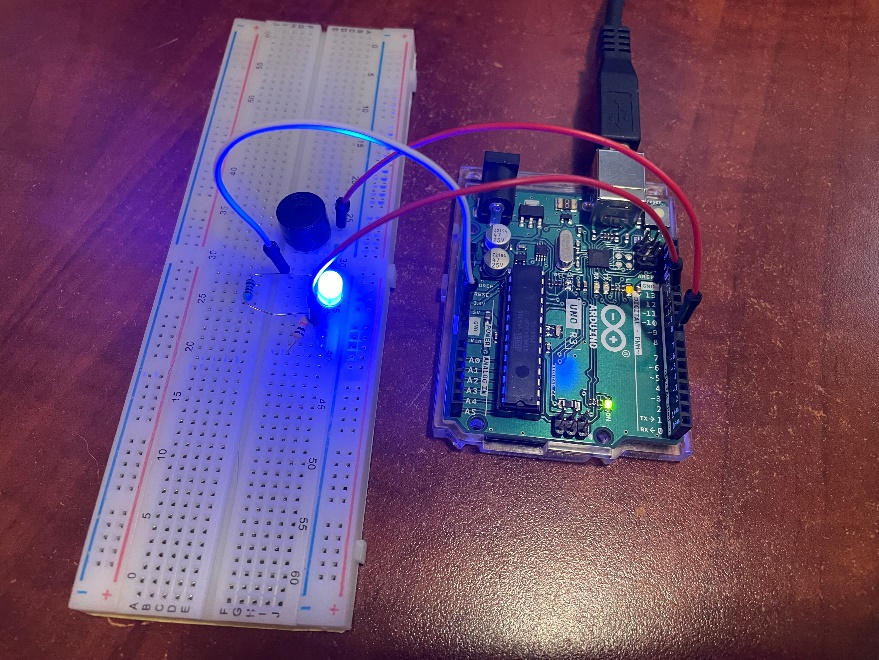
# Description

The Morse LED project is a MCU that transmit Morse Code signals through a LED. The input is given by the User via a serial interface.



# Requirements

The following requirements describes the functionality set forth by the Morse LED project:

* The MCU shall accept ASCII input from the User.
* The MCU shall convert ASCII characters to Morse Code in accordance with the ICU Standard.
* The MCU shall drive a LED to output the Morse Code translation of the ASCII input given by the User.
* The MCU shall implement a Round Robin scheduling algorithm for input and output.

# Design

The following sections describes the hardware, software, and configuration design decisions for the Morse LED project.

## Hardware

The Arduino UNO was used as the MCU for this project. In addition, the following materials were used:

* Jumper Cables
* Breadboard
* 1x100Ohm Resistor
* 1xLED

Optionally, a small piezo speaker may be used in this project.

The circuit schematic is relatively simple. The following diagram describes the project circuitry. Note that a speaker is optional and may be left out the design:

