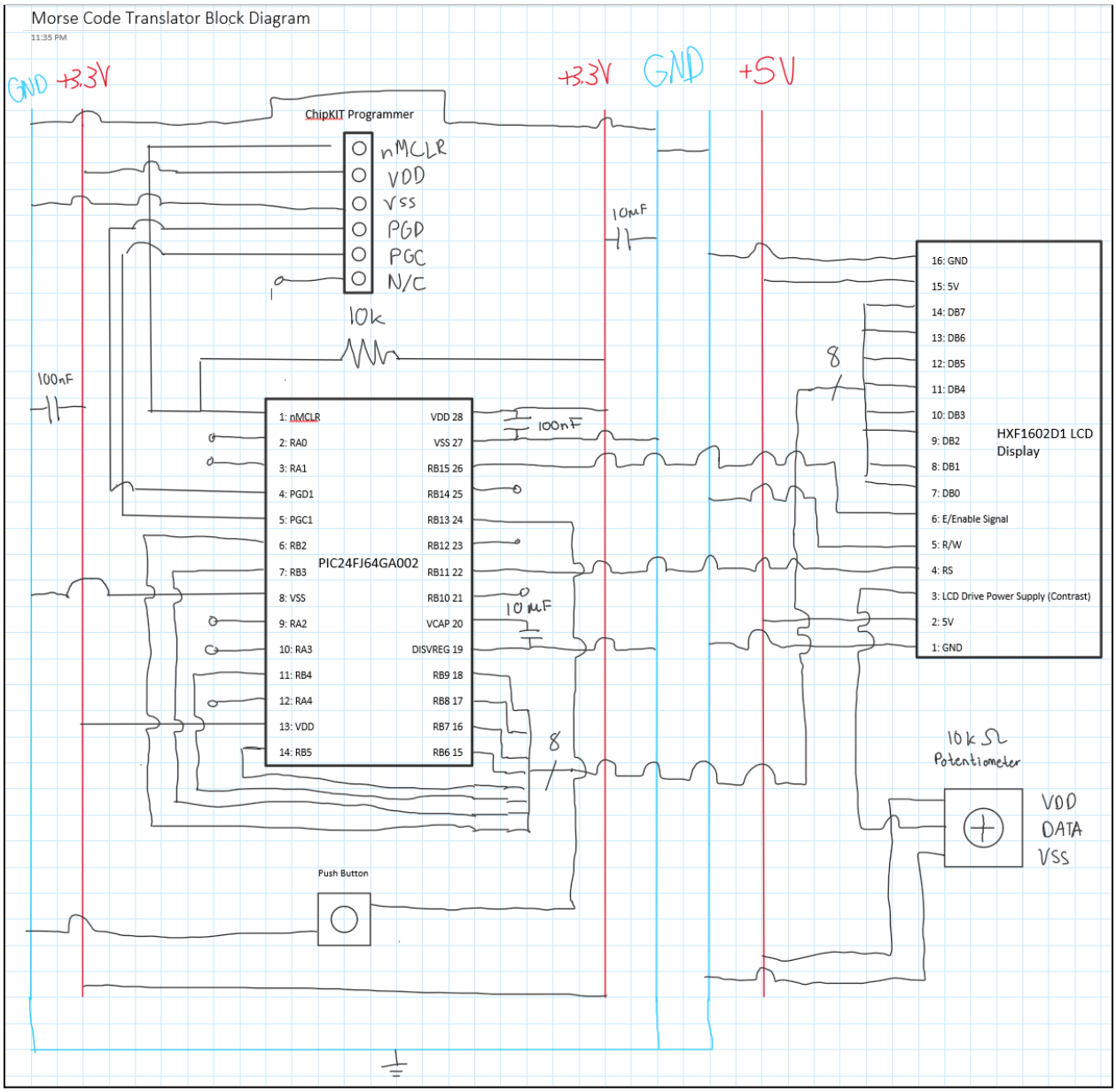
Hardware Components

Microchip PIC24FJ64GA002 Microcontroller
HXF1602D1 LCD Display
Display Datasheet: https://www.taydaelectronics.com/datasheets/A-1748.pdf
Display Driver data sheet: https://www.sparkfun.com/datasheets/LCD/st7066.pdf
Various wires, push buttons, 10K Potentiometer, resistors, and capacitors
Breadboard
ChipKit Programmer and MPLabX





