

Morse Code Decoder

ENGR 102 intro to electronics, Shoreline CC

HALIMA, GUSTI s.

Fall 2017

**Description of the problem and reason for building the circuit**

This project stores Morse Code by pressing buttons which will then be decoded and printed on the serial print screen. I chose to build this circuit to learn how to maintain values for the global variables which are set as counters. The circuit responds according to the values from the counter.

**Things to do with the circuit**

There are three buttons in the circuit. The button closest to the LED is the on or pause button for the decoder. If that button is pressed, the decoder can receive morse input. If no input is given but the decoder is on, the decoder automatically prints a space. To input the morse code, the button closest to the piezo buzzer is pressed either for a long or short period of time. When the decoder is paused, or the LED is off, the erase button in the middle can be used to erase a single character from the string.

**Prior knowledge needed to create this circuit**

This circuit requires basic understanding of how to connect the components. It is a little code heavy since keeping track of values and conditions are needed to control a single morse button which will then rely on the timing whether if anything or nothing currently happens on the circuit.

**Parts Required**:

1. Sparkfun / Arduino Board
2. Breadboard
3. 3 buttons
4. 1 LED
5. 1 330Ω resistor (for LED)
6. 3 1k Ω resistor (for buttons)

**Problem-Solving Approach**

Since the biggest challenge in this circuit is to use a single button for the morse code input, I initially used 2 buttons each for the dash and dots and another additional button to signal the next index in the array. After this worked, I then had to figure out a way to combine all the three buttons to one.

**Solved Problems**

1. Combining Buttons: to combine three buttons I had to determine what values for the timing I wanted as its functionality. For example, if the morse button was between 1 and under 10ms, it will read the morse as dot or short pressed and values larger than that is dash or long pressed. A delay of 1 ms was given in which after the first dash/dot the circuit can read the next value. If morse button is pressed after 50ms, it then stops placing values in an array. The array is then read and tied as a string to compare which letter corresponds by having the same string. Example, array = [ . (short pressed) , - (long pressed), [empty], [empty]] will print ‘a’ since the letter is represented by ‘.-‘ (dot, dash = short, long).
2. Timing issues: solving the timing issues was pretty much trial and error. I gave the next character / space delay only 50ms assuming that the user is quick to type morse code (It can easily be slowed down by changing the value of constant).
3. Buttons being too reactive: I learned from building this circuit that to decrease the sensitivity of some buttons, adding a delay is best. This can be best seen from the erase functionality. Had I not added a delay, it will erase lots of character. Adding a delay is also more convenient than having to fast tap the buttons.