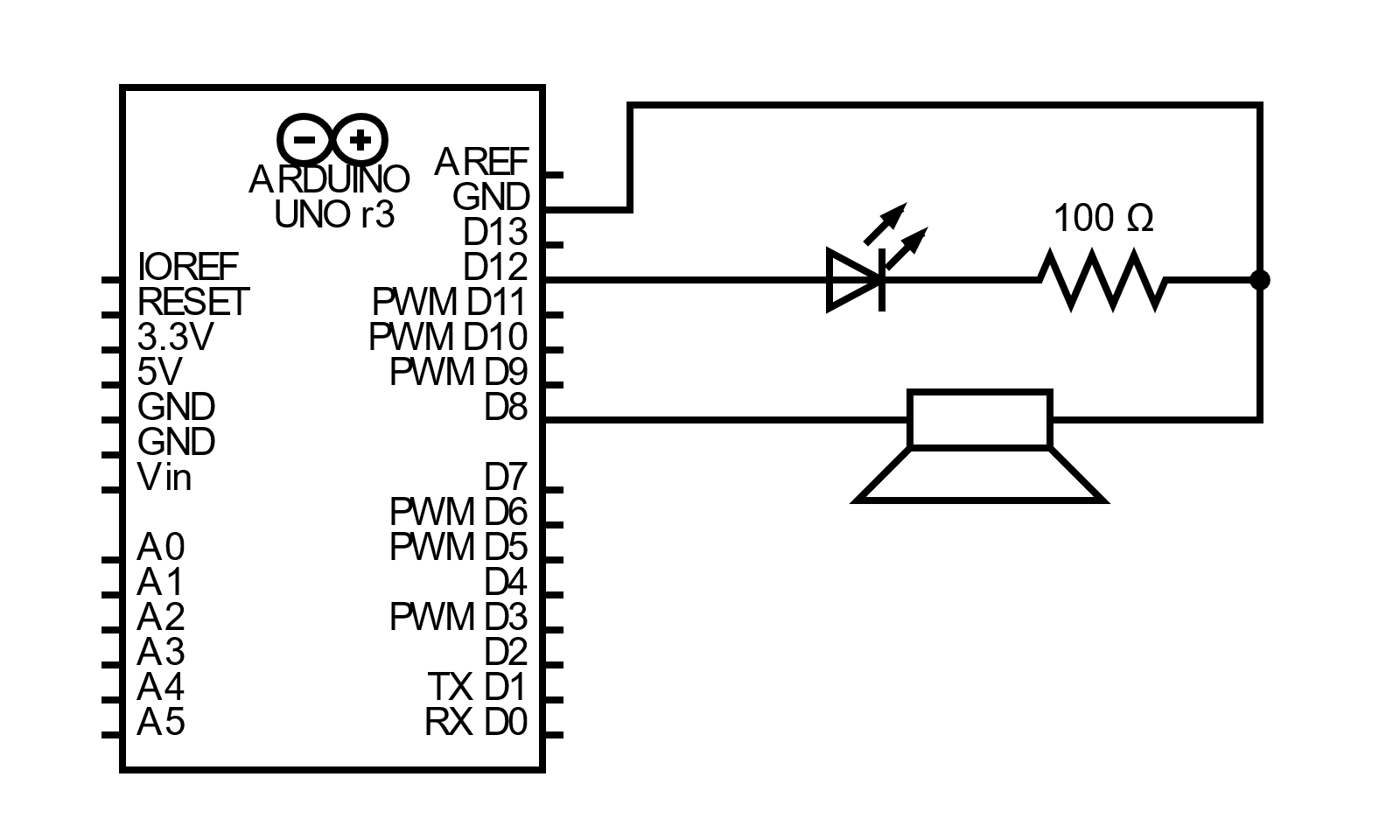


Figure – Schematic

## Software

The software language for the Arduino UNO is C++ using the built in Arduino libraries. There are three sections of the software: the Main Driver, Serial Input, LED Output and Morse Encoder. The following sections describes each section.

### Main Driver

The Main Driver contains the setup and loop code for the UNO.

#### Setup

The setup code includes the initialization of the following variables:

|  |  |  |
| --- | --- | --- |
| Label | Type | Initialization Value |
| message | char array | Zeroed out |
| length | int | 0 |
| halt | bool | false |

The setup code initializes pin LED\_PIN to OUTPUT and set it to low. The Serial class is initialized with baud rate 9600 and the initial enter prompt is written out to serial.

#### Loop

The main loop will check the if the halt variable is raised to true. If the halt is set to true, the loop does nothing (halt), otherwise the following procedure occurs:

1. The LED Output function is called with the message and length variables as input.
2. The Serial Input function is called with the message and length variables as input. Both message and input are modified during this operation. The returned value of this function is assigned to the halt variable.

### Serial Input

The Serial Input software modifies the given message string buffer and length variable pointer. The serial input is read from the terminal and each character is stored in the message string buffer. The length variable is updated accordingly. If the termination character (Ctrl-Z) is entered, the Serial Input software returns a halt flag that is returned to the Main Driver.

### LED Driver

The LED Driver software pulls up or down the LED pin that is defined by configuration. The message string buffer and length variables are passed as input. The software iterates through each character in the message string and transmits the corresponding morse code message.

### Morse Encoder

The Morse Encoder software contains the morse code definitions from the ICU Standard. This software provides helper functions that are used by the Serial Input and LED Driver software to perform MCU operations.

## Configuration

The Morse LED project offers configuration options for both hardware and software settings.

For hardware, an optional piezo speaker may be added according to the schematic described in Section 3.1. To enable functionality the ENABLE\_TONE constant variable must be set to true.

For software, the morse\_p1.ino file contains the following variables that may be configured. Note that the rebuild and load operation must be performed for these settings to take effect.

