

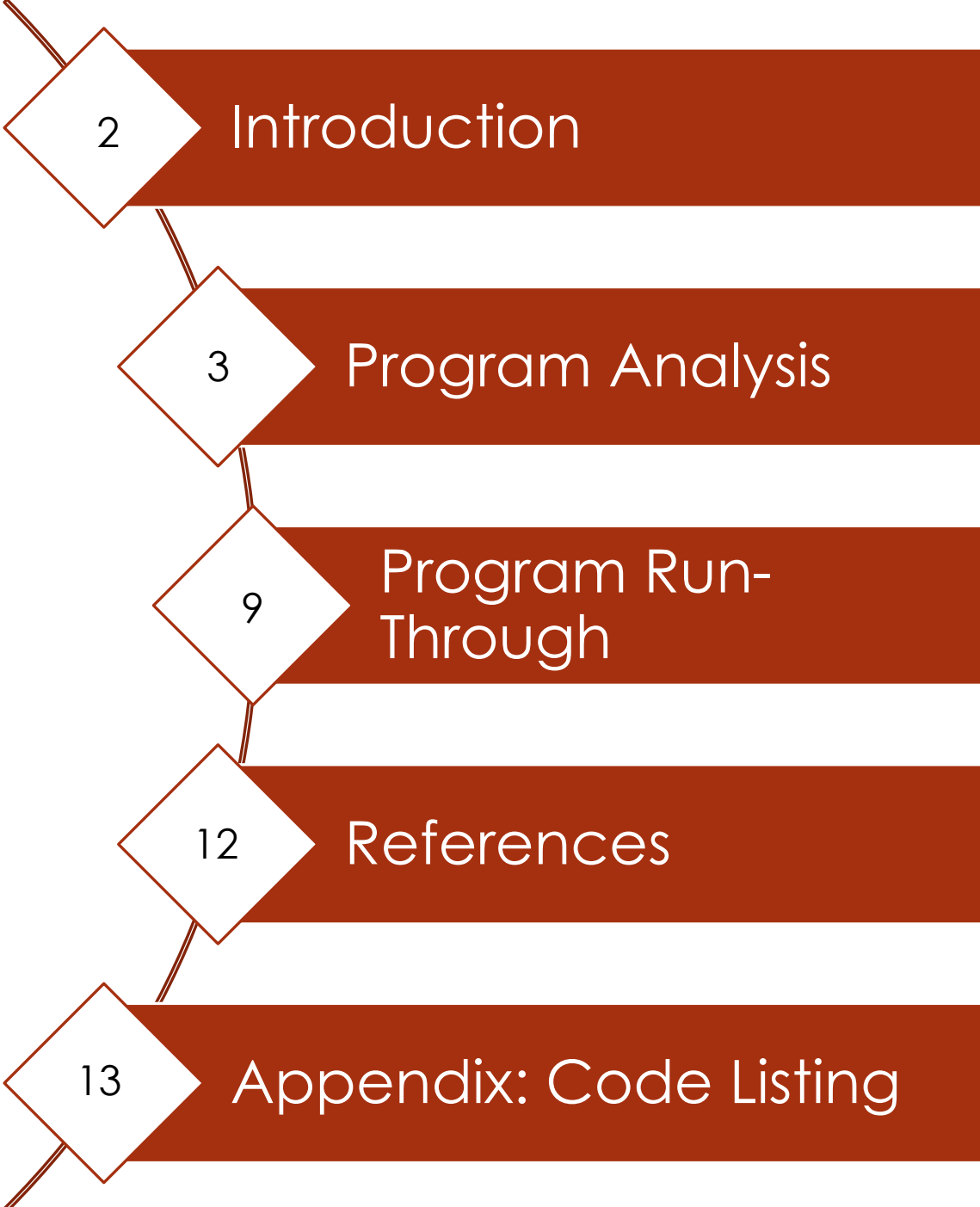


ARDUINO MORSE CODE

Eyecode4you



Contents

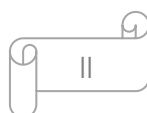


2	Introduction
3	Program Analysis
9	Program Run-Through
12	References
13	Appendix: Code Listing

1.0 Introduction

The program I developed is a Morse Code translation tool that utilizes components on the Arduino board in order to take an inputted message (In normal string format) and display each character in the message on the OLED screen, this is accompanied by a Morse Code translation being played on both the Piezo Buzzer and the Built-in LED on the Arduino board.

