EN.605.715.81.FA25 Project 1 - Arduino Morse Code

Kelly “Scott” Sims

Github Link:
https://github.com/Mooseburger1/johns_hopkins_masters/tree/main/software_development_for_rt_embedded_systems_EN_605_715_81/project_1/morse_code/src

YouTube Link: https://youtu.be/E6V-tuiLdfY

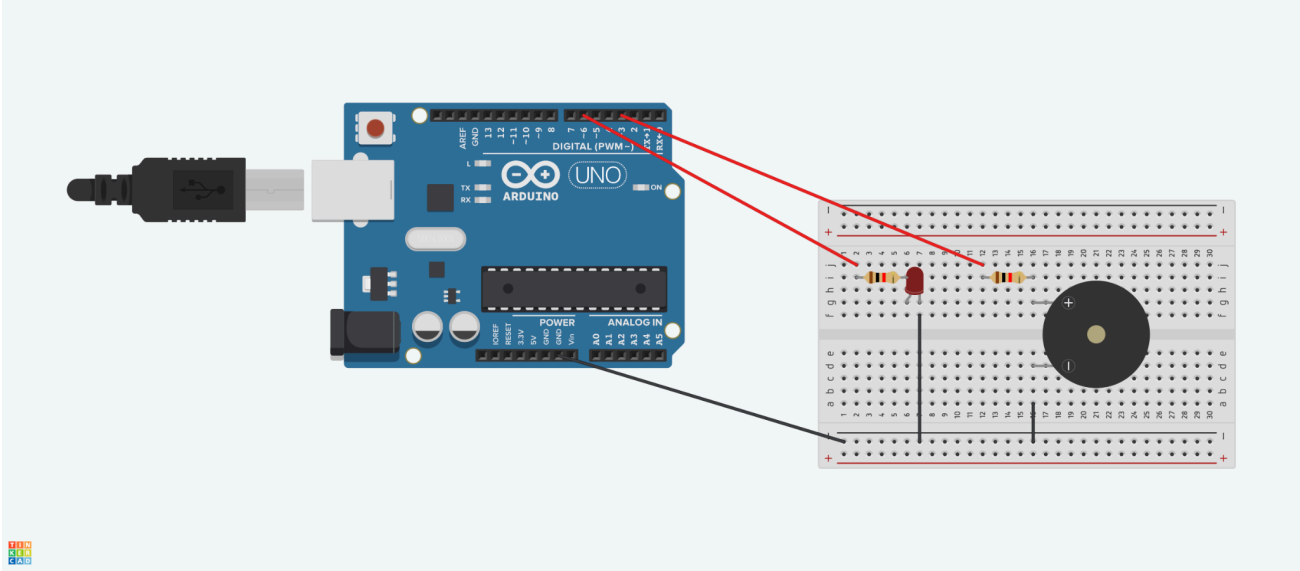
Requirements

For this project, I’m following the exact project requirements, which are as follows:

Develop a C or C++ (or whatever other language you prefer) application which executes on an Arduino and displays a user typed string, such as “Hello World”, as Morse Code on an LED (or several LEDs), or an LCD. Implement all of the Morse Code letters. The word or phrase should typed and followed by enter and need not be an entire sentence with periods, commas, apostrophes, or other punctuations.
Design using a Round Robbin Design where in a loop it waits for a string, displays it in Morse Code, and only exits the loop if a sentinel is entered, such as ctrl-z

Design

The design will support the configuration and activation of many pins without the need to alter any code for every new pin added. If one wishes to add multiple LEDs or even buzzers for audible tones, they need to only wire up their components to the desired pins on the Arduino. Then in main.cpp, add all pins that should be triggered, to a compile-time constant global variable.



LED and Passive Buzzer wired. The design simply allows one to configure pin 6 and 3 as pins to be activated during pulsing. The code will handle the rest with no further modifications needed.