**Schematic & PCB**

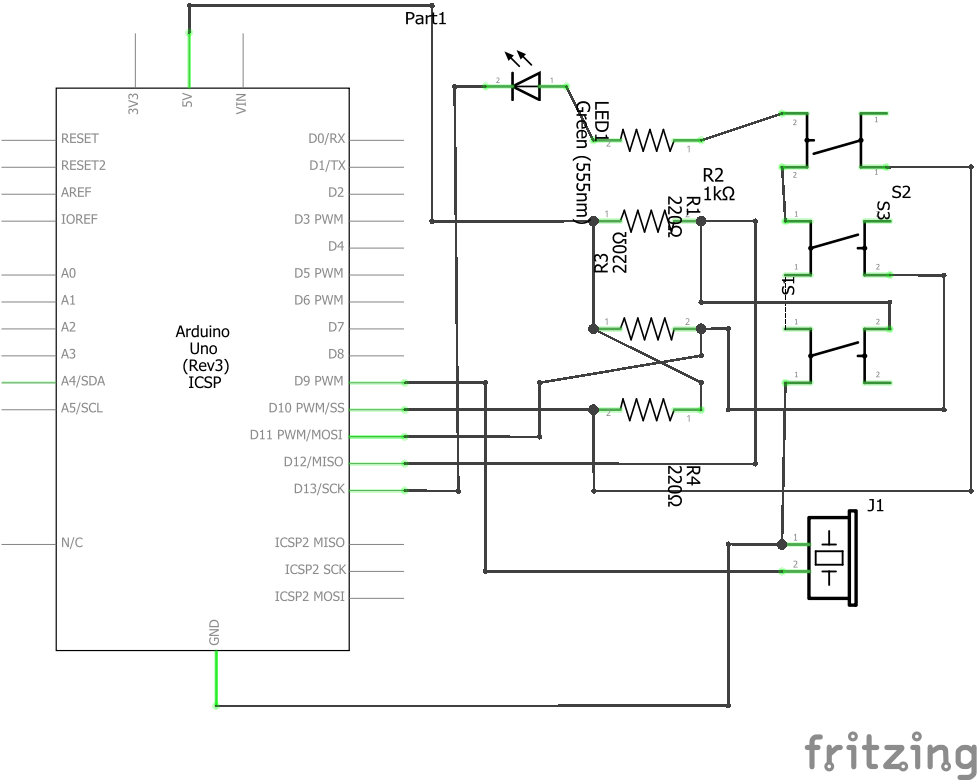
****

Figure . Schematic of board

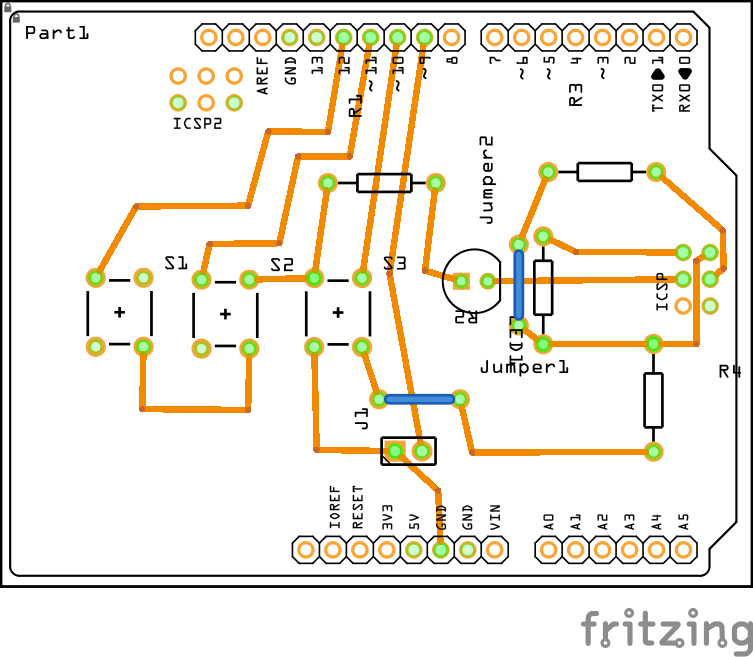
****

Figure . pcb of board

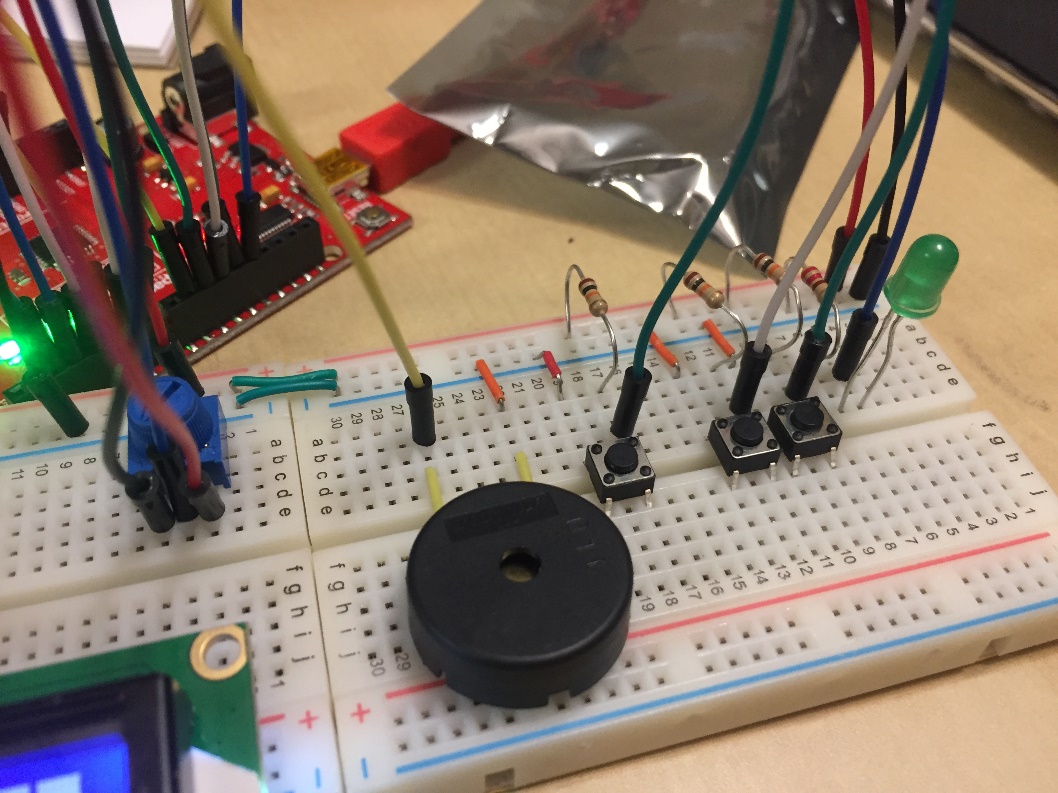
****

Figure . This circuit failed to implement an LCD so it only uses the top part

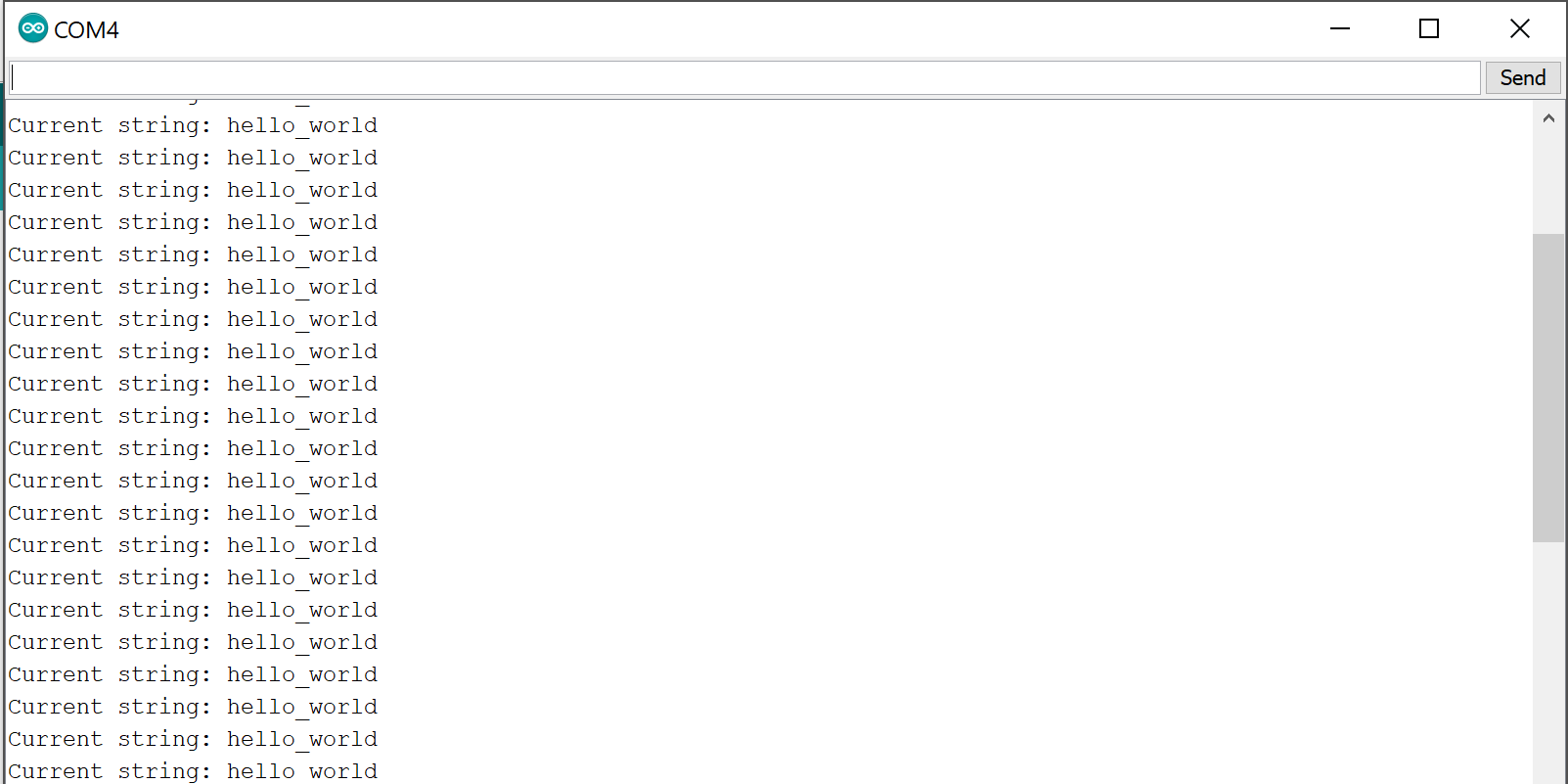
****

Figure . sample output

**Code:**

1. /\*
2. \* Name: Gusti Scarlett Halima
3. \* Date: November 1st 2017
4. \* Class: ENGR 102 intro to electronics (Fall 2017)
5. \* Software Version: Arduino 1.8.2
6. \* What it does: this project is a morse code decoder,
7. \* it accepts input from pressing the morse button and prints
8. \* a string to the computer.
9. \*/
11. /\*
12. \* LCD implementation failed
13. \*/
14. /\*#include <LiquidCrystal.h>\*/
15. /\*LiquidCrystal lcd(7, 6, 5, 4, 3, 2);\*/
17. **const** **int** onButton = 12; //this pin is connected to the on / pause button for the decoder
18. **const** **int** onLED = 13; //this pin signals an LED to high if onButton is LOW
19. **int** onState = 0; //this variable stores the onButton value: high = 0, low = 1 (A button has an active low logic or simply input/output state if the opposite of and LED)
20. **int** onCount = 0; //this variable will be tell the decoder whether what the onState is.
22. **const** **int** button = 10; //this pin is connected to the morse code button
23. **int** morseButton; //this variable stores the button state
24. **const** **int** buzzerPin = 9; //this pin is connected to the piezzo buzzer
26. **const** **int** erase = 11; //this pin is connected to the erase button
27. **int** eraseState; //this variable stores the erase button state which also has an active low logic

