Morse Code Arduino Machine

Kenshiro Iwaki, Osmar Hernandez

Kiwaki2, oherna25

kiwaki2@uic.edu, oherna25@uic.edu

University of Illinois at Chicago

CS 362 Computer Design

April 2, 2021

Abstract

Morse code is a way of representing alphabet, numerals, and punctuation marks with the dots (dit), dashes (dah), and spaces, according to the encyclopedia Britannica. One way of sending the code is with the use of buttons. A simple Arduino, without any expensive equipment, can be used to create a morse code machine and even translate the message. The machine can also be used to communicate in morse code with other devices.

Project Ideas

Overall project description

We are creating a morse code machine that translates morse code into plain English. The Arduino will understand the user's input as morse code and will show the equivalent message in English on the display. We will also send the message from Arduino via Bluetooth and send it to a smartphone. The Arduino will receive the morse code back and will display the message in English on the display. When the Arduino is not connected to a smartphone, it can still be used as a single player morse code translator.

Final Plan for Communication

A smartphone app will be used to connect the two devices via Bluetooth. There are multiple apps available for sending and receiving messages to the smartphone. The smartphone user will see the morse code and respond back in morse code. The morse code will be in text format with dot and dashes, so the user can translate it themselves. Since the Arduino can translate the message, the Arduino will see the translated English text instead of the morse code. This is still two devices communicating in morse code since the message was initially in morse code but was automatically translated for the user's convenience.

The two-way communication may allow the device to be an Arduino or not. It just needs accept the morse code from an Arduino and be able to send back a morse code. This means that the device sending the message doesn't care if it the receiving device was able to translate it, because it only communicates in morse code. This is original because the morse code translator ideas on the Internet is smaller scale in comparison without this two-way communication. The morse code class is also original since using a tree is a simple way to get the letter. Another way could be using a map/hash map, but a tree is easier to handle both the dot and dash codes because it doesn't count them.

Final Project Design: Input/Out

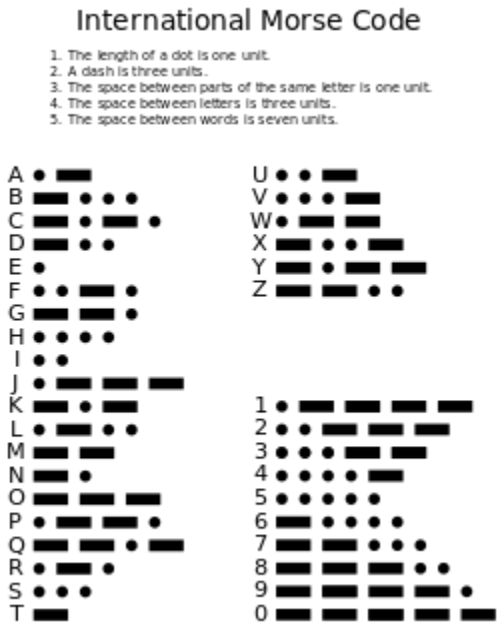
Two buttons are used for the input. One button is for entering the morse code and another one is for resetting the message. The 16x2 display will be the output to show the translation. A Bluetooth adapter will be another output device, sending the morse code to a computer. For the two-way communication, the adapter will also be used as the input to receive a morse code. The user's message will be displayed on the top screen while the receiving message will be displayed on the bottom screen.

Work report

We used a tree diagram as a reference to implement the translation, since it used a tree data structure for the logic. We found a better diagram than last time on a Medium article by Berk Ozer. The way it moves down the tree depends on whether the message was a dot or a dash.The current node in the tree is the current character. We have both the circuit and the code working for the single player mode. It translates the button presses to the correct letter.

The circuit of the single player mode is the same as lab 8, with 2 buttons and a 16x2 display. The buttons are connected to pin 2 and 3 and use 10k ohm resistors. The display uses pin 12, 11, 6, 7, 8, 9 for rs, en, d4, d5, d6, d7 respectively, and a potentiometer and a 220-ohm resistor. Use the breadboard to put them together and connect the jumper wires to 5V and GND of the Arduino. The Bluetooth module we bought has 4 connections: VCC, GND, TXD, and RXD. The adapter also should be put on the breadboard.

