# Morse Code to English Translation

Hesiquio Ballines

Sumin Park

# Description

Our original objective was to make a device that will translate regular text to Morse Code using a laptop or phone to type the text and send it to the Arduino. We would have also had a second Arduino that will be able to communicate with the first Arduino directly. The Morse Code would be displayed using a buzzer, LED's, and possibly an LCD screen. Input would have been taken from either a laptop for text, a button, or a photosensor.

We decided instead to remove some of the features that we felt weren't completely necessary, to allocate more time into debugging and working on properly implementing Morse Code to English translation.

**Removed Features**

1. **Removal of LED urgency levels**
    a. This could be implement easily but we already have an LCD screen to relay the message. In addition, we wanted to focus more on the Morse to English translation rather simply sending the code to the other Arduino
2. **Removal of Speaker and Photosensor**
    a. These are both features that were removed because we wanted to focus on Morse to English rather than English to Morse.

**Added Features**

1. **A Toggle Button and Status LED**
    a. The addition of a toggle button helped us run the same code in one Arduino.
    b. The two states are Receive Data and Send Data

2. **LCD Screen English output**
    a. Originally the LCD screen was going to be a feature that was going to be added if we had time. We realized that sending Morse and Displaying it in different outputs was simple and something that did not take as much effort and we settled on translating and displaying it in English on the LCD screen.

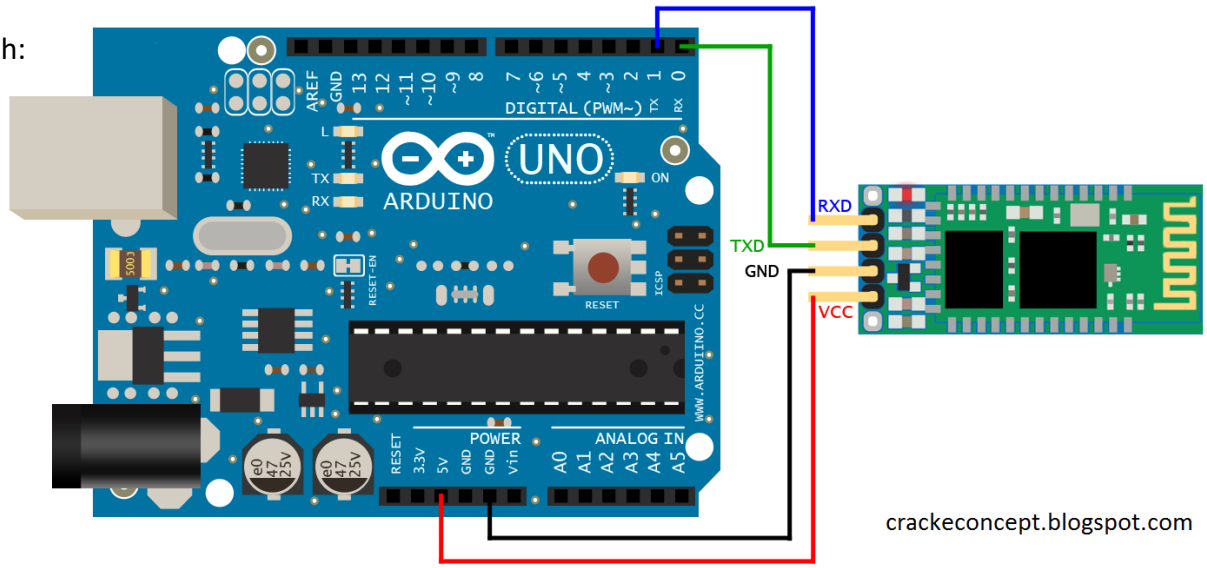
**Current Features Summed Up**

1. **Two Buttons**
    a. The Red button will be used to input the Morse Code
    b. The Yellow button will be used switch from receive and send data states
2. **LCD Screen**
    a. The LCD screen will be used to display the English translation one word at a time
3. **LED Indicator**
    a. The LED will be lit up if the state is currently set to blink when it is Read Mode
4. **Bluetooth Connection**
    a. This is used to communicate between two Arduinos
    b. Use of a phone or computer is possible

# Design

**NOTE: The Following Diagrams are not done by us and they were used for our final design**

Bluetooth:



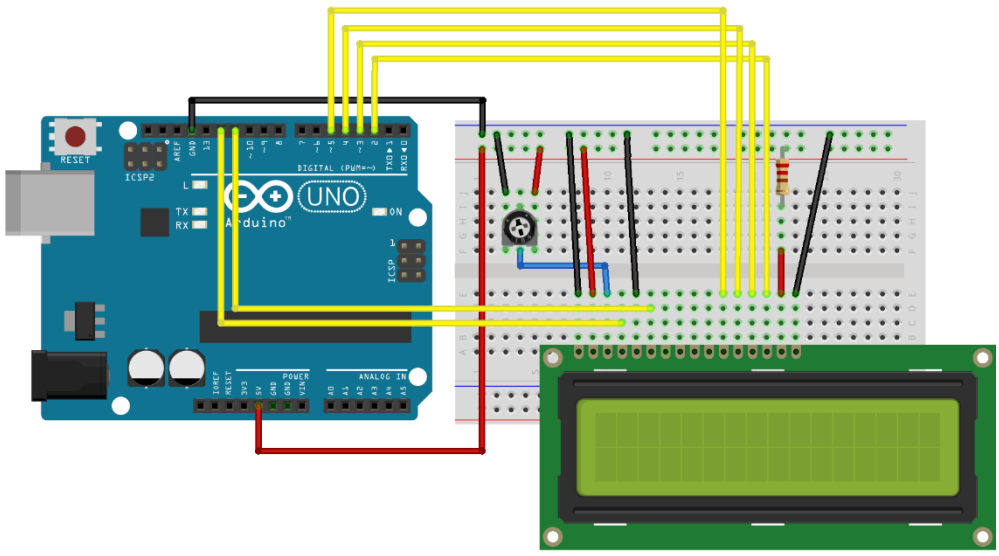

**For Bluetooth information we used the following resources**

Bluetooth basics: https://www.youtube.com/watch?v=sXs7S048eIo

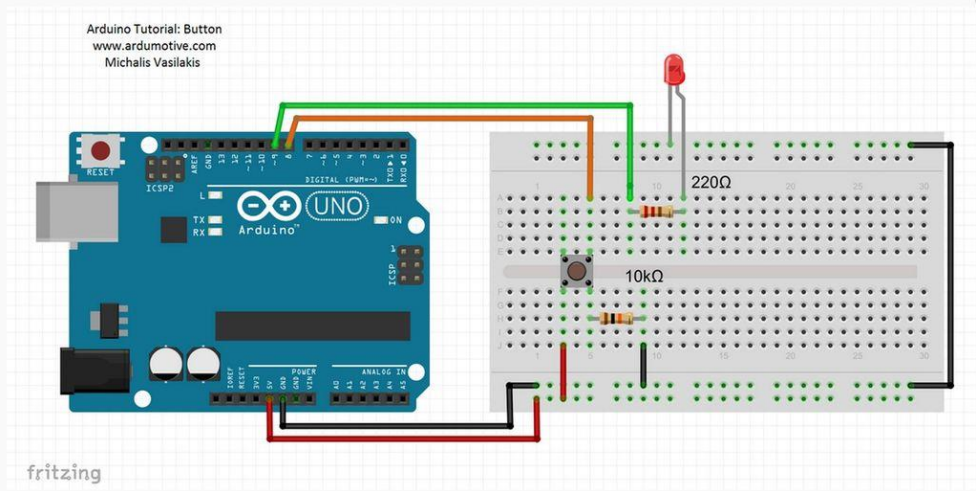
Bluetooth Communication between Arduinos: http://www.martyncurrey.com/arduino-to-arduino-by-bluetooth/

Used this Specific Bluetooth Module:
https://www.amazon.com/gp/product/B0093XAV4U/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

