Ring-Ring Please Pick-up 🐶

Mina Waheeb

EECE 4263: Embedded Systems

4/26/2022

Table of Contents

[**Introduction** 5](#_Toc101920339)

[Background 5](#_Toc101920340)

[Overview 5](#_Toc101920342)

[Hardware 6](#_Toc101920343)

[**Software Design** 8](#_Toc101920344)

[Flowchart 9](#_Toc101920345)

[Deep Dive in Software Design 10](#_Toc101920346)

[Testing Case 10](#_Toc101920347)

[**Summary and Conclusion** 10](#_Toc101920348)

[Works Cited 11](#_Toc101920349)

[**Appendix A - Digging Deeper Source Code** 12](#_Toc101920350)

[**Appendix B – Mems** 25](#_Toc101920351)

# **Introduction**

## Background

## It is assumed that the reader is familiar with the Arduino ide and language, void subroutines/functions, the JSON language, MQTT-Spy, and what a NodeMCU ESP8266 board is, and be familiar with embedded electronics. The written code was compiled and executed using the Arduino ide available on the Arduino website.

## Overview

This project involved using the NodeMCU ESP8266 board as an IoT device capable of exchanging Ring-Ring messages with other Ring-Ring devices. All messages are transmitted wirelessly via MQTT-Spy software using the ECE MQTT broker. “Ring-Ring" utilizes embedded systems design, hardware interfacing, MQTT messaging protocol, MQTT brokers, JSON payloads, and MQTT-Spy.

Refer to *Figure 1* for post construction layout of the Ring-Ring Please Pick-up 🐶embeded system.

A close-up of a circuit board

Description automatically generated with medium confidence

Figure Ring-Ring Please Pick-up 🐶 embeded system breadboard design layout, breadboard design borrowed from the lab procedure provided by Dr. Nordstrom [1]

The layout in *Figure 1* describes the circuit layout of the ring-ring device relative to the breadboard used, on startup the 7-segment display is blank. Using the rotary encoder, the operator alternates between the different ring-ring devices 0 - 15 in total 0x0 – 0xF in hex. The ring-ring device number is displayed in hex on the 7-segment display. The operator presses the encoder pushbutton to send a ring-ring signal to the device whose ID is displayed on the 7-segment display.

On receiving a ring-ring signal, the NodeMCU displays the device number of the sender on the 7-segment display and plays a ringtone on the piezo buzzer. The software design allows for the operator to press the encoder pushbutton, upon receipt, to reply to the sender with a follow-up ring-ring message directly using the original sender’s device ID displayed on the 7-seg display. Otherwise, the 7-segment display continues displaying the device number of the most recent ring-ring device.

## Hardware

The hardware used in the making of the Ring-Ring device are a NodeMCU ESP8266 board, a piezo buzzer, a Keyes KY-040 rotary encoder with a pushbutton shaft, a 74HC595 shift register, a seven-segment display, and seven 220Ω resistors shown in *Figure 2* below.

A close-up of a circuit board

Description automatically generated with medium confidence

*seven 220Ω resistors*

*Keyes KY-040 rotary encoder*

*NodeMCU ESP8266*

*74HC595 shift register*

*7-segment display*

Figure labeled Ring-Ring Please Pick-up 🐶 embeded system breadboard design layout, breadboard design borrowed from the lab procedure provided by Dr. Nordstrom [1]

*Figure 2* displays the labeled breadboard design of the ring-ring device fabricated, and in *Figure 3* below the circuit design is shown.

The schematic design in *Figure 3* belowis the circuit design used in the making of the breadboard construction layout in *Figure 2* above.

Diagram, schematic

Description automatically generated

Figure circuit design used in the making of the breadboard construction layout in Figure 2, circuit design borrowed from the lab procedure provided by Dr. Nordstrom [1]

The circuit in *Figure 3* above describes how the wiring was done for hardware components described earlier, and the wiring implementation using the breadboard in *Figure 2.*

# **Software Design**

**Ring-Ring Message Structure**

The ring-ring device nodes utilized MQTT messaging protocol for communication, and each message must contain a topic for the broker to use in forwarding messages to different nodes. Typically, each message contains a payload containing the data to transmit in byte format as JSON name-value pairs. The MQTT messaging protocol structure is shown below. [1]

**Received messages:**

1. Topic: **ece/node12/topics**

Usage: A request for a list of all the topics your node is registered for Payload: noneResponse: Send an ece/node12/registeredFor message (see below)