How to use/ implementation



Once the code is loaded, the machine will wait for the user to press button 1 for the first letter of the morse code. A short button press is for the dot code and a long button press is for the dash code. If the user presses button one after the previous button one press within a second, only the current letter will change. If button one wasn't pressed for around a second or more, a space will be added to the end of the message and that will be the next current letter. For example, if the user wants the message "HELLO" and it is currently "HELL", waiting for more than a second will change the message to "HELL ". Pressing button 1 within a second from there will change that space to a different letter. The user will press "---" to get "O" to finally get the message "HELLO".

Once the message is done, the user can long press button one for around a second or more to let the machine know that the message is done. Pressing button 2 will reset the message, so the user can start over when they made a mistake or completed a message. When the current letter is a space, another space won't be added. So, it will wait for a new word after the space like it does

at the beginning of the message. When the message is complete, the machine will send it to a smartphone.

Development timeline Description

Start of project	2/12/2021	
		Project brainstorm
Project idea submitted	2/26/2021	
		Code brainstorming
Circuit brainstorming		
		Hardware needed was finalized
Initial build of circuit completed	3/26/2021	
	4/2/2021	Updated project doc submitted
Presentation at expo	4/23/2021	
	4/30/2021	Final design doc due

The development for the one player mode was done during the spring break. The circuit for lab 8 was used since they both use two buttons and a 16x2 display. The MorseCode class was created, which translates the current letter. The class use the Node struct for each node, which contains a letter and the left and right pointers to other nodes. The class keeps track of the root node and the current node, so that the current node can be reset when moving to the next letter.