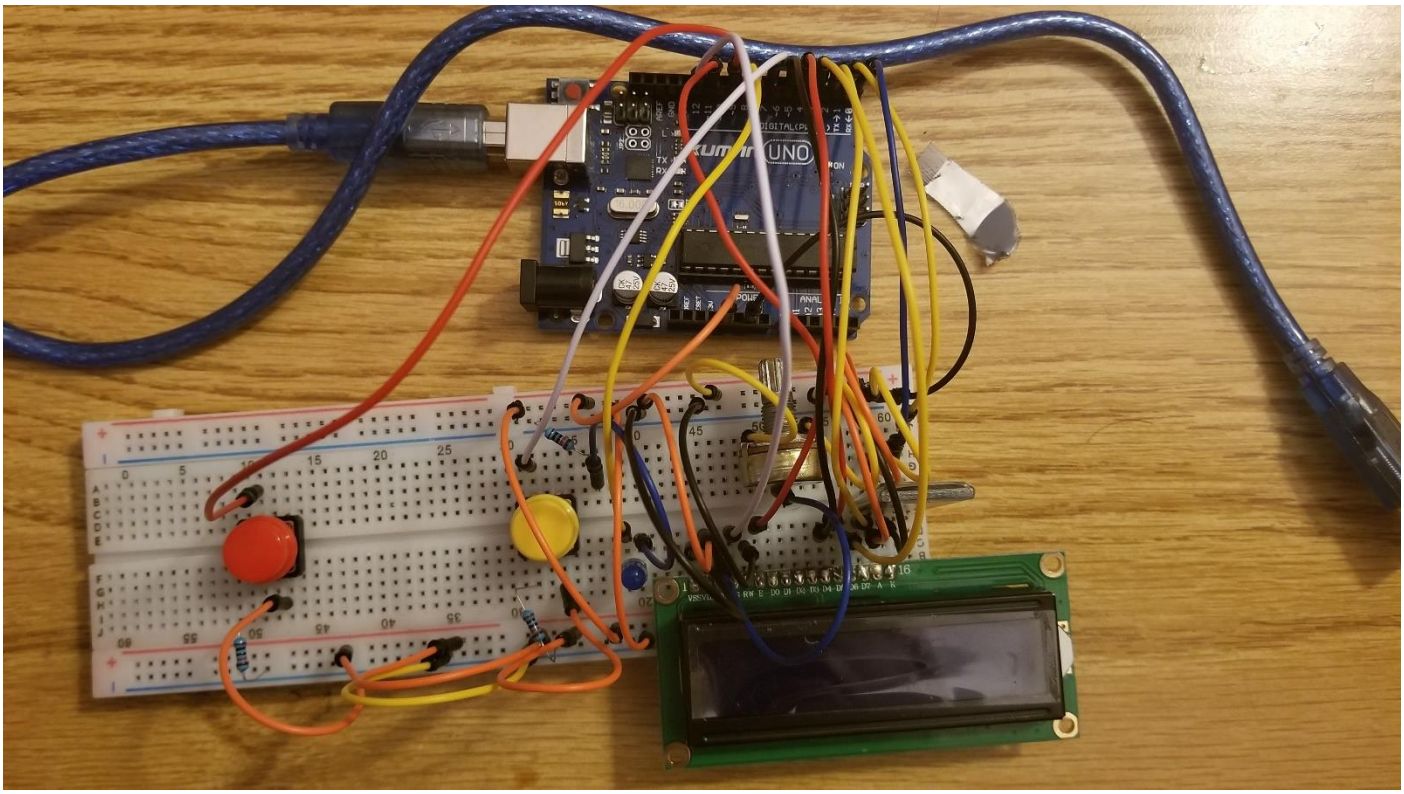
LCD Screen:



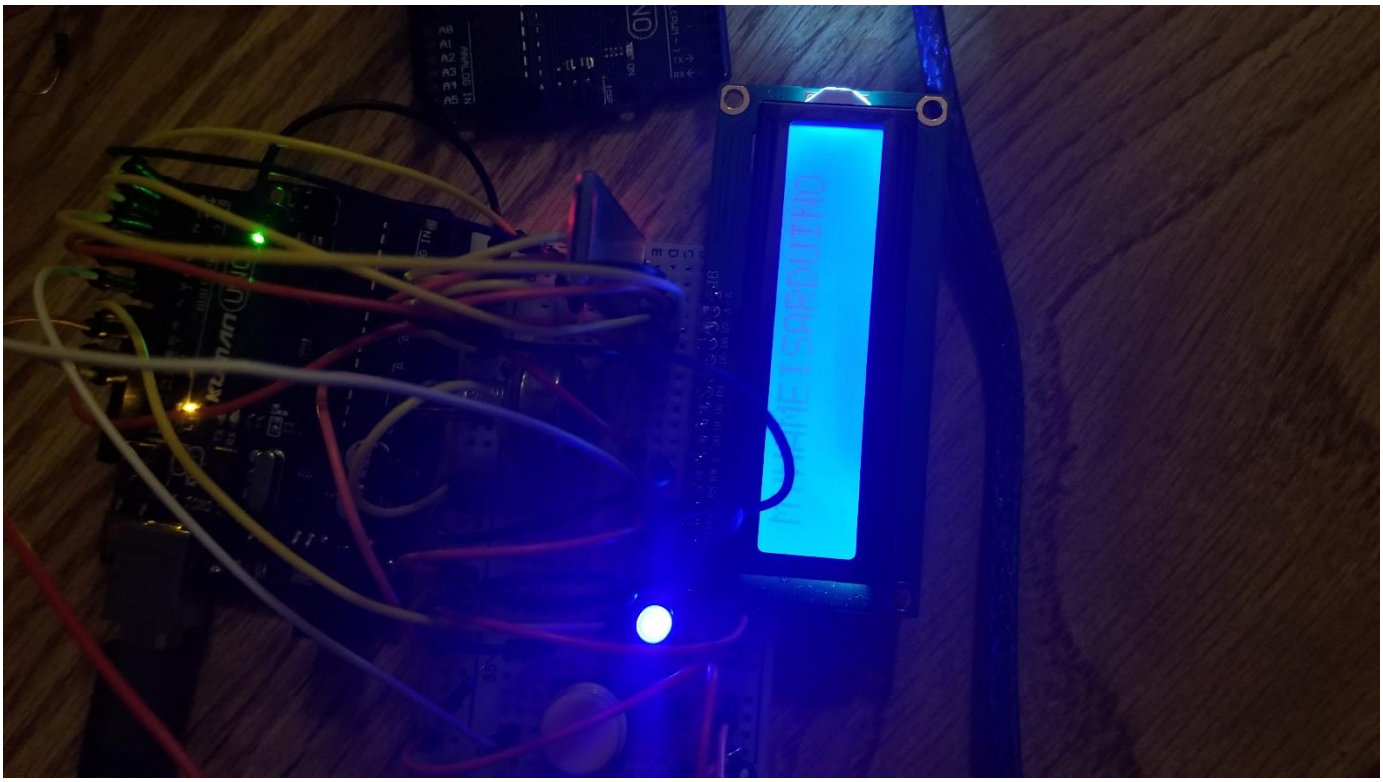
LED and Button:




## Everything Put Together



## Picture with output after translation



# Implementation and Challenges

## Using the Button to input Morse Code

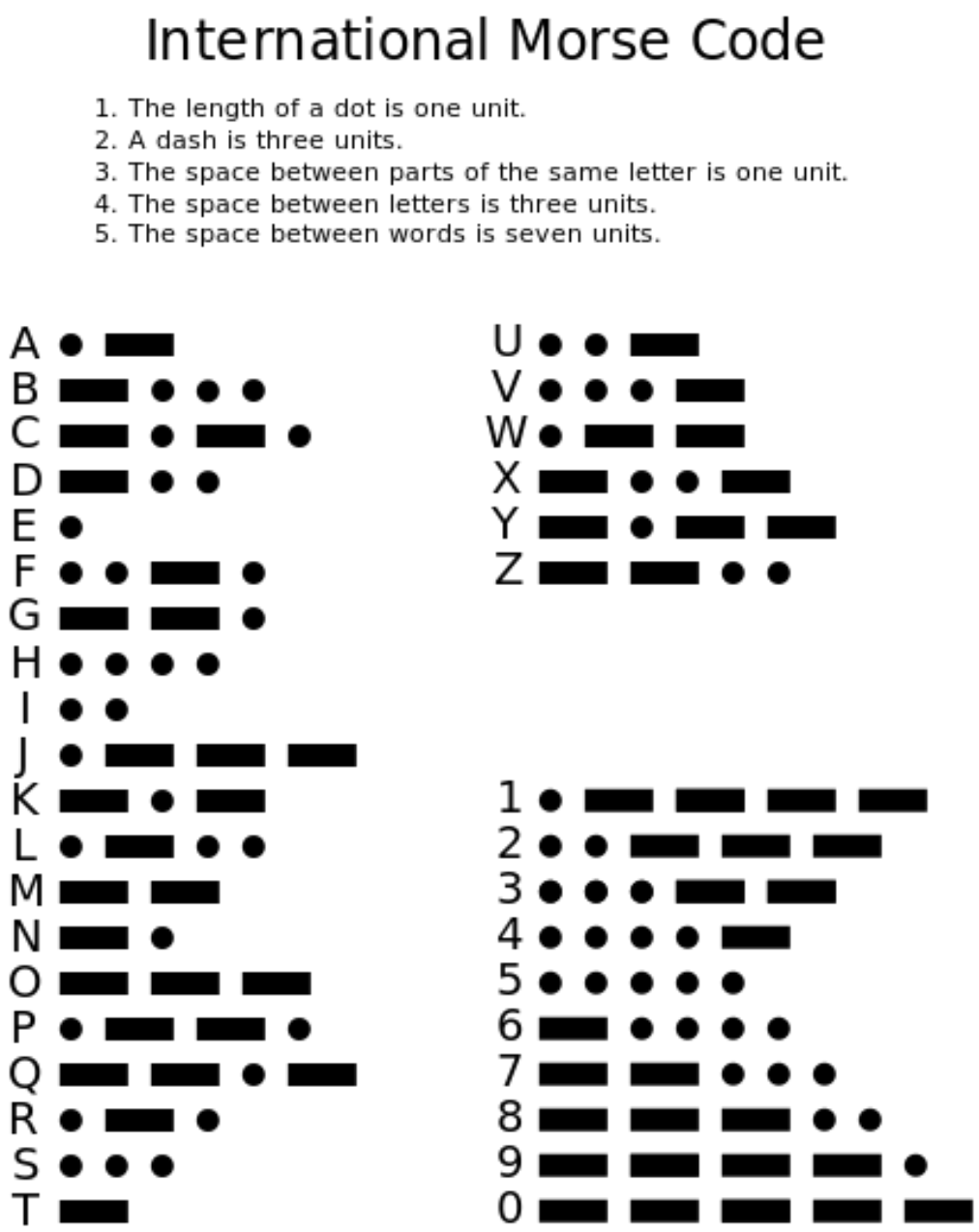
To make this happen we had to first research how Morse Code works. We found the diagram below and we used the descriptions as a start. We found a couple of very important pieces of information.