This functionality is controlled via the push button located below the OLED screen. Each character is printed to the screen in conjunction with the specific International Morse sequence for that specific character.



2.0 Program Analysis

The program source files contain the main program (**arduino_morse_code.ino**) which handles setting up and printing to the OLED screen with the use of the u8x8 graphics library. The handling of the inputted message is also performed here, as well as our main program loop.

Also contained within the program is a header file called **morse_check.h**, in here we define the behaviour of the 'dot' and 'dash' Morse sequences in conjunction with how they operate with each given alphanumeric character that can be translated. In here is also contained the code for checking each character in our input message and playing the corresponding Morse sequence through both the buzzer and built-in LED on the Arduino board.

arduino_morse_code.ino

At the beginning of the program, we setup our push button pin (D6). We then create a character array that will be used to store our message, we also create a length variable for our message which will be used for error handling later on.

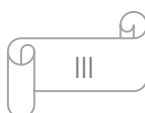
```
--  
11 const int b_pin = 6; //Push Button Pin  
12 char msg[17] = "tu dublin iot!"; //Word to convert to Morse  
13 int len = strlen(msg); //Length of word  
--
```

Figure 2.1: main globals

Beginning our setup() we set the push button as an input and setup our OLED, defining the orientation (L:20) and the font(L:21).

```
15 void setup() {  
16   pinMode(b_pin, INPUT); //Set push button as an input  
17  
18   //OLED SETUP  
19   u8x8.begin();  
20   u8x8.setFlipMode(1);  
21   u8x8.setFont(u8x8_font_chroma48medium8_r); //Set OLED Font  
--
```

Figure 2.2: Main setup()



The OLED screen integrated into our Arduino board supports a maximum of 16 characters (1x1 size) from left to right, if we try to print more than 16 chars on a single line of the OLED we would get overlap as characters from the beginning of the line will get overridden by characters exceeding the 16th position. To account for this, we check if the message length (len) is greater than 16 and if so, print error message and exit program.

```

23 //Due to size of OLED, message can only be up to 16 Characters in length
24 if(len>16){
25     u8x8.setCursor(0, 0);
26     u8x8.print("ERROR! MSG > 16!");
27     exit(1);
28 }
```

Figure 2.3: Main setup()

We convert characters in the message to upper case for unified checking later.

```

30 //Convert msg to upper case for checking characters
31 for(int i = 0; msg[i]; i++) msg[i] = toupper(msg[i]);
```

Figure 1.4: Main setup()

We display an introduction message at start of program in format:

Morse Code Msg:
//Message will be printed here
 Push Button
 To Begin

```

33 //DISPLAY INTRO MESSAGE
34 u8x8.setCursor(0, 0);
35 u8x8.print("Morse Code Msg:");
36 u8x8.setCursor(0, 4);
37 u8x8.print("Push Button");
38 u8x8.setCursor(0, 5);
39 u8x8.print("To Begin");
```

Figure 2.5: Main setup()

We begin our main loop by checking to see if the push button is pressed (its state), and if it is pressed (state == 1) we clear our message on screen (if there). We then iterate through each character in our message and perform morse_check (from morse_check.h) to check what character it is and play the corresponding Morse sequence on both the buzzer and built-in LED. After the morse sequence is played, we print the character on the screen and delay for 150ms before proceeding to the next character in our message. This repeats until all characters have been played

```

44 void loop() {
45   int b_state = digitalRead(b_pin); //Get state of button
46   if (b_state == 1) { //If button is pressed
47     u8x8.setCursor(0, 2);
48     u8x8.print(" "); //Clear message on screen
49     for(int x = 0; msg[x]; x++){ //For each character in message
50       morse_check(msg[x]); //Check character in message
51       u8x8.print(msg[x]); //Print character in message
52       delay(150); //Delay between each character in message
53     }
54   }
55 }