2. Topic: **ece/node12/ringring**

Usage: Another node’s ring-ring message sent to node12

Payload: {"srcNode":"node12","dstNode":"nodedd"}

where **ss** and **dd** are source and destination node numbers, respectively

Response: Display sender’s ID number on 7-seg display & play ring-ring tone

3. Topic: **ece/node12/beattime**

Usage: change “heartbeat” time interval

Payload: {"time":"5000"}

Response: Change interval at which “heartbeat” message is sent, if

{"time":"0"} the periodic heartbeat messages are disabled

**Generated messages:**

1. Topic: **ece/node12/registeredFor**

Usage: To publish the list of topics node12 is registered for in response

to a received ece/node12/topics message Payload: {"NodeName":"node12","topics":["topic1\_text","topic2\_text",...,

" topicN\_text"]}

2. Topic: **ece/nodedd/ringring**

Usage: To send a ring-ring message to node dd Payload: {"srcNode":"nodess","dstNode":"nodedd"}

where **ss** and **dd** are source and destination node numbers, respectively

3. Topic: **ece/node12/heartbeat** Usage: To periodically send a “heartbeat” message from node12. Interval of

3 seconds set via a HEARTBEAT\_INTERVAL constant in the program’s .h

file. Setting HEARTBEAT\_INTERVAL to zero disables the sending of

periodic heartbeat messages from node12.

Payload: {"NodeName":"node12","NodeType":"ESP8266"}

## Flowchart

The program controlling Ring-Ring Please Pick-up 🐶 device is best explained by observing the flowchart below in *Figure 4*, a deeper explanation describing the individual steps will follow.

Diagram

Description automatically generated

Block 2

Block 1

C

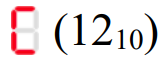
Figure flowchart diagram showing the process of the Ring-Ring Please Pick-up 🐶embeded system, for a complete overview of the written code reference the source code present in Appendix A

## Deep Dive in Software Design

The flowchart in *Figure 4* is a mid-level view of the Ring-Ring Please Pick-up 🐶 device logic process, a deeper dive into the software design of the flowchart will be the main focus of this section.

## Testing Case

The following steps describe an interaction between two Ring-Ring devices (Node05 and Node12) to help illustrate this. Assume each node is initialized and has entered the “ring-ring loop” from the flowchart in *Figure 4* above. [1]

1. Node05’s operator turns the rotary encoder until the 7-segment display shows . The operator then presses the encoder pushbutton, causing Node05 to send a Ring-Ring message to the MQTT broker. The message topic correlates to the destination node (in this case, Node12). This process is highlighted in red and labeled, *Block 1,* in the flowchart *Figure 4* above.
2. Node05 returns to the “ring-ring loop” upon completion

(The 7-segment display continues to show  ).

1. The broker forwards the Ring-Ring message to Node12. (Not shown in *Figure 4)*
2. Node12 receives the Ring-Ring message, decodes it, extracts the sender’s ID, displays the sender’s ID ( in this case) on its 7-segment display, and plays the “ring-ring” tone on the piezo buzzer. This process is highlighted in blue and labeled, *Block 2,* in the flowchart *Figure 4* above.
3. Node12 returns to the “ring-ring loop” upon completion

(The 7-segment display continues to show ).

1. Later, Node12’s operator presses the encoder pushbutton, causing a Ring-Ring message to be sent

back to Node05. Node12’s 7-segment display continues to show ).

This was how the software design of Ring-Ring Please Pick-up 🐶embeded system worked in correlation with the timing loop, the code written to implement this cool project can be referenced at your discretion in *Appendix A.*

# **Summary and Conclusion**

In summary, this project tested the ability of seniors to maintain focus during the final weeks of their college career…of course it was easy 😎. The program written for this project tested the understanding of MQTT protocols, JSON packages, and how MQTT broker was used in handling communication between different nodes.

From this project, it can be concluded that regardless of the amount of character building, even if done twice, carefully reading the spec requirements for a project is crucial. Skipping a spec requirement when working on a project can negate all the hard-working hours spent. Especially if a device fails to meet the spec requirements, therefore, making sure all spec requirements are met is vital for a successful project. I also learned that MQTT opens the door to a lot of possibilities in relation to embedded systems and wireless communication

# Works Cited

|  |  |
| --- | --- |
| [1] | E. 4. E. Systems, "Final Project—“Ring-Ring… Who’s Calling Please?”," Dr. Nordstrom. |
| [2] | T. H. Team, "MQTT Publish, Subscribe & Unsubscribe - MQTT Essentials: Part 4," [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/. |