30. /\*
31. \* This project controls the timing of the decoder using global variables.
32. \* Each loop either increments a value or resets it. For example if the morse code button is not pressed
33. \* the countDelay variable increments. If it is pressed, the countDelay gets resets to 0.
34. \* This decoder reads the morse from the morseArray[] with max reading of four values. The 'count' variable
35. \* sets the index for the array and adds either a series of long or short (dashes and dots) values in it. It
36. \* increments value after a certain amount of delay.
37. \*/
38. //const int TIMING = 50; //this controls how fast the letters between the morse code is inputted, later it will be used as what value for 'countDelay' will be less than
40. **int** morseArray[4];
41. **int** count = 0;
42. **int** stringIndex = 0;
44. **int** letterInput = 1;
45. **int** printSpace = 0;
47. **int** countDelay = 0;
48. **int** buttonCount = 0; //This variable stores how long the button was pressed
50. String ourString = ""; //Initially starts with an empty string
52. **void** setup() {
54. Serial.begin(9600);
56. //pausing morse decoder/turning on morse decoder
57. pinMode(onButton, INPUT);
58. pinMode(onLED, OUTPUT);
60. pinMode(buzzerPin, OUTPUT); //Piezzo buzzer
62. pinMode(button, INPUT);
63. pinMode(erase, INPUT);
65. }
67. **void** loop() {
69. eraseState = digitalRead(erase);
70. morseButton = digitalRead(button);
71. onState = digitalRead(onButton);
73. /\*when onState is 0 or onButton is pressed, onCount is set to 0 which indicates that the decoder is on\*/
74. **if** (onState == LOW) {
75. **if** (onCount == 0) {
76. digitalWrite(onLED, HIGH);
77. onCount++;
78. delay(200);
79. }
80. **else** {
81. digitalWrite(onLED, LOW);
82. onCount--;
83. delay(200);
84. }
85. /\*Note that for both the if-else condition, a delay is added to prevent the button being too sensitive
86. when pressed. If it were left out, it would almost be impossible to only increment a single value since
87. we're accidentally pressing the button for the next loop.\*/
88. }
90. //Erase functionality if enabled only if LED is off / decoder is in paused state
91. **if** (onCount == 0) {
92. **if** (eraseState == LOW) {
93. **if**(stringIndex != 0) { /\*Erase button has another condition where stringIndex cannot be 0. This prevents messing up the string index count\*/
94. tone(buzzerPin, 440, 100);
95. ourString.remove(stringIndex - 1, stringIndex); /\*Remember that we only want to erase a single letter\*/
96. stringIndex--;
97. delay(300); /\*Same reason as previous, delay is added to prevent overpressing the erase button\*/
98. }
99. **else** {
100. tone(buzzerPin, 300, 100);
101. Serial.println("[string is empty]");
102. delay(300);
103. }
104. }
105. }
107. //LED is on
108. **if** (onCount == 1) {
109. initializeDecoder(); /\*\*/
110. }
111. **else** {
112. printSpace = 0; /\*I created a printSpace variable to indicate a starting point for when the
113. decoder just turned on. I personally felt this made the timing more easy by having a cue when I should start.
114. It basically gives a single low beep at the beginning.\*/
115. Serial.print("current string: ");
116. Serial.println(ourString);
117. }
119. }
121. /\*METHODS\*/
123. **void** initializeDecoder() {
124. **if** (count > 4) { /\*This condition makes sure that the morseArray only has 4 \*/
125. Serial.println("Exceeds morse limit");
126. count = 0; /\*The array resets\*/
127. countDelay = 0;
128. }
130. /\*This is the main timing value. When the decoder is on and it is left untouched for 50ms,
131. it will either signal the first beep to input the morse, count it as a letter, print a space, \*/
132. **if** (countDelay > 50) {
134. /\*This first condition prevents the last morse signal not being read (there was a previous unread value)\*/
135. **if** (buttonCount != 0) {
136. letterCount();
137. }
139. **else** {
140. **if** (count != 0) { /\*If user inputs a morse, count will never be zero and therefore the input is read\*/
141. tone(buzzerPin, 990, 50);
142. readArray(); /\*This method is printed only to show what values are passed for the morse code\*/
143. ourString += addLetter(); /\*Reads a letter and adds it to the string\*/
144. stringIndex++;
146. Serial.print("Current string: ");
147. Serial.println(ourString);
149. countDelay = 0;
150. count = 0;
151. }
153. **else** **if** (printSpace == 0){ /\*This signals first beep\*/
154. tone(buzzerPin, 500, 50);
155. printSpace++;
156. countDelay = 0;
157. }
159. **else**{ /\*printing space\*/
160. tone(buzzerPin, 770, 50);
161. readArray();
162. ourString += "\_"; /\*Currently an underscore to see the amount of space printed\*/
163. stringIndex++;
165. Serial.print("Current string: ");
166. Serial.println(ourString);
168. countDelay = 0;
169. count = 0;
170. }
171. }
173. }
175. /\*This condition increments the countDelay value when the morseButton isn't pressed\*/
176. **if** (morseButton == HIGH) {
177. countDelay++;
178. }
180. /\*This condition determines how long the morseButton was pressed\*/
181. //Long && short button
182. **if** (morseButton == LOW) {
183. buttonCount++;
184. countDelay = 0;
185. tone(buzzerPin, 880, 50);
186. }
188. /\*letterCount method is initialized after a 1 ms delay. This makes the morse code input sensitive
189. enough to store the long & short values in a single index of the morseArray\*/
190. **if** (countDelay > 0) {
191. letterCount();
192. }