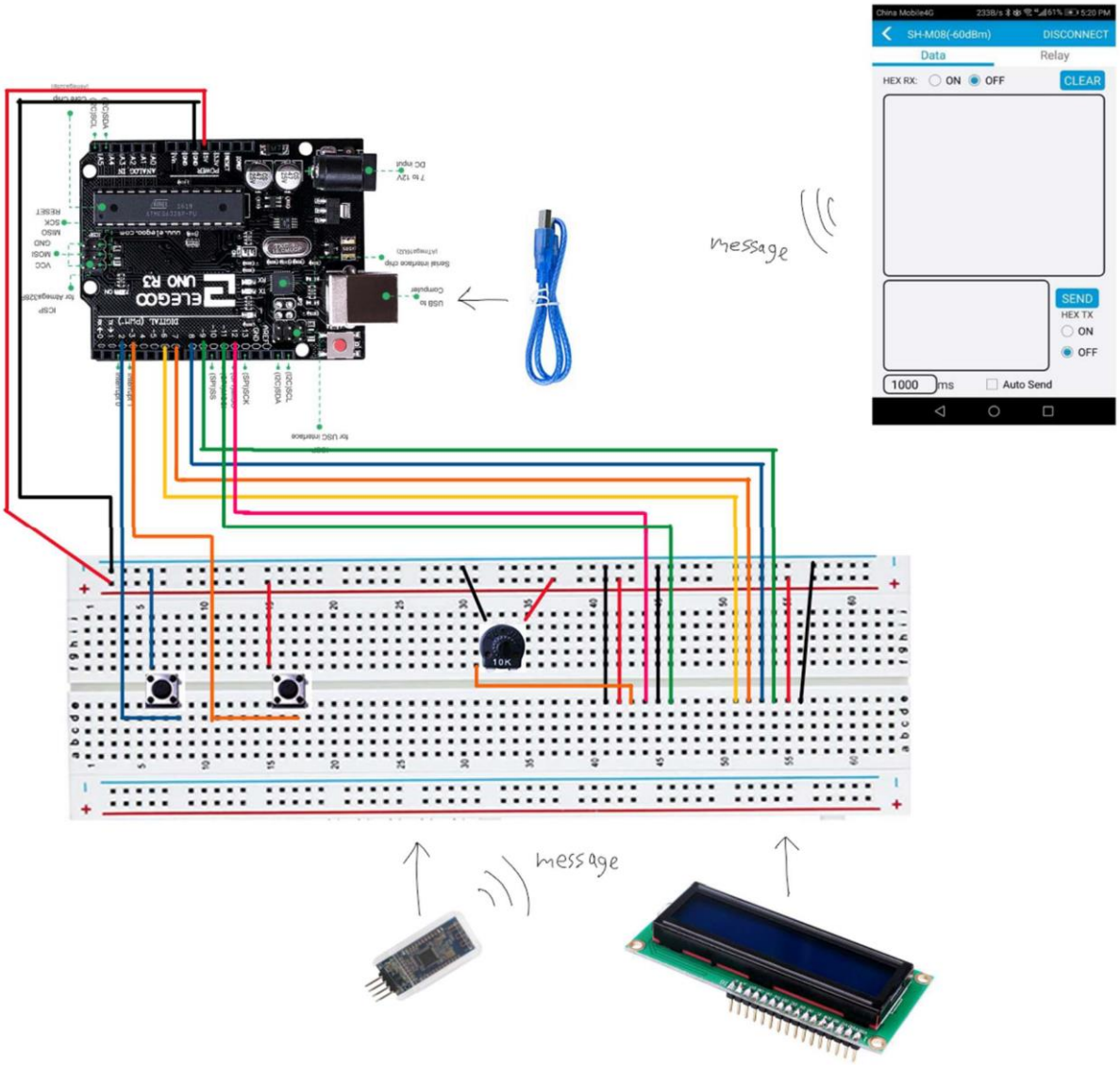
The implementation of getting the dot and dash codes from the button presses was done next. Counter variables were used to determine the time elapsed since the last button press every loop. That means how long the user pressed the button changed whether it was a dot code, dash code, or complete command. Same logic applies to the reset button, although that only does reset with instead of multiple options. The improvement was made at the start of the program to wait for the user to press button one for the first time, since not pressing the button also counts as a code to add a space to the message. This way, time is only counted starting from the button press. The code was tested on the circuit with success: the morse code machine was showing the translated message in real time! We bought the Bluetooth adapter during the break.

We will get the Bluetooth connection working to implement the sending and receiving of messages for milestone 5 part 2 by 4/23. Several improvements will be made like removing a space at the end of a completed message and scrolling the screen if the message is long. For part 3, we will be assigned to watch videos of two other teams. We will evaluate the videos using the Google forms after the videos are released after 4/23. Since we need to get the rest of the functionalities working by milestone 5, there shouldn't be a need for a design change in milestone 6. Most of the changes should be about what was done and new references. This time, all the code will be added as an .ino file as an attachment instead of just a code snippet and submit the pdf by 4/30.

Hardware list:

- 1 Arduino
- 2 buttons
- 1 16x2 display
- 1 breadboard
- 1 USB cable (for power)
    - Battery power is possible
- 1 potentiometer
- 2 10k ohm resistors
- 1 220-ohm resistor
- A collection of jumper wires
- 1 Bluetooth 4.0 adapter.
- Everything here except the Bluetooth adapter is included in the ELEGOOUNO Project Super Start Kit that is available on Amazon.
- The Bluetooth adapter we bought is DSD TECH HM-10 and was also bought on Amazon.

The following is the diagram of the morse code machine.