# **Appendix A - Digging Deeper Source Code**

--ringring.c--

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* This code was created by : Mina Waheeb

\* Date created : 04/08/2022

\* Date last modified : 04/22/2022

\* modifications :

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Multiuser Wireless IoT Communication -“Ring-Ring… Who’s Cal DOOOOM! starts playing”

ETHOS-e gathers temperature and relative humidity information using:

-NodeMCU

-7-segment display

-piezo buzzer

-Keyes KY-040 rotary encoder with pushbutton shaft

-74HC595 shift register

-seven 220Ω resistors

-Mqtt-spy software

"Ring-Ring" is a wireless IoT device capable of exchanging Ring-Ring messages with other Ring-Ring devices. All messages are transmitted wirelessly via MQTT using the ECE MQTT broker."Ring-Ring" utilizes embedded systems design, hardware interfacing, MQTT messaging protocol, MQTT brokers, JSON payloads, and MQTT-Spy.

\*/

/////////////////////////////////////////////////////////////////

//libs

#include <Button2.h>;

#include <Rotary.h>;

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

#include <ArduinoJson.h>

#include "rotary7-sigBuzz.h"

#include "mqtt\_ringNode.h"

////////////////////////////////////////////////////////////////

//functions

void updatePBStatus();

void myfnUpdateDisplay(byte eightBits);

byte myfnNumToBits(int someNumber);

void ringRing();

WiFiClient wfClient;

PubSubClient psClient(wfClient);

void connect\_wifi();

void reconnect();

/////////////////////////////////////////////////////////////////

//globals

Rotary r;

Button2 b;

char json\_msgBuffer[200]; // char buffer to store incoming/outgoing messages

char sbuf[80]; // buffer to store sprintf formatted strings for printing

int hbInt = 3000;

String selectedNode;

String nodeNotation;

int topicIndex;

int senderNodeMCU;

int msg;

byte bits;

/////////////////////////////////////////////////////////////////

//Mqtt "Ring-Ring" function

void sendMQTTMessage(String senderID, String ringring)

{

// create a parse tree buffer and point to the root of the buffer

StaticJsonBuffer<200> jsonBuffer;

JsonObject& root = jsonBuffer.createObject();

root["srcNode"] = senderID;

root["dstNode"] = ringring;

//send message

root.printTo(json\_msgBuffer, sizeof(json\_msgBuffer));

String msgTopic = "ece/" + ringring + "/ringring";

const char \*cmd = msgTopic.c\_str();

psClient.publish(cmd, json\_msgBuffer);

}

//Mqtt "heartbeat" function

void sendHeartbeat(String senderID, String heartbeat)

{

// create a parse tree buffer and point to the root of the buffer

StaticJsonBuffer<200> jsonBuffer;

JsonObject& root = jsonBuffer.createObject();

root["NodeName"] = senderID;

root["NodeType"] = heartbeat;

//send message

root.printTo(json\_msgBuffer, sizeof(json\_msgBuffer));

String msgTopic = "ece/" + senderID + "/heartbeat";

const char \*cmd = msgTopic.c\_str();

psClient.publish(cmd, json\_msgBuffer);

}

//Mqtt "registerdFor" function

void registerdForResponse(String senderID)

{

// create a parse tree buffer and point to the root of the buffer

StaticJsonBuffer<200> jsonBuffer;

JsonObject& root = jsonBuffer.createObject();

root["NodeName"] = senderID;

root["topics"] = "[topics, ringring, beattime]";

//send message

root.printTo(json\_msgBuffer, sizeof(json\_msgBuffer));

String msgTopic = "ece/" + senderID + "/registerdFor";

const char \*cmd = msgTopic.c\_str();

psClient.publish(cmd, json\_msgBuffer);

}

//Mqtt "process-Message" function

void processMQTTMessage(char\* topic, byte\* json\_payload, unsigned int length)

{

//process messages by topic

for(int j = 0; j <= sizeof(arrayRec); j++)

{

sprintf(sbuf,"ece/%s/%s", clientID, arrayRec[j]);

if(strcmp(topic, sbuf) == 0)

{

StaticJsonBuffer<200> jsonBuffer;

JsonObject& root = jsonBuffer.parseObject((char\*)json\_payload);

String regfor = root["topics"];

String senderID = root["srcNode"];

String clientID = root["dstNode"];

msg = root["time"];

if(senderID == NULL && j == 1)

j = -1;

senderID.remove(0,4);

senderNodeMCU = senderID.toInt();

topicIndex = j;

break;