196. /\*The following Serial print can be commented out to see the current values for delay value and button count\*/
197. //  Serial.print("Delay: ");
198. //  Serial.println(countDelay);
199. //  Serial.print("button: ");
200. //  Serial.println(buttonCount);
202. }
204. **void** letterCount() {
205. /\*The conditions are set to read the values of the button. Values larger than 0 and smaller than equal 10 sets the
206. reading as short pressed (dot) while values above 10 sees it as long pressed (dash)\*/
207. **if** (buttonCount > 0 && buttonCount <= 10) {
208. morseArray[count] = 1; //short button is symbolized as 1
209. count++;
210. Serial.println("short");
211. countDelay = 0;
212. buttonCount = 0;
213. }
214. **else** **if** (buttonCount > 10) {
215. morseArray[count] = 2; //long button is symbolized as 2
216. count++;
217. Serial.println("long");
218. countDelay = 0;
219. buttonCount = 0;
220. }
221. **else** {
223. }
224. }
226. /\*This method is only to show what the values are in an index\*/
227. **void** readArray() {
228. **for** (**int** i = 0; i < count; i++) {
229. **if** (morseArray[i] == 2) {
230. Serial.print("long, ");
231. }
232. **else** **if** (morseArray[i] == 1) {
233. Serial.print("short, ");
234. }
235. }
236. Serial.println();
237. }
239. /\*This methods determines what the array is trying to decode\*/
240. String addLetter() {
241. String currentArray = ""; //We first create a string based on the array which will then be compared to the letter that matches the value
242. **for** (**int** i = 0; i < count; i++) {
243. **if** (morseArray[i] == 2) {
244. currentArray += "-"; //2 is a value for dash '-'
245. }
246. **else** {
247. currentArray += "."; //1 is a value for dot '.'
248. }
249. }
251. **if** (currentArray.equals(".-")) {
252. **return** "a";
253. }
254. **else** **if** (currentArray.equals("-...")) {
255. **return** "b";
256. }
257. **else** **if** (currentArray.equals("-.-.")) {
258. **return** "c";
259. }
260. **else** **if** (currentArray.equals("-..")) {
261. **return** "d";
262. }
263. **else** **if** (currentArray.equals(".")) {
264. **return** "e";
265. }
266. **else** **if** (currentArray.equals("..-.")) {
267. **return** "f";
268. }
269. **else** **if** (currentArray.equals("--.")) {
270. **return** "g";
271. }
272. **else** **if** (currentArray.equals("....")) {
273. **return** "h";
274. }
275. **else** **if** (currentArray.equals("..")) {
276. **return** "i";
277. }
278. **else** **if** (currentArray.equals(".---")) {
279. **return** "j";
280. }
281. **else** **if** (currentArray.equals("-.-")) {
282. **return** "k";
283. }
284. **else** **if** (currentArray.equals(".-..")) {
285. **return** "l";
286. }
287. **else** **if** (currentArray.equals("--")) {
288. **return** "m";
289. }
290. **else** **if** (currentArray.equals("-.")) {
291. **return** "n";
292. }
293. **else** **if** (currentArray.equals("---")) {
294. **return** "o";
295. }
296. **else** **if** (currentArray.equals(".--.")) {
297. **return** "p";
298. }
299. **else** **if** (currentArray.equals("--.-")) {
300. **return** "q";
301. }
302. **else** **if** (currentArray.equals(".-.")) {
303. **return** "r";
304. }
305. **else** **if** (currentArray.equals("...")) {
306. **return** "s";
307. }
308. **else** **if** (currentArray.equals("-")) {
309. **return** "t";
310. }
311. **else** **if** (currentArray.equals("..-")) {
312. **return** "u";
313. }
314. **else** **if** (currentArray.equals("...-")) {
315. **return** "v";
316. }
317. **else** **if** (currentArray.equals(".--")) {
318. **return** "w";
319. }
320. **else** **if** (currentArray.equals("-..-")) {
321. **return** "x";
322. }
323. **else** **if** (currentArray.equals("-.--")) {
324. **return** "y";
325. }
326. **else** **if** (currentArray.equals("--..")) {
327. **return** "z";
328. }
329. /\*The else statement is never actually used but I decided to add it in case there's a possibility
330. of having a non alphabet combination. I chose a space instead of an empty string so I would still
331. increment the stringIndex\*/
332. **else** {
333. **return** " ";
334. }
335. }

Video: <https://www.youtube.com/watch?v=YDx5sWZCjo0>

Syntax Highlighter used:<http://www.planetb.ca/syntax-highlight-word>

Schematic created using Fritzing