Complete code:

```
/*
 * Kenshiro Iwaki - 664864905
 * Project - Morse Code Machine
 * Description: Create a morse code and display the translated
message on the 16x2 display. Tap the left button for "dot",
long press for "dash". Press the right button to reset.
 *            Waiting at least 1 second after the last left button
press will add a space to the message. Long pressing the left
button for at least 1 second will stop the message.
 * Assumptions: I can use a similar circuit for lab8 for the
16x2 display and the buttons.
 * References: https://medium.com/swlh/how-tree-data-
structures-help-us-understand-morse-code-a95f6f7f2219
 *           https://www.boxentriq.com/code-breaking/morse-
code
 *
https://www.arduino.cc/reference/en/language/variables/data-
types/stringobject/
 *
https://www.arduino.cc/en/Tutorial/BuiltInExamples/Deboun
ce
 *
https://create.arduino.cc/projecthub/mayooghgirish/arduino-
bluetooth-basic-tutorial-d8b737
 */

#include <LiquidCrystal.h>

struct Node {
  char c;
  Node *left;  // dot
  Node *right;  // dash
};

class MorseCode {
private:
  Node *root;
  Node *curr;  // node for the current character
  String message;
  int len;  // length of the message

public:
  MorseCode();  // set up the tree
  void dot();  // add dot to current character
  void dash();  // add dash to current character
  void addSpace();  // add a space to the current message
  String getMessage() const;  // get the entire message
  int getLength() const;  // get the length of the message
  void reset();  // reset to empty message
};
```

```
MorseCode::MorseCode() {
  Node *C0 = new Node{'0', nullptr, nullptr};
  Node *C1 = new Node{'1', nullptr, nullptr};
  Node *C2 = new Node{'2', nullptr, nullptr};
  Node *C3 = new Node{'3', nullptr, nullptr};
  Node *C4 = new Node{'4', nullptr, nullptr};
  Node *C5 = new Node{'5', nullptr, nullptr};
  Node *C6 = new Node{'6', nullptr, nullptr};
  Node *C7 = new Node{'7', nullptr, nullptr};
  Node *C8 = new Node{'8', nullptr, nullptr};
  Node *C9 = new Node{'9', nullptr, nullptr};
  Node *plus = new Node{'+', nullptr, nullptr};
  Node *equal = new Node{'=', nullptr, nullptr};
  Node *slash = new Node{'/', nullptr, nullptr};

  Node *H = new Node{'H', C5, C4};
  Node *V = new Node{'V', nullptr, C3};
  Node *F = new Node{'F', nullptr, nullptr};
  Node *unknown1 = new Node{' ' , nullptr, C2};
  Node *L = new Node{'L', nullptr, nullptr};
  Node *unknown2 = new Node{' ', plus, nullptr};
  Node *P = new Node{'P', nullptr, nullptr};
  Node *J = new Node{'J', nullptr, C1};
  Node *B = new Node{'B', C6, equal};
  Node *X = new Node{'X', slash, nullptr};
  Node *C = new Node{'C', nullptr, nullptr};
  Node *Y = new Node{'Y', nullptr, nullptr};
  Node *Z = new Node{'Z', C7, nullptr};
  Node *Q = new Node{'Q', nullptr, nullptr};
  Node *unknown3 = new Node{' ', C8, nullptr};
  Node *unknown4 = new Node{' ', C9, C0};

  Node *S = new Node{'S', H, V};
  Node *U = new Node{'U', F, unknown1};
  Node *R = new Node{'R', L, unknown2};
  Node *W = new Node{'W', P, J};
  Node *D = new Node{'D', B, X};
  Node *K = new Node{'K', C, Y};
  Node *G = new Node{'G', Z, Q};
  Node *O = new Node{'O', unknown3, unknown4};

  Node *I = new Node{'I', S, U};
  Node *A = new Node{'A', R, W};
  Node *N = new Node{'N', D, K};
  Node *M = new Node{'M', G, O};
  Node *E = new Node{'E', I, A};
  Node *T = new Node{'T', N, M};

  root = new Node{' ', E, T};
  curr = root;
```

```
  message = " ";
  len = 1;
}

void MorseCode::dot() {
  if (curr->left != nullptr) {
    curr = curr->left;
  }
  message[len - 1] = curr->c;  // update the last character
}

void MorseCode::dash() {
  if (curr->right != nullptr) {
    curr = curr->right;
  }
  message[len - 1] = curr->c;  // update the last character
}

void MorseCode::addSpace() {
  message += " ";
  len++;
  curr = root;  // reset the current character
}

String MorseCode::getMessage() const {
  return message;
}

int MorseCode::getLength() const {
  return len;
}

void MorseCode::reset() {
  curr = root;
  message = " ";
  len = 1;
}

// global variables
const int rs{11}, en{12}, d4{6}, d5{7}, d6{8}, d7{9};
const int btn1{4}, btn2{5};

int btn1State{}, btn2State{};
int btn1StatePrev{LOW}, btn2StatePrev{LOW};
int counter1{}, counter2{};
long debounceTime1{}, debounceTime2{};
const long debounceDelay{50};
bool finished{false};
char r_data{' '};  // receiving data from an external device
connected via bluetooth
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