Because all letters and numbers have well defined morse code encodings, no dynamic allocations or runtime constants are required. Instead, a compile-time array will contain all the mappings from a letter to its morse code encodings. A character can then be mapped to its morse code encodings by utilizing its ASCII ordinal value. This is explained in detail below.

For the demonstration, two pins will be configured. One will be wired to a LED to pulse the morse code signal. The other will be wired to an active buzzer. This is to make the morse code pulses audible.

Mapping of Char to Index in Morse Code Aray

Only the character set [A-Za-z0-9]+ will be supported. The first nine indices of the vector will contain the morse code encodings for the values of “0” - “9” respectively. The following 26 indices (10-36) will contain the morse code encodings for the values of “A” - “Z” respectively.

Characters “0” - “9”

All characters in the “0”-”9” range can be converted to their appropriate indices by subtracting 48 from their ASCII ordinal value. For example:

ASCII Char: “0”
ASCII Integer Value: 48
Index Position Mapping: 48 - 48 = 0

ASCII Char: “5”
ASCII Integer Value: 53
Index Position Mapping: 53 - 48 = 5

Characters “A” - “Z”

All characters in the “A” - “Z” will begin at the index position of 10. Obviously this is because “0”-”9” consumes the first nine indices of the morse code array. This leaves 10 - 36 for the capital letter character set.

Capital letters can be converted to their index positions by subtracting 55 from their ASCII Ordinal
value. This is because A-Z are 65 - 90 respectively. And because we want “A” to start at index position 10, we subtract 55. For example

ASCII Char: “A”
ASCII Integer Value: 65
Index Position Mapping: 65 - 55 = 10

ASCII Char: “X”
ASCII Integer Value: 88
Index Position Mapping: 88 - 55 = 33

Characters “a” - “z”

Lower case letters don’t need a direct mapping in the morse code array. Instead they can simply be converted to uppercase letters and utilizing the existing mappings for those character sets. Conversion from lower to upper can be done by subtracting 32 from the ASCII ordinal value of the lower case letter. For example

ASCII Char: “a”
ASCII Integer Value: 97
Lower To Upper Conversion: 97 - 32 = 65 = “A”

