

Final Project

“Ring-Ring... Who’s Calling Please?”

Multiuser Wireless IoT Communication

Overview

In this project you’ll create a wireless IoT device capable of exchanging Ring-Ring messages with other Ring-Ring devices. All messages will be transmitted wirelessly via MQTT using the ECE MQTT broker. This project requires an understanding of embedded systems design, hardware interfacing, the MQTT messaging protocol, MQTT brokers, JSON payloads, and MQTT-Spy.

Ring-Ring Behavior

The Ring-Ring application’s high-level behavior is shown in Figure 1 below.

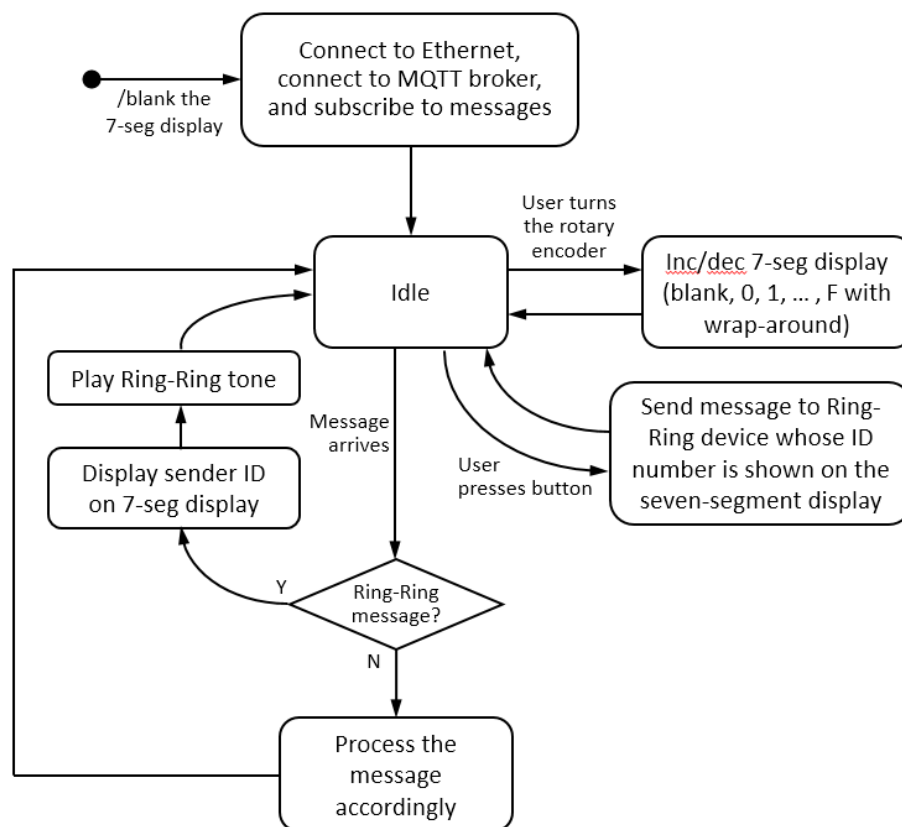







Figure 1. The Ring-Ring application logic.

The following steps describe an interaction between two Ring-Ring devices (Node05 and Node12). Assume each node is initialized and has entered the Idle state *for the first time* (see Figure 1. above).

- a. Node05's operator turns the rotary encoder until the 7-segment display shows  (12₁₀) and 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).
- b. Node05 returns to the Idle state (the 7-segment display continues to show .
- c. The broker forwards the Ring-Ring message to Node12.
- d. 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.
- e. Node12 returns to the Idle state (the 7-segment display continues to show .
- f. 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 .

You have been assigned an ID number which will form part of your MQTT `clientID` and `nodeName` identifiers. For example, if your ID number is 4, your `clientID` will be `ece04` and your `nodeName` will be `node04`.

You must be able to communicate with clients 0 through 15, displaying their ID numbers as single hex digits (0–F) on your device's 7-segment display as explained in the interaction example above. Note: To make the digits and numbers unique, you should strongly consider using lowercase for "b" and "d" as the upper case will be identical to "8" and "0".

Finally, *all* Ring-Ring devices must use an MQTT `nodeType` of ESP8266.

Ring-Ring Message Structure

To ensure full connectivity and proper communication between all ring-ring devices, the following message formats have been defined. You must make sure your device sends and receives messages using these formats.

Received messages

1. Topic: **ece/nodeNN/topics** (where nodeNN is your nodeName)
 Usage: A request for a list of all the topics your node is registered for.
 Payload: none
 Response: Send an **ece/nodeNN/registeredFor** message (see below).
2. Topic: **ece/nodeNN/ringring**
 Usage: Another node's ring-ring message sent to your node
 Payload: {"srcNode":"nodeSS","dstNode":"nodeDD"} where SS and DD are source and destination node numbers, respectively.
 Response: Display the sender's ID number on your 7-segment display and play the ring-ring tone.