}

}

//see which topic mqtt is communicating to

Serial.println(topicIndex);

switch(topicIndex)

{

case 0:

Serial.print("in topics 0=");

Serial.println(topicIndex);

registerdForResponse(clientID);

break;

case 1:

Serial.print("in ringring 1=");

Serial.println(topicIndex);

r.resetPosition(senderNodeMCU,true);

bits = myfnNumToBits(senderNodeMCU);

myfnUpdateDisplay(bits);

ringRing();

break;

case 2:

Serial.print("in beattime 2=");

Serial.println(topicIndex);

hbInt = msg;

break;

}

}

//register to receive messages

void register\_myself()

{

// register with MQTT broker for topics of interest to this node

Serial.print("Registering for topics...");

for(int i = 0; i <= sizeof(arrayRec); i++)

{

sprintf(sbuf,"ece/%s/%s", clientID, arrayRec[i]);

psClient.subscribe(sbuf);

}

//sprintf(sbuf,"ece/%s/ringring", clientID);

//psClient.subscribe(sbuf);

Serial.println(" done");

}

void setup() {

Serial.begin(9600);

delay(50);

Serial.println("\n\nSimple Counter");

r.begin(ROTARY\_PIN1, ROTARY\_PIN2, CLICKS\_PER\_STEP);

r.setChangedHandler(rotate);

r.setLeftRotationHandler(showDirection);

r.setRightRotationHandler(showDirection);

r.setUpperBound(16);

r.setLowerBound(-1);

b.begin(BUTTON\_PIN, INPUT\_PULLUP);

b.setTapHandler(click);

//b.setLongClickHandler(resetPosition);

currentMillis = millis();

// attach a debouncer to rotary pushbutton and set debounce interval

debouncer.attach(BUTTON\_PIN);

debouncer.interval(DEBOUNCE\_INTERVAL);

// Create a timer to periodically call the updatePBStatus routine

pbTimer.attach\_ms(PB\_UPDATE\_TIME, updatePBStatus);

// initialize counters

pressedCount = 0;

releasedCount = 0;

// initialize I/O pins for 7-sig

pinMode(dataPin, OUTPUT);

pinMode(latchPin, OUTPUT);

pinMode(clockPin, OUTPUT);

connect\_wifi();

psClient.setServer(mqttBroker, mqttPort); // specify MQTT broker's domain name (or IP address) and port number

psClient.setCallback(processMQTTMessage); // Specify callback function to process messages from broker

}

void loop() {

// reconnect to MQTT server if connection lost

if (!psClient.connected())

{

reconnect();

}

//check rotary and button

r.loop();

b.loop();

//if heartbeat is zero, stop pulsing mqtt

if(hbInt != 0)

{

if(millis() - currentMillis >= HEARTBEAT\_INTERVAL)

{

Serial.println("heartbeat");

sendHeartbeat(clientID, "ESP8266");

currentMillis = millis();

}

}

psClient.loop(); // call periodically to keep client alive and well

}

/////////////////////////////////////////////////////////////////

//rotary on change function

void rotate(Rotary& r) {

Serial.println(r.getPosition());

if(r.getPosition() == r.getUpperBound())

{

r.resetPosition(0, true);

}

else if(r.getPosition() == r.getLowerBound())

{

r.resetPosition(15,true);

}

bits = myfnNumToBits(r.getPosition());

myfnUpdateDisplay(bits);

}

//rotary on left or right rotation function

void showDirection(Rotary& r) {

Serial.println(r.directionToString(r.getDirection()));

}

// single click

void click(Button2& btn) {

Serial.println("Click!\n");

if(r.getPosition() <= 9)

{

nodeNotation = "node0";

}

else if(r.getPosition() >= 10)

{

nodeNotation = "node";

}

selectedNode = nodeNotation + (String)r.getPosition();

sendMQTTMessage(clientID, selectedNode);

Serial.println(r.getPosition());

}

/\*uncomment if you want the long press functionality\*/

// long click

//void resetPosition(Button2& btn) {

// r.resetPosition();

// Serial.println("Reset!");

//}

/////////////////////////////////////////////////////////////////

//functions that can be afforded to be out the way

//reconnect to wifi

void reconnect()

{

// Loop until the pub-sub client connects to the MQTT broker

while (!psClient.connected()) {

// attempt to connect to MQTT broker

Serial.print("Connecting to MQTT broker (");

Serial.print(mqttBroker);

Serial.print(") as ");