```

Figure 2.6: Main loop

morse_check.h

We start by defining the tone for our buzzer (Musical note C7) with frequency of 2093. We also create a global for the built-in LED pin.

```
11 #define NC7 2093 //Buzzer tone for dots and dashes
12
13 const int l_pin = LED_BUILTIN; //BUILTIN LED Pin
```

Figure 2.7: Header globals

We define a dot() function which turns on the built-in LED for its duration. We also construct the tone of the buzzer sound with our previously defined C7 note and the duration with a $1000/8 = 125\text{ms}$ duration (nDuration), this creates a quick pulse of the buzzer which is equivalent to a Morse dot sequence. We create sound through the buzzer with the tone() function, passing in the buzzer pin (5), the sound frequency (dot_sound), and the duration (nDuration).

We created the delay between 'dots' to be 1.30 times longer than the duration of a single 'dot', this creates a suitable interval between 'dots' played. We then call noTone() to stop the buzzer, turn off the LED, adding an 8ms delay to prevent it from appearing continuous.

```
15 // DOT - Dot tone + duration
16 void dot(){
17     pinMode(l_pin, OUTPUT); //Set LED as output
18     digitalWrite(l_pin,HIGH); //Turn on LED for duration of dot
19     int dot_sound = NC7;
20     int dot_duration = 8;
21     int nDuration = 1000/dot_duration;
22     int pNotes = nDuration * 1.30;
23     tone(5, dot_sound, nDuration);
24     delay(pNotes);
25     noTone(5);
26     digitalWrite(l_pin,LOW);
27     delay(8);
28 }
```

Figure 2.8: dot()

The dash() function operates much to the same effect of the dot() function, however here the duration is longer: $1000/4 = 250\text{ms}$ (Twice as long as a 'dot' which conforms to the Morse standard). The delay between 'dashes' is of the same factor (1.30) as the 'dots'.

```

30 // DASH - Dash tone + duration
31 void dash() {
32   pinMode(l_pin, OUTPUT); //Set LED as output
33   digitalWrite(l_pin,HIGH); //Turn on LED for duration of dot
34   int dash_sound = NC7;
35   int dash_duration = 4;
36   int nDuration = 1000/dash_duration;
37   int pNotes = nDuration * 1.30;
38   tone(5, dash_sound, nDuration);
39   delay(pNotes);
40   noTone(5);
41   digitalWrite(l_pin,LOW);
42   delay(8);
43 }

```

Figure 2.9: dash()

Here we have functions that define the Morse sequence for each letter character (See the morse chart in references). This is done in order of increasing time to transmit (Sequence Complexity) which is "eitsanhurdmwgvlfbkopxczjyq". This process is repeated for numbers and special characters.

```

45 //Letters in order of increasing time to transmit i.e. "eitsanhurdmwgvlfbkopxczjyq"
46 void Le(){dot();}
47 void Li(){for(int y=0; y<2; y++){dot();}}
48 void Lt(){dash();}
49 void Ls(){for(int y=0; y<3; y++){dot();}}
50 void La(){dot(); dash();}
51 void Ln(){dash(); dot();}
52 void Lh(){for(int y=0; y<4; y++){dot();}}
53 void Lu(){dot(); dot(); dash();}
54 void Lr(){dot(); dash(); dot();}
55 void Ld(){dash(); dot(); dot();}
56 void Lm(){for(int y=0; y<2; y++){dash();}}
57 void Lw(){dot(); dash(); dash();}
58 void Lg(){dash(); dash(); dot();}
59 void Lv(){Ls(); dash();}
60 void Ll(){Lr(); dot();}
61 void Lf(){Lu(); dot();}
62 void Lb(){Ld(); dot();}
63 void Lk(){dash(); dot(); dash();}
64 void Lo(){for(int y=0; y<3; y++){dash();}}
65 void Lp(){Lw(); dot();}
66 void Lx(){Ld(); dash();}