Block Diagram



Functions:

Global Variables

volatile unsigned long int prevEdge
volatile unsigned long int curEdge

These variables are used in calculating the time between the press and release of the button, and also the time since the button was released. Used in IC1Interrupt() and main() functions.

volatile unsigned int count

This variable keeps track of which edge has most recently occurred on the button, a press or release (rising or falling).

volatile int rollover

Used to keep track of timer 2 rollover occurrences.

volatile char pulseArray[6] = {'0','0','0','0','0','0'};

This array contains the character which is currently being translated. Each element of the array is either a short (‘S’) or long (‘L’) pulse, or a “none” character, which signifies that the character’s morse code is not 6 characters long. This array is used to translate the pulses into characters which can then be displayed.

volatile int tempArray

This variable is simply the current index of the pulseArray, used in the IC1 interrupt. This variable is reset to 0 when the end-of-char signal is received.

Generic Delay Functions

```
void delay(int ms){ //delay in milliseconds
   int i;
   for(i=0; i<ms; i++){
      asm("repeat #15993"); //1 cycle to load and prep
      asm("nop"); //15993+1 cycles
   }
}
```

The delay()function creates a delay in milliseconds. The functions take an integer, which is the duration of the delay in milliseconds. This function is used in the setup, initialization, and communication functions.

Setup and Initialization Functions

```
void setup(void)
{
   //Executes Once
   CLKDIVbits.RCDIV = 0;  //Set RCDIV=1:1 (default 2:1) 32MHz or FCY/2=16M
   AD1PCFG = 0x9fff;
   TRISA = 0b0000000000011111;  //set port A to inputs,
   TRISB = 0b0010000000000011;  //and port B to outputs
   LATA = 0xffff;              //Set all of port A to HIGH

}
```

The setup() function initializes the PIC24 Microcontroller, sets the pins to digital, PORTA to inputs, PORTB to outputs, and sets the clock speed to 16MHz.

```
void initPushButton(){
   AD1PCFG = 0xffff; //set all pins digital
   __builtin_write_OSCCONL(OSCCON & 0xbf); // unlock PPS
   RPINR7bits.IC1R = 13;  // Use Pin RP8 = "8", for Input Capture 1 (Table 10-2)
    __builtin_write_OSCCONL(OSCCON | 0x40); // lock   PPS
   TRISBbits.TRISB13 = 1; //set RB13 to input, this is the button input
   CNPU1bits.CN13PUE = 1; //set pull up resistor on RB13 (pin 24)

   T2CON = 0; //reset timer 2 configuration register
   T2CONbits.TCKPS = 0b11; //set prescaler to a value of 256
```

```
    TMR2 = 0; //reset timer
    PR2 = 62500-1; //sets up the period to be 1s
    // 1s/(62.5ns*256)=PR2
    //62500 is the value we get by doing the calculation, then subtract 1 because of the cycle it
takes for
    //the PIC24 to realize the change
    IEC0bits.T2IE = 1; //enable the timer 2 interrupt



    IC1CON = 0; //reset internal state of IC1
    IC1CONbits.ICTMR = 1; //use TMR2 for capture source
    IEC0bits.IC1IE = 1; //enable IC1 interrupt

    T2CONbits.TON = 1; //turn on timer
    IC1CONbits.ICM = 0b001; //Turn on and capture rising AND falling edge
}
```

The initPushButton() function sets up the button and the corresponding timers. This function also sets up the IC1 input capture peripheral.

```
void initLCD(void){
    TRISB = TRISB & 0b0111000000000011;
    delay(100);
   //Turn on the display, cursor
    send_CMD( DISPONOF|0b111);
   //Clear the display
    send_CMD(CLRDISP);
   //Return the cursor to the far left of the screen
    send_CMD(RTNHOME);
    delay(2);
}
```

The initLCD() function sets up the LCD screen, and prepares it for receiving and displaying the translated Morse code.