Serial.print(clientID);

Serial.print("...");

if (psClient.connect(clientID)) {

Serial.println(" connected");

// clientID MUST BE UNIQUE for all connected clients

// can also include username, password if broker requires it

// (e.g. psClient.connect(clientID, username, password)

// once connected, register for topics of interest

register\_myself();

sprintf(sbuf,"MQTT initialization complete\r\nReady!\r\n\r\n");

Serial.print(sbuf);

}

else {

// reconnect failed so print a console message, wait, and try again

Serial.println(" failed.");

#ifdef LIPSCOMB

Serial.println("Trying again in 5 sec. (Is processor whitelisted?)");

#else

Serial.println("Trying again in 5 sec.");

#endif

// wait 5 seconds before retrying

delay(5000);

}

}

}

--mqtt\_ringNode.h—

// This file accompanies the mqtt\_btnNode\_starter\_code.ino v2.1 program

// system defines

// uncomment for Lipscomb broker, comment out for "brokerX"

#define LIPSCOMB

// Network and MQTT broker credentials

#ifdef LIPSCOMB

const char\* mqttBroker = "10.51.97.101"; // ECE mosquitto server (wlan0)

const char\* ssid = "LipscombGuest"; // no PW needed for Lipscomb guest wifi

#else

const char\* mqttBroker = "1.1.1.1"; // address of brokerX

const char\* ssid = "ssid"; // ssid of brokerX's network

const char\* password = "password"; // password for brokerX's network

#endif

int mqttPort = 1883;

// Client ID of this "Ring-Ring" controller

const char\* clientID = "node12"; //node ID

String topic1\_topics = "topics";

String topic2\_ringring = "ringring";

String topic3\_heartbeatTime = "beattime";

String arrayRec [3] = {topic1\_topics,topic2\_ringring,topic3\_heartbeatTime};

////////////////////////////////////////////////////////////////////////////////////

//system functions

void connect\_wifi()

{

// in an attempt to remove the annoying garbled text on

// startup, print a couple of blank lines (with delay)

Serial.println();

delay(100);

Serial.println();

delay(100);

// attempt to connect to the WiFi network

Serial.print("Connecting to ");

Serial.print(ssid);

Serial.print(" network");

delay(10);

#ifdef LIPSCOMB

WiFi.begin(ssid); // Lipscomb WiFi does NOT require a password

#else

WiFi.begin(ssid, password); // For WiFi networks that DO require a password

#endif

// advance a "dot-dot-dot" indicator until connected to WiFi network

while (WiFi.status() != WL\_CONNECTED) {

delay(300);

Serial.print(".");

}

// report to console that WiFi is connected and print IP address

Serial.print(" connected as ");

Serial.println(WiFi.localIP());

}

/\*------------------------------------------------------------------\*/

--rotary7-sigBuzz.h--

////////////////////////////////////////////////////////////////////////////////////

//system defines

//for debounce

#define DEBOUNCE\_INTERVAL 5 // 5mS works well for circuit-mount PBs

#define PB\_UPDATE\_TIME 8 // number of mS between button status checks

//for rotary

#define ROTARY\_PIN1 D2 //right

#define ROTARY\_PIN2 D1 //left

#define BUTTON\_PIN D3 //click

#define CLICKS\_PER\_STEP 4 // this number depends on your rotary encoder

#define HEARTBEAT\_INTERVAL hbInt

//for buzzer

#define NOTE\_B2 123

#define NOTE\_C3 131

#define NOTE\_D3 147

#define NOTE\_E2 82

#define NOTE\_E3 165

#define NOTE\_AS2 117

#define REST 0

////////////////////////////////////////////////////////////////////////////////////

//system functions

#include <Bounce2.h> // to debounce pushbuttons

#include <Ticker.h> // to make Bounce2 work

// create a Ticker object to periodically check the

// the state of a debounced pushbutton

Ticker pbTimer;

// create a Bounce object to debounce a pushbutton switch

Bounce debouncer = Bounce();

// global millisecond tick counter

unsigned long currentMillis = millis();

// event counters

int pressedCount, releasedCount;

/\*----------------------------------------------------------------------\*/

void updatePBStatus() {

// this function is called every PB\_UPDATE\_TIME milliseconds

// (currently 8) to update the state of the pushbutton

// debounder (adjust the time according to PB characteristics)

debouncer.update();

if(debouncer.fell()) {

printf("Pressed [%d]\r\n", ++pressedCount);