Implementation

```
// main.cpp
    #include "pulse.h"

// Main entry point for pulsing a message in morse code
//
// IMPORTANT: Configure which pins you want to be activated in the int PULSE_PINS[] array
//
// Multiple pins can be configured such that they are activated for each pulse in a letter. To
// adjust the timing of ONE_UNIT to be longer or shorter, thus the timing of all other units, the
// value can be updated in morse_code.h
namespace {

using arduino::String;
using morse::ONE_UNITS;
using morse::THREE_UNITS;
using morse::SEVEN_UNITS;
using morse::TERMINATING_INT;
using pulse::PulseWords;


// Configure which pins should be pulsed for morse code
// Array should always end with TERMINATING_INT
constexpr int PULSE_PINS[] = {3, 6, TERMINATING_INT};

void ConfigurePinModes(const int* pins, int mode) {
  int pin_index = 0;
  while (pins[pin_index] != TERMINATING_INT) {
    pinMode(pins[pin_index], mode);
    ++pin_index;
  }
}

} // namespace

void setup()
{
  ConfigurePinModes(PULSE_PINS, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available()) {
    String msg_to_encode(Serial.readStringUntil('\n'));
    msg_to_encode.trim();
    PulseWords(msg_to_encode, PULSE_PINS);
  }
}
```

```cpp
// morse_code.h.

#ifndef MORSE_CODE_H
#define MORSE_CODE_H

// This file contains the mapping of letters to morse code pulses.
//
// IMPORTANT: Only the character set [A-Za-z0-9]+ is supported
//
// Each character is mapped to an index position in the array `int* MORSE_CODES[]` by its ASCII
// value. The values 0-9 are queued first in the array, followed by A-Z. E.g "0" is at index
// position 0 of the array MORSE_CODES[] and "A" is at index 10.
//
// String numbers are converted to their index values by subtracting 48 from their ASCII Ordinal
// value. This is because "0" - "9" are 48 - 57 respectively. E.g. "5" ASCII integer value is 53;
// 53 - 48 = 5;
//
// Capital letters are converted to their index values by subtracting 55 from their ASCII Ordinal
// value. This is because A-Z are 65 - 90 respectively. Because "0" - "9" consume the first 9
// indices in the array, this means A must start at index position 10. E.g. A ASCII integer values
// is 65; 65 - 55 = 10.
//
// Lower case letter are first converted to upper case letters by subtracting 32 from their ASCII
// Ordinal value. E.g. "a" is 97; 97 - 32 = 65 = A. Then the above logic follows.
//
// See ascii_helpers.h for Atoi conversions as discussed here.

namespace morse {

inline constexpr int ONE_UNITS = 300; // milliseconds
inline constexpr int THREE_UNITS = ONE_UNITS * 3;
inline constexpr int SEVEN_UNITS = ONE_UNITS * 7;
inline constexpr int TERMINATING_INT = 999;

inline constexpr int zero[] = {THREE_UNITS, THREE_UNITS, THREE_UNITS, THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int one[] = {ONE_UNITS, THREE_UNITS, THREE_UNITS, THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int two[] = {ONE_UNITS, ONE_UNITS, THREE_UNITS, THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int three[] = {ONE_UNITS, ONE_UNITS, ONE_UNITS, THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int four[] = {ONE_UNITS, ONE_UNITS, ONE_UNITS, ONE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int five[] = {ONE_UNITS, ONE_UNITS, ONE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int six[] = {THREE_UNITS, ONE_UNITS, ONE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int seven[] = {THREE_UNITS, THREE_UNITS, ONE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int eight[] = {THREE_UNITS, THREE_UNITS, THREE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int nine[] = {THREE_UNITS, THREE_UNITS, THREE_UNITS, THREE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int A[] = {ONE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int B[] = {THREE_UNITS, ONE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int C[] = {THREE_UNITS, ONE_UNITS, THREE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int D[] = {THREE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int E[] = {ONE_UNITS, TERMINATING_INT};
inline constexpr int F[] = {ONE_UNITS, ONE_UNITS, THREE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int G[] = {THREE_UNITS, THREE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int H[] = {ONE_UNITS, ONE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int I[] = {ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int J[] = {ONE_UNITS ,THREE_UNITS , THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int K[] = {THREE_UNITS, ONE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int L[] = {ONE_UNITS, THREE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int M[] = {THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int N[] = {THREE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int O[] = {THREE_UNITS, THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int P[] = {ONE_UNITS, THREE_UNITS, THREE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int Q[] = {THREE_UNITS, THREE_UNITS, ONE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int R[] = {ONE_UNITS, THREE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int S[] = {ONE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};