Timer 2 Interrupt

```
void __attribute__((interrupt, auto_psv)) _T2Interrupt(){
   rollover++; //increment rollover, indicating a rollover has occurred
   _T2IF = 0; //clear TMR2 interrupt flag
}
```

The T2Interrupt() function simply updates the rollover global variable each time a rollover occurs. Required for calculating button press/release times.

Input Capture Interrupt

```
void __attribute__((interrupt, auto_psv)) _IC1Interrupt(){


   _IC1IF = 0; //clear Input Capture 1 interrupt flag

   //We need this conditional so that the program doesn't mistakenly think the time in between pulses is a short pulse
   if(!count){ //the edge that just occurred is a rising edge, set prevEdge
      prevEdge = IC1BUF + (rollover*62500);
      count++; //increment count, so the next edge will be a falling edge
   }
   else{ //falling edge just occurred, we can calculate pulse lengths
      count = 0; //reset count
      curEdge = IC1BUF + (rollover*62500); //current edge = most recent rising edge from the buffer, + any rollover times
      if((curEdge - prevEdge)<125){ // (1ms/(62.5ns*256))*2 gives us the desired value for 2ms
         //debouncing block, do  nothing

      }
      else if(curEdge - prevEdge <=20833){ //short pulse has occurred, between 2ms and 333ms
         pulseArray[tempArray] = dot;
         tempArray++;
         TMR2=0;
         rollover=0;

      }
      else if(curEdge - prevEdge <=62500){ //long pulse has occurred, between 333ms and 1s)
         pulseArray[tempArray] = dash;
         tempArray++;
         TMR2=0;
```

```
            rollover=0;

        }

    }

}
```

The IC1Interrupt() function is the function that translates the duration of the button presses into pulses. Using interrupts and timers, the time at which the button is pressed is recorded, then the time at which the button is released is recorded. If the press/release combination occurs within 333ms of each other, a short press is detected. If the press/release is longer than that but within 1s of each other, a long press is detected. The pulse signal is then stored in a global array, which is used for further translation.

translateMorseCode- Translate Function

Function:

```
char translateMorseCode(char morse[6])
{
   int key;
   int index;
   for(key = 0; key < 38; key++)
   {
      for(index = 0; index<6; index++)
      {
         if(codes[key][index] != morse[index])
         {
            break;
         }
         if(index == 5)
         {
            return order[key];
         }
      }
   }
   return '#';
}
```

The Translate function itself is relatively simple. It relies on two global arrays, codes and order. (These are documented below) Codes is a 2D array that contains the series of dots and dashes for each character. Order contains the characters in the order they appear in codes.

The Translate function itself functions similar to a dictionary. After passing in a character array the function will search through codes looking for a match, if it finds one it will return that character, if not it will return a # to signify that no matches exist.

Order Array:

```
volatile char order[38] = {'A', 'B', 'C', 'D', 'E', 'F',
                           'G', 'H', 'I', 'J', 'K', 'L',
                           'M', 'N', 'O', 'P', 'Q', 'R',
                           'S', 'T', 'U', 'V', 'W', 'X',
                           'Y', 'Z', '1', '2', '3', '4',
                           '5', '6', '7', '8', '9', '0', '|', ';'};
```

The order array contains the letters and numbers that can be translated in alphabetical, then numerical order. This is important due to how keys are accessed in the translateMorseCode() function.