}

else if(debouncer.rose()) {

printf("Released [%d]\r\n\r\n", ++releasedCount);

}

}

/\*######################################################################\*/

const int dataPin = D8; // blue wire to 74HC595 pin 14

const int latchPin = D7; // green to 74HC595 pin 12

const int clockPin = D6; // yellow to 74HC595 pin 11

//const char common = 'a'; // common anode

const char common = 'c'; // common cathode

/\*----------------------------------------------------------------------\*/

//7-seg display function to update display

void myfnUpdateDisplay(byte eightBits) {

if (common == 'a') { // using a common anonde display?

eightBits = eightBits ^ B11111111; // then flip all bits using XOR

}

digitalWrite(latchPin, LOW); // prepare shift register for data

shiftOut(dataPin, clockPin, LSBFIRST, eightBits); // send data

digitalWrite(latchPin, HIGH); // update display

}

//7-seg display switch function

byte myfnNumToBits(int someNumber) {

switch (someNumber) {

case 0:

return B11111100;

break;

case 1:

return B01100000;

break;

case 2:

return B11011010;

break;

case 3:

return B11110010;

break;

case 4:

return B01100110;

break;

case 5:

return B10110110;

break;

case 6:

return B10111110;

break;

case 7:

return B11100000;

break;

case 8:

return B11111110;

break;

case 9:

return B11110110;

break;

case 10:

return B11101110; // Hexidecimal A

break;

case 11:

return B00111110; // Hexidecimal B

break;

case 12:

return B10011100; // Hexidecimal C or use for Centigrade

break;

case 13:

return B01111010; // Hexidecimal D

break;

case 14:

return B10011110; // Hexidecimal E

break;

case 15:

return B10001110; // Hexidecimal F or use for Fahrenheit

break;

default:

return B10010010; // Error condition, displays three vertical bars

break;

}

}

/\*######################################################################\*/

// change this to make the song slower or faster

int tempo = 225;

// change this to whichever pin you want to use

int buzzer = D5;

const int melody[] PROGMEM = {

// At Doom's Gate (E1M1)

// Score available at https://musescore.com/pieridot/doom

NOTE\_E2, 8, NOTE\_E2, 8, NOTE\_E3, 8, NOTE\_E2, 8, NOTE\_E2, 8, NOTE\_D3, 8, NOTE\_E2, 8, NOTE\_E2, 8, //1

NOTE\_C3, 8, NOTE\_E2, 8, NOTE\_E2, 8, NOTE\_AS2, 8, NOTE\_E2, 8, NOTE\_E2, 8, NOTE\_B2, 8, NOTE\_C3, 8,

NOTE\_E2, 8, NOTE\_E2, 8, NOTE\_E3, 8, NOTE\_E2, 8, NOTE\_E2, 8, NOTE\_D3, 8, NOTE\_E2, 8, NOTE\_E2, 8,

NOTE\_C3, 8, NOTE\_E2, 8, NOTE\_E2, 8,

};

// sizeof gives the number of bytes, each int value is composed of two bytes (16 bits)

// there are two values per note (pitch and duration), so for each note there are four bytes

int notes = sizeof(melody) / sizeof(melody[0]) / 2;

// this calculates the duration of a whole note in ms

int wholenote = (60000 \* 4) / tempo;

int divider = 0, noteDuration = 0;

/\*----------------------------------------------------------------------\*/

//"Ring-Ring" function that plays melody on buzzer

void ringRing()

{

// iterate over the notes of the melody.

// Remember, the array is twice the number of notes (notes + durations)

for (int thisNote = 0; thisNote < notes \* 2; thisNote = thisNote + 2) {

// calculates the duration of each note

divider = pgm\_read\_word\_near(melody+thisNote + 1);

if (divider > 0) {

// regular note, just proceed

noteDuration = (wholenote) / divider;

} else if (divider < 0) {

// dotted notes are represented with negative durations!!

noteDuration = (wholenote) / abs(divider);

noteDuration \*= 1.5; // increases the duration in half for dotted notes

}

// we only play the note for 90% of the duration, leaving 10% as a pause

tone(buzzer, pgm\_read\_word\_near(melody+thisNote), noteDuration \* 0.9);

// Wait for the specief duration before playing the next note.

delay(noteDuration);

// stop the waveform generation before the next note.

noTone(buzzer);

}

}

# **Appendix B – Mems**

# 

# Graphical user interface, text Description automatically generated

# Challenge **🤔**

# Text Description automatically generated

# Had to at least mention Elon

# Graphical user interface, application Description automatically generated