1. The length of a dot is one unit
2. A dash is three units
3. The space between letters is three units
4. The space between words is seven units

This is where we started for writing our code. One of the issues we ran into was figuring out how long “One Unit” should be. First, we started with one second being one unit and stored it into a variable. We then used this number inside of a couple of if statements. Based on the value of the integer we decided whether it should be a dot or a hold. This worked okay but it felt sluggish and this would become an issue with faster presses. In addition, the timing had to be very specific because of the way delays work so two fast taps would be registered as one tap for example. To get rid of this problem we had to sacrifice a little bit of processing power by lowering the delay and adjusting the ranges accordingly. The effect of lowering the delay value was the variable holding the time held would iterate a lot more and faster. This means more calculations but this was needed to make input more responsive and possible.

The other two things we had to figure out how to do is determining when there is a space and end of word. For this we used a similar method but instead of iterating a variable when the button is held down we iterated a variable whenever the button was released.

**Morse Code Table We Used for Reference**



## Problems with Button Input

One of the main issues was what we were going to do about the long pause indicating a new word. Originally we thought about using a 2D-Array storing a lot of potential words to be sent to be translated. However, the limitations of a 16x2 display caused us to believe that this approach was impractical. Memory could have also been a potential issue. To fix this we got rid of the 2D array in favor of a single array of characters that will be overwritten with each set of inputs. We also ran into an issue with the original implementation where the Change State button would be ignored almost completely. The only time the state change button could be pressed would be right after a Morse Code Button press. To fix this we decided that, after each set of Morse Code (one English character) we would halt the program until there was more input. This way the State Change button could be pressed at appropriate times.

## Using a Button to Change States

The Inter-arduino communication is the key feature of the project. In general cases, there are master bluetooth and slave bluetooth in an inter-arduino communication. One of the functionality that took a lot of effort in the project is configuring master / slave relation when both arduino devices are identical. We implemented Read mode / send mode button and state, which enables user to switch from one state to the other, and depending on the correct

configuration of the other arduino, devices are capable of sending morse code and recieving morse code.

In read mode, bluetooth device recieve signal from its master counterpart, and add it to the LCD display. The LCD display will also reset, or clear once the displaying text goes over the limit. The device also comes with translation functionality, therefore, even though the bluetooth signal is in morse code, the correct translated alphabet will be displayed on the LCD screen. In read mode, red button, which represents the push button of Morse Code, is disabled. That is, the arduino functionality is completely unaffected by red button input.

In send mode, bluetooth device will send signal to its slave counterpart, once the user takes a pause(which is calibrated to be about 750 ms) after making certain input consisting of long beep and short beep, the arduino will send signal. The LCD display will show current input from the red button, so that user can easily see the code they are sending. Once data is sent, the LCD will be reset, or cleared, which will visually assist the user whether the message is sent or not.