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Default | Description |
| LED\_PIN | uint8\_t | 12 | LED output pin. |
| UNIT\_TIME | unsigned long | 50 | The Morse Code unit time in milliseconds. |
| MESSAGE\_LIMIT | int | 255 | The max number of bytes (characters) that the User can enter as input. |
| ENABLE\_TONE | bool | FALSE | If set to TRUE, enables speaker output from pin 8. |

# Implementation

The source code of the Morse LED project is as follows:

## morse\_p1.ino

/\*

  Morse Code Transmitter

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#include <string.h>

#include "driver.hpp"

// ----- CONSTANTS ----- //

const uint8\_t LED\_PIN = 12;

const unsigned long UNIT\_TIME = 50;

const int MESSAGE\_LIMIT = 255;

bool ENABLE\_TONE = false;

// ----- GLOABALS ----- //

char message[MESSAGE\_LIMIT];

int length;

bool halt;

/// @brief Initialization

void setup()

{

  pinMode(LED\_PIN, OUTPUT);

  digitalWrite(LED\_PIN, LOW);

  Serial.begin(9600);

  memset(message, 0, MESSAGE\_LIMIT);

  length = 0;

  halt = false;

  Serial.println("MORSE CODE TRANSMITTER");

  Serial.println("Author: Nate Lao (nlao@jh.edu)");

  Serial.print("Enter: ");

}

/// @brief Main Loop

void loop()

{

  if (!halt)

  {

    DRIVER::output(LED\_PIN, message, length, UNIT\_TIME, ENABLE\_TONE);

    halt = DRIVER::input(MESSAGE\_LIMIT, message, &length);

  }

  // halt indefinitely

}

## driver.hpp

/\*

  Morse Code Transmitter

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#ifndef \_\_DRIVER\_HPP\_\_

#define \_\_DRIVER\_HPP\_\_

#include <Arduino.h>

#define DEBUG 0

namespace DRIVER

{

    bool input(const int limit, char \*message, int \*length);

    void output(uint8\_t led\_pin, char \*message, size\_t length, unsigned long unit\_time = 75, bool enable\_tone = false);

}

#endif

## driver\_input.cpp

/\*

  Morse Code Transmitter

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#include "driver.hpp"

#include "morse.hpp"

bool DRIVER::input(const int limit, char \*message, int \*length)

{

    bool halt = false;

    int input\_len = Serial.available();

    while (input\_len > 0)

    {

        char input = (char) Serial.read();

        Serial.print(input);

        // Escape singleton Ctrl-Z

        halt = input == 26;

        // Populate the message buffer

        if (!halt && (MORSE::valid(input)))

        {

            message[\*length] = input;

            \*length = (\*length + 1) % limit;

        }

        input\_len--;

    }

    if (halt)

    {

        Serial.print("\r\n");

        Serial.println(">>> HALT! <<<");

    }

    return halt;

}

## driver\_output.cpp

/\*

  Morse Code Transmitter

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#include "driver.hpp"

#include "morse.hpp"

static void transmit\_morse(char in, uint8\_t pin, const unsigned long unit, bool enable\_tone);

static void debug(const char \*in);

void DRIVER::output(uint8\_t led\_pin, char\* message, size\_t length, unsigned long unit\_time, bool enable\_tone)

{

    size\_t idx = 0;

    while (idx < length)

    {

        transmit\_morse(message[idx], led\_pin, unit\_time, enable\_tone);

        idx++;

    }

    debug("\r\n");

    delay(1000);

}

static void transmit\_morse(char in, uint8\_t led\_pin, const unsigned long unit, bool enable\_tone)

{

    const MORSE::CODE \*code = MORSE::encode(in);

    if (code > 0)

    {

        unsigned int idx = 0;

        while (idx < MORSE::LIMIT)

        {

            switch (code[idx])

            {

            case MORSE::DIT:

                digitalWrite(led\_pin, HIGH);

                if (enable\_tone) tone(8, 440);

                delay(unit);

                digitalWrite(led\_pin, LOW);

                if (enable\_tone) noTone(8);

                debug(".");

                break;

            case MORSE::DAH:

                digitalWrite(led\_pin, HIGH);

                if (enable\_tone) tone(8, 440);

                delay(unit \* 3);

                digitalWrite(led\_pin, LOW);

                if (enable\_tone) noTone(8);

                debug("-");

                break;

            default:

                break;

            }

            idx++;

            debug(" ");

            delay(unit);

        }

        debug("  ");

        delay(unit \* 2);

    }

    else if (in == ' ')

    {

        debug("    ");

        delay(unit \* 4);

    }

}

static void debug(const char \*in)

{

#if DEBUG

    Serial.print(in);

#endif

}

## morse.hpp

/\*

  Morse Code Transmitter

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#ifndef \_\_MORSE\_HPP\_\_

#define \_\_MORSE\_HPP\_\_

namespace MORSE

{

    enum CODE

    {

        NON,

        DIT,

        DAH

    };

    const unsigned int LIMIT = 5;