```

Figure 2.10: Letter Functions

Our `morse_check()` function accepts a character parameter (Current character in our message) and checks which character it is equal to. When it finds a match, the corresponding morse sequence for that character is played on both the buzzer and built-in LED. An exception to this is the space or a " " character, if this is encountered a delay of 25ms is executed instead. The order of character checking is based on Letter Frequency in the English language from highest to lowest: "ETAOINSHRDLCLUMWFGYPBVKJXQZ". We then check for numbers and special characters: ",?\\!/:;=+-_\"@".

```
156 //Check Letters in message and play tone
157 void morse_check(char L){
158     if(L == 'E'){
159         Le();
160     }else if(L == 'T'){
161         Lt();
162     }else if(L == 'A'){
163         La();
164     }else if(L == 'O'){
165         Lo();
166     }else if(L == 'I'){
167         Li();
168     }else if(L == 'N'){
169         Ln();
170     }else if(L == 'S'){
171         Ls();
172     }else if(L == 'H'){
173         Lh();
174     }else if(L == 'R'){
175         Lr();
176     }else if(L == 'D'){
177         Ld();
```

Figure 2.11: Character Checking

3.0 Program Run-Through

Before the button is pressed, the introduction message is printed to the screen.

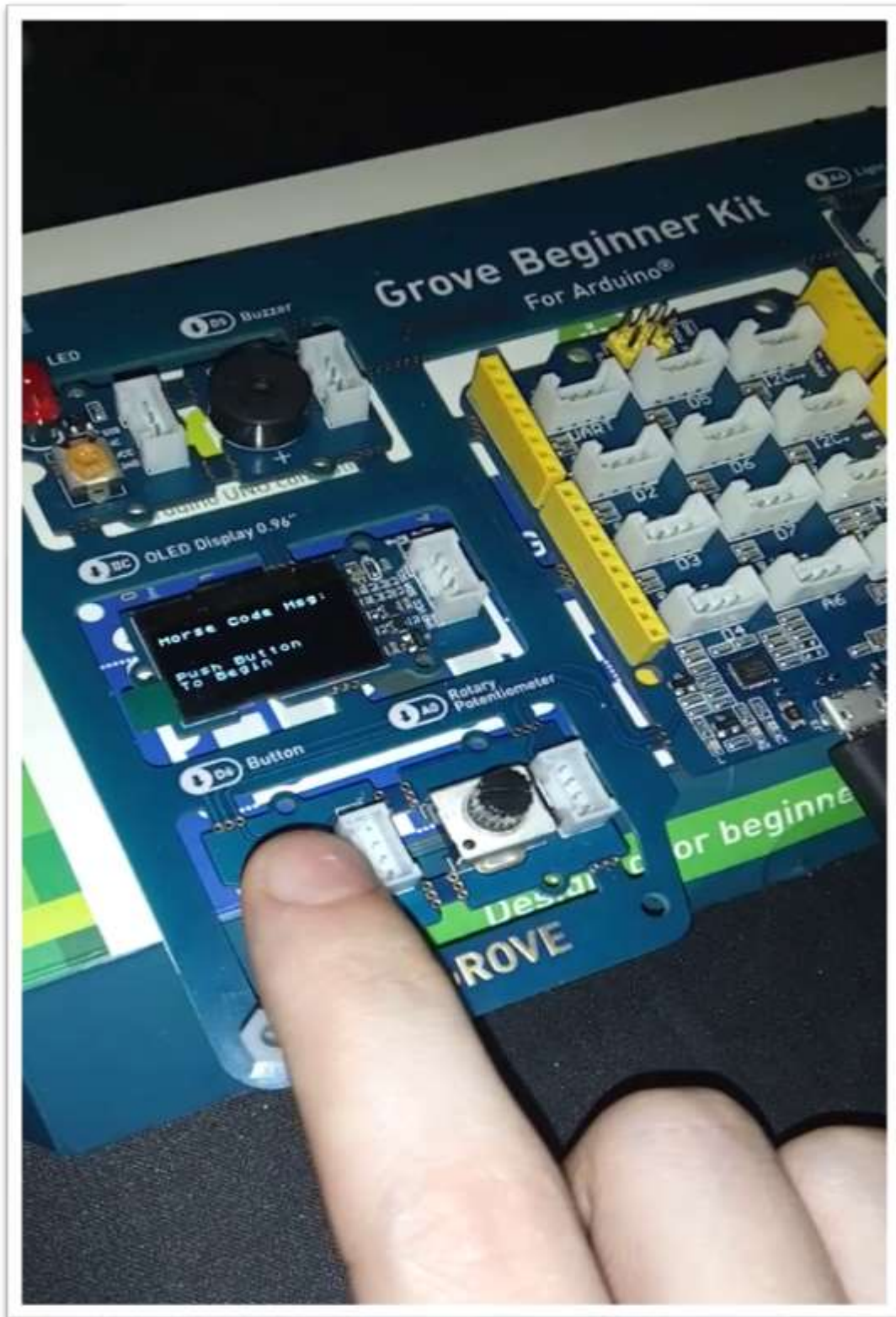


Figure 2.1: Program Start

When button is pressed, each character is printed to the screen, followed by the Morse sequence played on both the Buzzer and Built-in LED.