Generated messages

1. Topic: **ece/nodeNN/registeredFor**
 Usage: To publish the list of topics your node is registered for in response to a received ece/nodeNN/topics message
 Payload: {"nodeName":"nodeNN","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/nodenn/heartbeat**
 Usage: To periodically send a "heartbeat" message from this node. The interval should be set via a HEARTBEAT_INTERVAL constant in your program's .h file. Setting HEARTBEAT_INTERVAL to zero must disable the sending of periodic heartbeat messages from your node.
 Payload: {"nodeName":"nodenn","NodeType":"ESP8266"}
 where nn is your node's ID number

Ring-Ring Hardware and Software

The hardware for this lab is:

- a NodeMCU ESP-12E board
- a piezo buzzer
- a Keyes KY-040 rotary encoder with pushbutton shaft
- a 74HC595 shift register
- a seven-segment display
- seven 220Ω resistors

A suggested layout is shown in Figure 2 below, and a schematic is included in Appendix A. You are under no obligation to use either one in your design—these are only intended to get you started quickly.

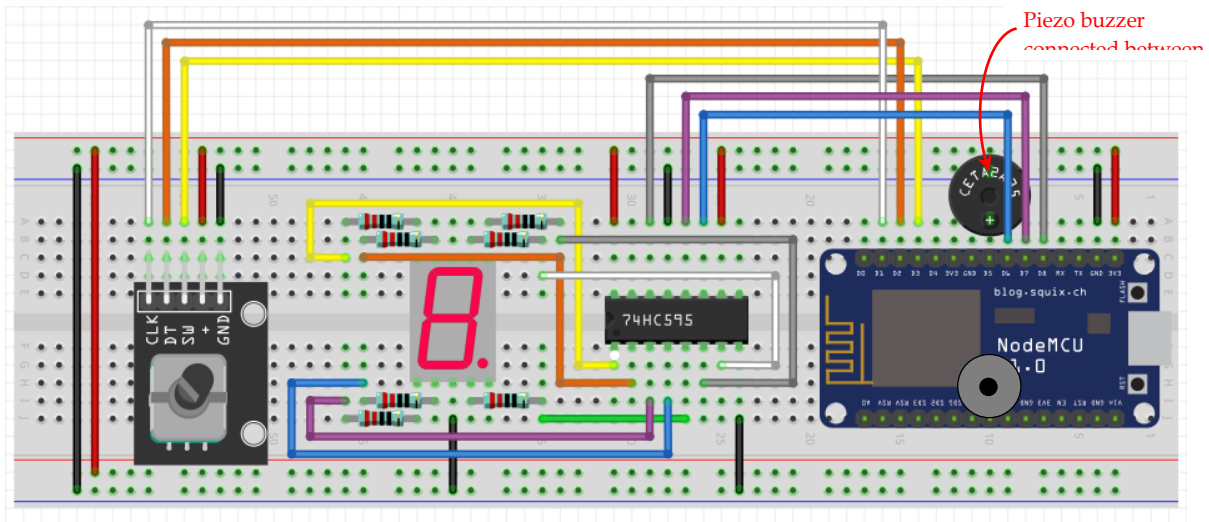


Figure 2. Suggested *Ring-Ring* breadboard layout.

Use the Bounce2 Arduino library to debounce the rotary encoder's pushbutton switch. Recall from an earlier lab that the Bounce() object must be periodically sampled to ensure you capture the current pushbutton state. A Ticker object (another Arduino library) is ideal for automatically sampling Bounce() objects, but this can also be done with careful `loop()` code. There is an Arduino library called "Rotary" that can manage the rotary encoder. The built-in "tone" command can be used to generate tones on the buzzer.

Also, you should use the Arduino PubSubClient library to create and manage your MQTT client object, and the ArduinoJson library to encode and decode JSON formatted payloads. And don't forget the ESP8266WiFi library for managing your WiFi connection to the MQTT broker. The broker is located at IP address **10.51.97.101** and can be accessed using the wireless SSID "LipscombGuest" (no password needed).

Project Report

You must submit a project report that includes the following sections:

- Abstract
- Introduction
- Methods
 - Include some kind of Project Overview, including a complete list of its features and hardware diagrams
 - Include a Theory of Operation, including (but not limited to) a description of the MQTT messaging protocol (i.e. a full spec of each MQTT command and payload)
 - Software Design, including software design artifacts (StateCharts, flow charts, state machines, code snippets, etc.) **NB: Figure 1 above would in no way be appropriate here--too simple!**
- Results and Analysis
 - A summary of project results, including a test matrix w/ expected and actual results (MQTT Spy is your friend here, along with your classmates).