    /// @brief Defines the Morse Encodings for Numbers

    const CODE NUMER[10][LIMIT] =

    {

        {DAH, DAH, DAH, DAH, DAH}, // 0

        {DIT, DAH, DAH, DAH, DAH}, // 1

        {DIT, DIT, DAH, DAH, DAH}, // 2

        {DIT, DIT, DIT, DAH, DAH}, // 3

        {DIT, DIT, DIT, DIT, DAH}, // 4

        {DIT, DIT, DIT, DIT, DIT}, // 5

        {DAH, DIT, DIT, DIT, DIT}, // 6

        {DAH, DAH, DIT, DIT, DIT}, // 7

        {DAH, DAH, DAH, DIT, DIT}, // 8

        {DAH, DAH, DAH, DAH, DIT}, // 9

    };

    /// @brief Defines the Morse Encodings for Letters

    const CODE ALPHA[26][LIMIT] =

    {

        {DIT, DAH, NON, NON, NON}, // A

        {DAH, DIT, DIT, DIT, NON}, // B

        {DAH, DIT, DAH, DIT, NON}, // C

        {DAH, DIT, DIT, NON, NON}, // D

        {DIT, NON, NON, NON, NON}, // E

        {DIT, DIT, DAH, DIT, NON}, // F

        {DAH, DAH, DIT, NON, NON}, // G

        {DIT, DIT, DIT, DIT, NON}, // H

        {DIT, DIT, NON, NON, NON}, // I

        {DIT, DAH, DAH, DAH, NON}, // J

        {DAH, DIT, DAH, NON, NON}, // K

        {DIT, DAH, DIT, DIT, NON}, // L

        {DIT, DIT, NON, NON, NON}, // M

        {DAH, DIT, NON, NON, NON}, // N

        {DAH, DAH, DAH, NON, NON}, // O

        {DIT, DAH, DAH, DIT, NON}, // P

        {DAH, DAH, DIT, DAH, NON}, // Q

        {DIT, DAH, DIT, NON, NON}, // R

        {DIT, DIT, DIT, NON, NON}, // S

        {NON, DAH, NON, NON, NON}, // T

        {DIT, DIT, DAH, NON, NON}, // U

        {DIT, DIT, DIT, DAH, NON}, // V

        {DIT, DAH, DAH, NON, NON}, // W

        {DAH, DIT, DIT, DAH, NON}, // X

        {DAH, DIT, DAH, DAH, NON}, // Y

        {DAH, DAH, DIT, DIT, NON}, // Z

    };

    bool valid(char in);

    /// @brief Returns the address to the morse code encoding of the given alphanumeric character.

    /// @param in Alphanumeric character to encode.

    /// @return The starting address of the morse code CODE encoding. It is guaranteed that the address segment is 5 bytes long.

    /// @return 0 if the character is invalid

    const CODE\* encode(char in);

}

#endif

## morse.cpp

/\*

  Morse Code Transmitter

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#include "morse.hpp"

bool MORSE::valid(char in)

{

    return (in >= 48 && in <= 57) || (in >= 65 && in <= 90) || (in >= 97 && in <= 122);

}

const MORSE::CODE\* MORSE::encode(char in)

{

    const MORSE::CODE \*code = 0;

    // Numbers

    if (in >= 48 && in <= 57)

    {

        in = in - 48;

        code = &MORSE::NUMER[in][0];

    }

    // Uppercase

    else if (in >= 65 && in <= 90)

    {

        in = in - 65;

        code = &MORSE::ALPHA[in][0];

    }

    // Lowercase

    else if (in >= 97 && in <= 122)

    {

        in = in - 32; // to uppercase

        in = in - 65; // offset

        code = &MORSE::ALPHA[in][0];

    }

    else

    {

        code = 0;

    }

    return code;

}

# Usage

The following instructions describes the steps to use the Morse LED project:

1. Implement the circuit as described in Section 3.1.
2. Using an Arduino IDE, load the Morse LED software to the Arduino UNO.
3. Using a Serial interface program, such as PuTTY. Open a serial port to the UNO with baud rate of 9600.
   1. The following prompt should appear:

Text

Description automatically generated

Figure – Serial Input Prompt

1. Enter a message to the serial window.
   1. The LED (and optionally the speaker) will transmit the Morse Code translation of the given message.
   2. Note that the UNO will “stall” when transmitting the message. Input will still be accepted, but updates to the serial window and the morse code output will only take effect after the current transmission has ceased.
2. To exit the program, enter Ctrl-Z in the serial window. This will suspend the UNO until reset.
   1. The “HALT!” message will appear in the serial window when this occurs:

Text

Description automatically generated

Figure – Serial Halt Prompt

# DEMO

A demonstration of the Morse LED project can be found here:

<https://youtu.be/pEsDTozM0jc>