Codes Array:

```
volatile char codes[38][6] = {{dot, dash, none, none, none, none},
 {dash, dot, dot, dot, none, none}, {dash, dot, dash, dot, none, none},
{dash, dot, dot, none, none, none}, {dot, none, none, none, none, none},
 {dot, dot, dash, dot, none, none},{dash, dash, dot, none, none, none},
 {dot, dot, dot, dot, none, none}, {dot, dot, none, none, none, none},
{dot, dash, dash, dash, none, none}, {dash, dot, dash, none, none, none},
 {dot, dash, dot , dot, none, none},{dash, dash, none, none, none, none},
 {dash, dot, none, none, none, none}, {dash, dash, dash, none, none, none},
{dot, dash, dash, dot, none, none}, {dash, dash, dot, dash, none, none},
{dot, dash, dot, none, none, none},{dot, dot, dot, none, none, none},
 {dash, none, none, none, none, none}, {dot, dot, dash, none, none, none},
{dot, dot, dot, dash, none, none}, {dot, dash, dash, none, none, none},
 {dash, dot, dot, dash, none, none},{dash, dot, dash, dash, none, none},
 {dash, dash, dot, dot, none, none}, //done with letters
{dot, dash, dash, dash, dash, none}, {dot, dot, dash, dash, dash, none},
{dot, dot, dot, dash, dash, none},{dot, dot, dot, dot, dash, none},
{dot, dot, dot, dot, dot, none}, {dash, dot, dot, dot, dot, none},
 {dash, dash, dot, dot, dot, none},{dash, dash, dash, dot, dot, none},
{dash, dash, dash, dash, dot, none}, {dash, dash, dash, dash, dash, none},
{dash, dot, dash, dot, dash, none}, {dot, dot, dot, dash, dot, dash} };
```

The code array is a 2D array that contains the set of Morse Digits required for each translation. These are added in the same order as the alphabetical array.
If you desire to use a different translation such as “American Morse Code” this code’/s array is the only array/code that would need to be modified to accept a new translation.

send_CMD(int cmd)

Function:

```
void send_CMD(int CMD){
    int command = CMD*4;
   //Load command onto the pins
     LATB= LATB | (command);
    delay(1);
   //Enable pin turned on
    LATB = LATB|0b1000000000000000;
    delay(1);
   //Turn off pins
    LATB= LATB & 0b0110000000000011;
}
```

This code takes in a command that is listed in the controller’s documentation and then tells the LCD display to execute that command. The command is shifted to the left by two bits to correspond to the DB0 bit on the LCD display connected to RB2 (pin 6) on the PIC24.
Bit-masking is used in this function to turn on the enable bit, and clear bits that were used in the LATB register afterward.

send_Char(char myChar)

Function:

```
void send_Char(char myChar){
    int letter = myChar;
   //Have the character be masked onto the write command.
    letter = WRITE | letter;
  //Send the write command with the character to the LCD display
```

```
    send_CMD(letter);
}
```

This code takes a character as a parameter and uses the send_CMD function to write the character to the LCD display screen after masking the input into a new variable containing the WRITE command for the LCD display.

int main(void)

Function:

```
int main(void) {
    setup();
    initLCD();
    initPushButton();
    unsigned long int currentTime;
    while(1){
        currentTime=IC1BUF+(62499*rollover);
        if ((currentTime >= 187500 )&& (PORTBbits.RB13 != 0)){ //if it has been 3 seconds
since the last button press
            //Get character from array
            char translation=translateMorseCode(pulseArray);
            //if the end transmission code was entered, turn off all input.
            if(translation==';'){
                end=1;
                send_CMD(DISPONOF|0b100);
                first=0;
            }
            //If the start transmission code is entered, reset the display
            else if(translation=='|'){
                send_CMD(CLRDISP);
                send_CMD(DISPONOF|0b111);
                send_CMD(RTNHOME);
                end=0;
                first=0;
            }
            //if a valid character was entered, print out the character
            else if(translation !='#'&& end!=1){
            send_Char(translation);
            first=0;
```

```
        }
        //if no valid character has been entered for 6 seconds, send a space
        else if(first!=0 && end!=1){
            send_Char(' ');
            first=0;
        }
        //if no valid character has been entered for 3 seconds, increment first to signify that it
has been 3 seconds already
        else{
            first++;
        }
        int i=0;
       //Resets the pulse array
        for(i=0;i<6;i++){
            pulseArray[i]='0';


      }
       //reset the timers
       TMR2=0;
       rollover=0;
       tempArray=0;

   }
   }
   return 0;
}
```