- Should have some level of subsystem testing (may be simple in some cases) and integrated testing.
- Discussion
 - Include a narrative on lessons learned, and any recommendations you have for future enhancements to the project.
- Conclusion
- References
- Appendix (if needed)

NOTE: If your Ring-Ring project uses the exact breadboard layout and schematic that are in this lab, you may include those figures in your final report (properly cited, of course!) If you do a different hardware design, include both a schematic and breadboard layout that match your project.

Utilizes figures/tables/equations and cross-references as appropriate.

Your code will be checked in through GitHub classroom, so you do NOT need to include it in the appendixes.

Turn-in and Project Demonstration

You must **demonstrate your working project on Friday, April 21, 2023** (the last day of lab), and you must **submit your Project Report on or before Tuesday, April 25, 2023** (the last lecture day). **Upload both your project code is uploaded to GitHub Classroom and your project report is uploaded on Canvas by April 25th.** Ensure your report and code comments include a full description of all MQTT messages that your Ring-Ring device can send and receive.

Appendix A

Ring-Ring Schematic

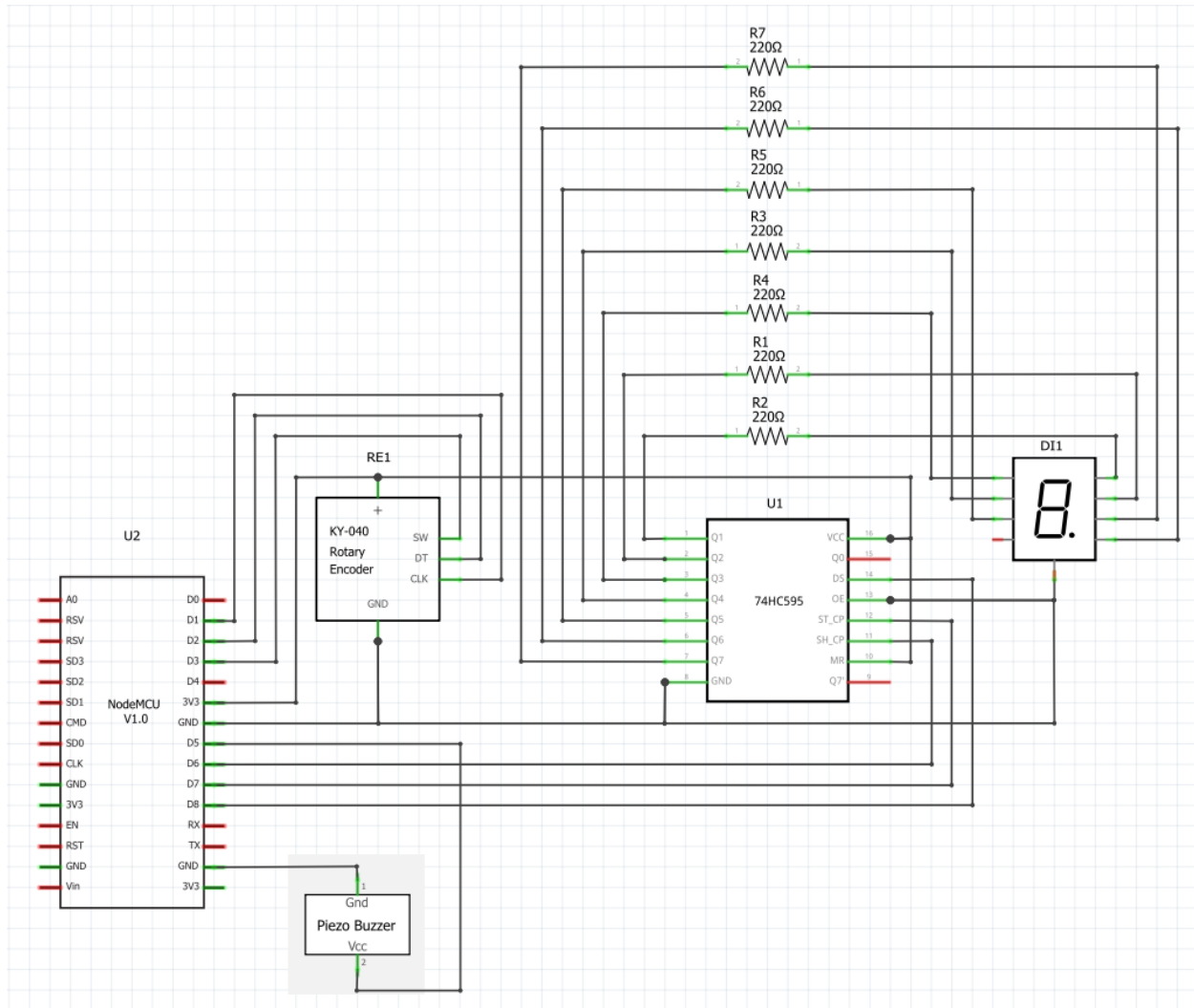


Figure A1. Ring-Ring Schematic