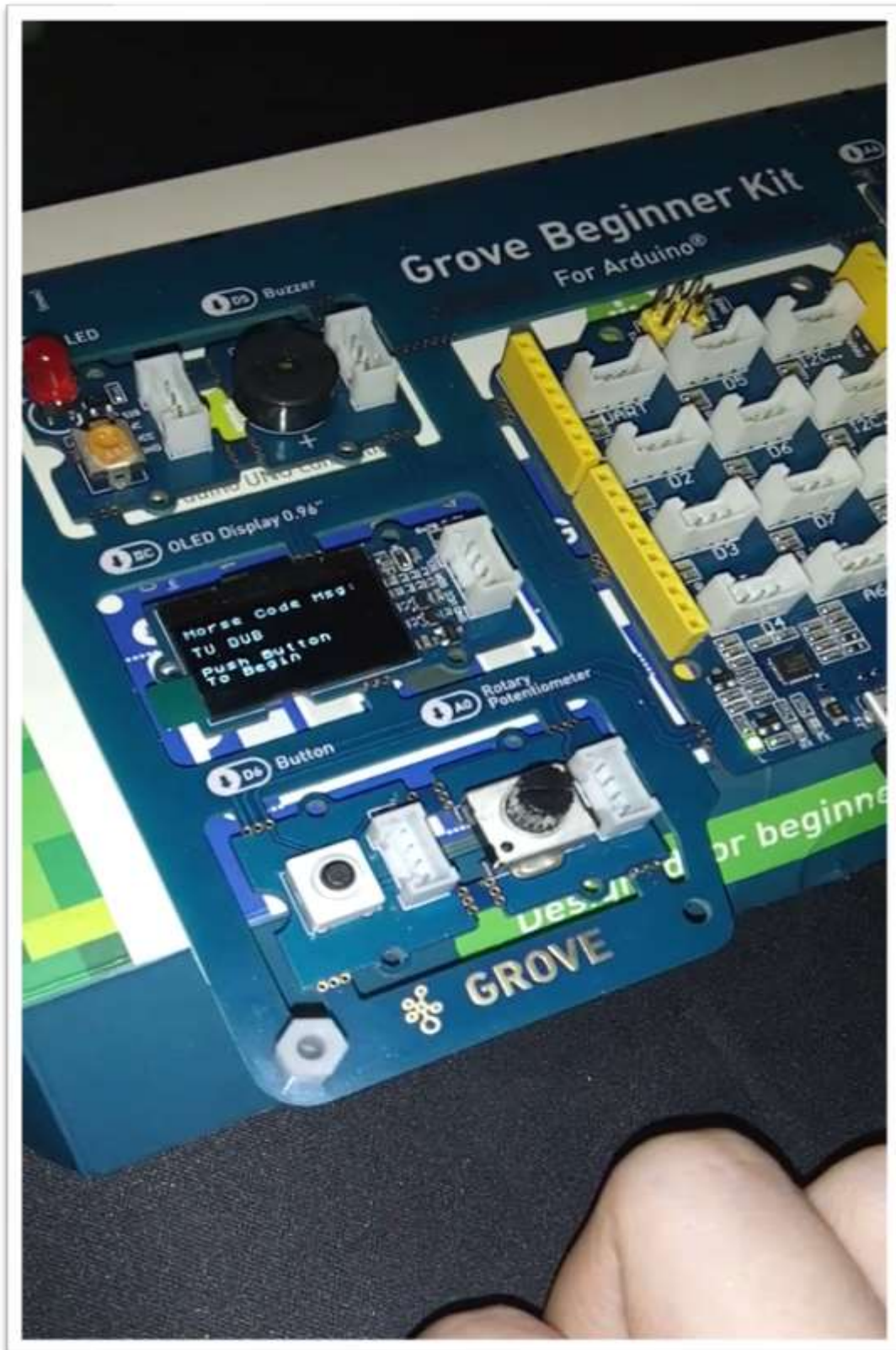


Figure 3.2: After pressing button

The completed message on screen.

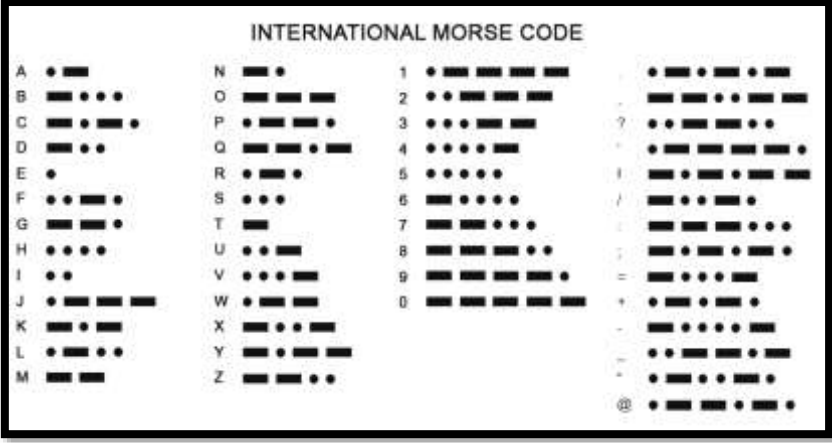


Figure 3.3: Program finish

4.0 References

International Morse Code Chart from:

<https://otasurvivalschool.com/signalling/learn-the-morse-code/>



A	• —	N	— •	1	• — — — —	.	• — — — —
B	— • • •	O	— — —	2	• • — — —	,	— • — — —
C	— • — •	P	• — — •	3	• • • — —	?	• • — — — •
D	— • •	Q	— • — —	4	• • • • —	!	• — — — — •
E	•	R	• — •	5	• • • • •	@	— • — — — •
F	• • — •	S	• • •	6	— • • • •	/	— • • • •
G	— — •	T	—	7	— — • • •	:	— — — • •
H	• • • •	U	• • —	8	— — — • •	;	— • — • —
I	• •	V	• • • —	9	— — — — •	=	• • • • •
J	• — — — —	W	— • — —	0	— — — — —	+	— • — • —
K	— • • —	X	— • • —			-	— • — — —
L	• — • •	Y	— • — —			_	• • — — —
M	— —	Z	— — • •			•	• • — — —

Figure 4.1: Morse Chart

Order of Operation based on Highest Letter Frequency in English language:

https://en.wikipedia.org/wiki/Letter_frequency

U8x8 Graphics Library Documentation:

<https://github.com/olikraus/u8g2/wiki/u8x8reference>

Buzzer Tone Operation Derived From:

<https://www.arduino.cc/en/Tutorial/BuiltInExamples/toneMelody>

Arduino Header Reference:

<https://github.com/esp8266/Arduino/blob/master/cores/esp8266/Arduino.h>