The main function is pretty straightforward, thanks to the chunking of most of the code into separate functions in order to make it more reusable in the future. First, our PIC24 Microcontroller, LCD, and push buttons are initialized, and the variable current time, which is used to test if it has been 3 seconds, is initialized as well. After 3 seconds have passed since the last button press, it will first translate the pulse array into a character. If that character corresponds to the end transmission code, it will block the entering of any characters further. If the morse code entered was the start transmission code, it will reset the display similar to the initialization function for the LCD, and then allow characters to be entered again. Thirdly, if the morse code entered corresponds to a valid character and is not either of the special codes and is not in end transmission mode, it will print the character out onto the LCD display. After, it checks to see if no valid morse code has been entered after 6 seconds. If so, it will print a space to the screen. Finally, if no valid code has been entered for 3 seconds, it will increment the variable first, to signify that it has gone 3 seconds without inputting a valid code.

Functionality

To test functionality, and general proper circuit operation setup the circuit as mentioned in the Hardware section. The pin connections are below the diagram showing an outline of the circuit. Once you have made these connections connect your 5v power. This should allow you to control the contrast of your display using the potentiometer. After adjusting the contrast to an appropriate level, load the library and main function to the PIC24. After a few seconds with no button inputs the button should begin to send spaces to the display signifying no valid input was sent for about 6 seconds. This should verify the minimum functionality of the circuit.

In-Depth Example

As a more in-depth example showing the functionality of the project. Begin with the same setup as the basic functionality demo. Make all the connections, see Hardware for details on pin connections. Connect the LCD to 5V and program the microcontroller. This time we will feed an input through the button to cause a character to appear.

To avoid accidentally entering an invalid character please use the document found on page 2 in order to properly operate the circuit. Once you have selected a phrase, begin inputting the characters individually. Each short press should be around 20ms and the long press can last up to 1 second. The short presses correspond to a dot in the translation and a long press represents a dash.

For this example select the letter “L” from the translation guide. In order to get “L” to be sent to the LCD Display input a short press, followed by a long press, followed by two short presses. After this, wait for approximately 3 seconds as the display finishes the word cycle, confirming the translation and sending it to the display. The character “L” should appear where the cursor once was and the cursor should have moved to the next available space. If you wait another 6 seconds you should be able to see a space appear after the “L” as no secondary input has been detected. Continue this process until you have displayed your desired phrase or output.

In order to test every possible input it is a good idea to use a Pangram, followed by the digits 0-9. A recommended pangram is:

“The quick brown fox jumps over the lazy dog 0123456789”

The input for this is as follows:

- .... . --.- ..- .. -.-. -.- -... .-. --- .-- -. ..-. --- -..- .--- ..- -- .--. ... --- ...- . .-. - .... . .-.. .- --.. -.-- -.. --- --. ----- .---- ..--- ...-- ....- ..... -.... --... ---.. ----.

To test out the functionality of the end transmission mode and the start transmission mode, enter in the character CT without a break between them (this should be -.-.-) to test the start transmission mode. When entered correctly, the display should reset and the cursor will be on the left side of the display. To test the end transmission mode, enter the characters SK without a break in between them (this should be ...-.-). When this mode is entered, the cursor should disappear and no other characters should be entered.