```
MorseCode mc1;  // client morse code
MorseCode mc2;  // morse code from external device

void setup() {
  lcd.begin(16, 2);
  pinMode(btn1, INPUT);
  pinMode(btn2, INPUT);
  Serial.begin(9600);
}

void loop() {
  // read the state of the button press
  btn1State = digitalRead(btn1);
  btn2State = digitalRead(btn2);

  // get message from bluetooth device
  if (Serial.available() > 0) {
    r_data = Serial.read();

    if (r_data == '.') {
      mc2.dot();
    } else if (r_data == '-') {
      mc2.dash();
    } else if (r_data == ' ') {
      mc2.addSpace();
    }
  }

  if (btn1State != btn1StatePrev) {
    debounceTime1 = millis();
  }
  if (btn2State != btn2StatePrev) {
    debounceTime2 = millis();
  }

  if ((millis() - debounceTime1) > debounceDelay) {
    if (btn1State == HIGH) {
      counter1++;
      counter2 = 0;
    } else {
      if (counter1 < 15 && counter1 > 0 && !finished) {  // dot
        mc1.dot();
      } else if (counter1 >= 15 && counter1 < 100 && !finished)
{  // dash
        mc1.dash();
      } else if (counter1 >= 100) {
        finished = true;  // message is complete
        Serial.println(mc1.getMessage());
      }
      counter1 = 0;
      counter2++;
```

```
    }
  }

  // add a space to message after at least 1 second after last
button press (10 * 100 = 1000 millisecond = 1 second)
  if (counter2 >= 100) {
    if (mc1.getMessage() != " " && !finished) {
      mc1.addSpace();
    }
    counter2 = 0;
  }

  if ((millis() - debounceTime2) > debounceDelay) {
    // reset the message
    if (btn2State == HIGH) {
      mc1.reset();
      mc2.reset();
      counter2 = 0;
      finished = false;
    }
  }

  // print translated user's morse code
  lcd.setCursor(0, 0);
  lcd.print(mc1.getMessage());
  // print translated received morse code
  lcd.setCursor(0, 1);
  lcd.print(mc2.getMessage());

  if (finished) {

  }

  delay(10);
  lcd.clear();
  btn1StatePrev = btn1State;
  btn2StatePrev = btn2State;
}
```

References

DSD TECH HM-10 Bluetooth 4.0 BLE iBeacon UART Module with 4PIN Base Board for Arduino UNO R3 Mega 2560 Nano. (2011). Retrieved April 03, 2021, from https://www.amazon.com/gp/product/B06WGZB2N4/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1

ELEGOO UNO Project Super Starter Kit with Tutorial and UNO R3 Compatible with Arduino IDE Visit the ELEGOO Store. (2011). Retrieved April 03, 2021, from https://www.amazon.com/gp/product/B01D8KOZF4/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&psc=1

Morse code. (n.d.). Retrieved February 27, 2021, from https://www.britannica.com/topic/Morse-Code

Ozer, B. (2020, April 06). How tree data structures help us Understand morse code. Retrieved April 02, 2021, from https://medium.com/swlh/how-tree-data-structures-help-us-understand-morse-code-a95f6f7f2219