## Problems With State Change

Some of the challenges of developing working code for this projects in terms of state change includes multiple button glitch. If the code is not programmed in specific if statements, lot of delay() functions can cause unexplainable glitches. That is, since we want to make sure we can change state mode at any point of the operation, I divided the majority of code into two

while loops, which represents the two different readMode setting. Yet, there were still some problems. The yellow button was unresponsive for 70 unitseconds after becoming send mode. This turned out to be a minor inconvenience, since the Arduino machine ignores input once, and did not cause more serious affected glitches. User simply has to wait 70 unitseconds to turn back to read mode. Currently, project uses specific pre-determined threshold to determine between long beep and short beep. With more optimization of the code, a functionality to customize this time period could be added, and will make user to have much efficient experience using the Arduino project.

## Translating from Morse to English

This was the biggest challenge in terms of coding. As a start we had to figure out we were going to be displaying information on the LCD we settled on sending one character at a time after being translated. To achieve this, we had to figure out how we were going to handle the input we were given. We decided that the Morse input should be put into a string variable. This string variable will then be translated and sent to the other Arduino one letter at a time. In terms of the actual translation we created a look up table that looks like this:

```
//Defining the morse code alphabet that will be used as a lookup table
String morseAlphabet[36] = {"._ ", "_... ", "_._. ", "_.. ", ". ",
"..._. ", "__. ", ".... ", ".. ", ".___ ", "_._ ", "._.. ", "__ ",
"_. ", "___ ", ".__. ", "__._ ", "._. ", "... ", "_ ", ".._ ", "..._ ",
".__ ", "_.._ ", "_.__ ", "__.. ",".____ ","..___ ","...__ ", "...._ ","..... ","_.... ",
"__... ","___.. ","____. ","_____ "};
//End of Morse alphabet definition
```

We also created a function called translateLetter that takes in one string of Morse Code as a parameter. What this function does is it compares the Morse string with each possible alphabet character. One issue is that looking at every single possible string would not make sense so we optimized this function a little bit. The first optimization was limiting which words are actually compared by first comparing the lengths of the strings. If the strings are different lengths, then they are obviously not the same. In addition, instead of using a string compare function we created a loop that exits and goes to the next word as soon as a difference is found. This in theory should save a little bit of processing power. The comparison itself is just a loop that iterates through every character and breaks the loop if a difference is found. If both strings reach the whitespace character then you can return the character of the alphabet at the index of the loop counter (return alphabet[i]; )

## Problems with Translation

One of the issues we ran into while writing this function is the way we were trying to read strings from serial. Originally one Arduino was going to send a whole word in Morse Code and translate. We tried many things like taking in one character at a time then we tried doing a word at a time. This created some problems; how would we separate each letter and how would we translate and put it into a string.

There was also some incompatible data types and functions. When I tried using strlen or strcmp functions, they simply would refuse to work because of the way data is read from serial. To fix this we created our own stringLength function, that way we didn't have to fight with

datatype and the way things are stored. The other thing that simplified everything was that we simply did not need to send the Morse Code. The translateLetters function was already complete so we translated one character at a time in the first Arduino then it was sent to the Reading Arduino to be displayed on the LCD.

Side Note: The alphabet is stored as

char alphabet[37] = “ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890”;

## translateLetter Function

```
//Beggining of translateLetter Function***************
char translateLetter(char letterBefore[10])
{
  int wordLength;
  int wordLength2 = stringLength(letterBefore);

  for(int i = 0; i<36; i++)
  {
   wordLength = stringLength(morseAlphabet[i]);

   if(wordLength == wordLength2)
   {
    for (int j = 0; j<wordLength; j++)
    {
      if(letterBefore[j]!= morseAlphabet[i][j])
      {
        break;
      }

      if(morseAlphabet[i][j] == ' ' && letterBefore[j] =
      {
        //Serial.println(alphabet[i]);
        return alphabet[i];
      }
    }
   }

  }
  return '$';
}
//End of translateLetter Function***********
```

# Conclusion

We believe our project came out fine, there are a few potential improvements that we could implement with more time. Some of the features that were cut out are good ideas but with our team of two we had to focus on making the features we would have work well. There were some unexpected and very time-consuming problems like the translate function. The logic itself was not expected to be too difficult but because we changed the structure of how data was going to be sent, we had to change the function several times. In the future we could also implement a translate from English function and have a toggle for that.

Another change we would do is increasing the memory and processing power of the Arduino. One our ideas involved having a big array that could hold and be translated into long sentences had to be scratched because of memory limitations.

We tried our best to debug everything but I'm sure there's a combination somewhere that causes an error. I believe that adding other features would also cause some of our functionality to be an issue. For example adding a feature to translate from English to Morse would require more addition states to specify what we want to be output on the LCD.