inline constexpr int T[] = {THREE_UNITS, TERMINATING_INT};
inline constexpr int U[] = {ONE_UNITS, ONE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int V[] = {ONE_UNITS, ONE_UNITS, ONE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int W[] = {ONE_UNITS, THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int X[] = {THREE_UNITS, ONE_UNITS, ONE_UNITS, THREE_UNITS, TERMINATING_INT};
```

```
inline constexpr int Y[] = {THREE_UNITS, ONE_UNITS, THREE_UNITS, THREE_UNITS, TERMINATING_INT};
inline constexpr int Z[] = {THREE_UNITS, THREE_UNITS, ONE_UNITS, ONE_UNITS, TERMINATING_INT};


inline constexpr const int* MORSE_CODES[] = {
    zero, one, two, three, four, five, six, seven, eight, nine, 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
};

} // morse

#endif MORSE_CODE_H
```

```
// ascii_helpers.h

#ifndef ASCII_HELPERS_H
#define ASCII_HELPERS_H

// Helper functions to manage the ASCII conversions needed to appropriately map a character
// to its index position in the MORSE_CODES array in morse_code.h

namespace ascii {

// Checks if the provided ASCII value is an integer char. E.g. "0"
bool IsNumber(int int_letter) {
  if ((int_letter >= 48) & (int_letter <= 57)) {
   return true;
  }

  return false;
}

// Checks if the provided ASCII value is an upper case letter. E.g. "A"
bool IsUpperCase(int int_letter) {
 if ((int_letter >= 65) & (int_letter <= 90)) {
   return true;
  }

  return false;
}

// Checks if the provided ASCII value is a lower case letter. E.g. "a"
bool IsLowerCase(int int_letter) {
 if ((int_letter >= 97) & (int_letter <= 122)) {
   return true;
  }

  return false;
}

// Converts the ASCII value of a number to its index position in MORSE_CODES array
int NumberToIndex(int int_letter) {
  return int_letter - 48;
}

// Converts the ASCII value of a capital letter to its index position in MORSE_CODES array
int CapitalLetterToIndex(int int_letter) {
  return int_letter - 55;
}

// Converts a lower letter to a capital by subtracting 32 from its ASCII value.
int ToUpper(int int_letter) {
  return int_letter - 32;
}

// Maps a char to its index position in the MORSE_CODES array
int AsciiToIndex(const char letter) {
  int int_letter(letter);
```

```
  if (IsNumber(int_letter)) {
    return NumberToIndex(int_letter);
  }

  if (IsUpperCase(int_letter)) {
    return CapitalLetterToIndex(int_letter);
  }

  if (IsLowerCase(int_letter)) {
    return CapitalLetterToIndex(ToUpper(int_letter));
  }

  return -1;
}

} // ascii


#endif ASCII_HELPERS_H
```

```
// pulse.h

#ifndef PULSE_H
#define PULSE_H

#include "ascii_helpers.h"
#include "morse_code.h"

namespace pulse {
namespace {

using morse::ONE_UNITS;
using morse::THREE_UNITS;
using morse::SEVEN_UNITS;
using morse::TERMINATING_INT;

} // namespace

// Iterates through an array of pins and writes the value to them. The array is expected to have
// the sentinel TERMINATING_INT to determine when to stop iterating and writing.
void WritePins(const int* pins, int value) {
  int i = 0;
  while (pins[i] != TERMINATING_INT) {
    digitalWrite(pins[i], value);
    i++;
  }
}

// Iterates through an array of pulses for a letter. Ensures each configured pin is activated for
// the appropriate pulse length before deactivating. Respects the morse code rules of ONE_UNIT of
// pause between consecutive pulses for the letter.
void PulseLetters(const int* letter_pulses, const int* pins) {
  int i = 0;
  while (letter_pulses[i] != TERMINATING_INT) {
    int pulse_length = letter_pulses[i];

    Serial.print("Pulse for ");
    Serial.println(pulse_length);
    WritePins(pins, HIGH);
    delay(pulse_length);
    WritePins(pins, LOW);

    // If this is the last pulse for the letter, we don't want to add an extra ONE_UNIT delay
    // to the delay between two letters. Conversely we don't want to add an extra ONE_UNIT delay
    // between words if this is the last letter in a word.
    if (letter_pulses[i + 1] != TERMINATING_INT) {
      delay(ONE_UNITS);
    }
    i++;
  }
  Serial.println("Finished with letter");
```

```
}

// Iterates through an array of characters that make up words in a sentence. Pulse each character in
// a word. Applies the morse code rules of THREE_UNITs of pause between each letter that makes
// up a word. Also applies the rule of SEVEN_UNITS between each word. The start of a new word is
// identified as the first char seen AFTER a ' ' character.
void PulseWords(String words, const int* pins) {
  char word_separator = ' ';
  for (const auto& letter : words) {
    Serial.print("Letter ");
    Serial.println(letter);
    if (word_separator == letter) {
      delay(SEVEN_UNITS);
      continue;
    }

    const int* letter_pulses = morse::MORSE_CODES[ascii::AsciiToIndex(letter)];
    PulseLetters(letter_pulses, pins);
    delay(THREE_UNITS);
  }
}
} // pulse

#endif PULSE_H
```