ECE 445L Lab 4

IEE802.11 Wi-Fi communication, TCP, Websockets, MQTT Publish-Subscribe, IoT

This laboratory assignment accompanies the book, [*Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154*](https://www.amazon.com/Embedded-Systems-Real-Time-Interfacing-Microcontrollers/dp/1463590156), by Jonathan W. Valvano, copyright © 2021.

# Table of Contents

[Table of Contents 1](#_Toc1854702796)

[Team Size 1](#_Toc1074048456)

[Goals 2](#_Toc2138146806)

[Review 2](#_Toc1654092334)

[Starter Files 2](#_Toc1969758477)

[Required Hardware 2](#_Toc434252607)

[Lab Overview 3](#_Toc300270852)

[Preparation 6](#_Toc181677851)

[Hardware Setup 6](#_Toc1212487510)

[Pinout Connection 7](#_Toc234231442)

[Powering the ESP8266 9](#_Toc1084265830)

[Software Review 9](#_Toc504985091)

[LaunchPad Starter Project 10](#_Toc1208115265)

[ESP8266 Starter Project 10](#_Toc1748540726)

[MQTT Web Application Starter Project 11](#_Toc1947670270)

[Pub-Sub API 11](#_Toc1106109956)

[Procedure 11](#_Toc810143198)

[ESP8266 Procedure 12](#_Toc1801327663)

[Web App Procedure 14](#_Toc850774622)

[LaunchPad Procedure 18](#_Toc359715594)

[Merge Lab 3 Code 20](#_Toc1335867022)

[Deliverable 1 21](#_Toc768239504)

[Deliverable 2 21](#_Toc1867884925)

[Deliverable 3 21](#_Toc396254412)

[Lab Checkout 22](#_Toc431245364)

[Lab Report 22](#_Toc1686375833)

[Deliverables 22](#_Toc1326950310)

[Analysis and Discussion Questions 23](#_Toc599612819)

[Extra Credit 23](#_Toc1874604912)

[Hints 23](#_Toc2097089211)

[Browser Debugging 23](#_Toc388886915)

[Internet Traffic Analysis & Debug 24](#_Toc1253141838)

[Arduino Setup 25](#_Toc331967380)

# Team Size

The team size for this lab is **2.**

# Goals

* Implement a “smart object” that connects to a web application using a MQTT broker.
* Use the ESP8266 to interface between the internet and the TM4C123.
* Remotely control the clock developed in Lab 3 via the MQTT publish/subscribe interface.

# Review

* Valvano Section 11.4 on IoT
* MQTT architecture (see Lab Overview)

# Starter Files

* Starter project:
  + Lab 5 template provided on the Github Classroom repo.

# Required Hardware

|  |  |  |  |
| --- | --- | --- | --- |
| Parts | Datasheet | Price | Source (**price source)** |
| EK-TM4C123GXL | [EK-TM4C123GXL datasheet](https://github.com/ECE445L/ECE445L-Lab5/blob/main/resources/TM4C_Datasheet.pdf) | $16.99 | **TI** |
| Sitronix ST7735R Color LCD | [ST7735R Driver datasheet](https://cdn-shop.adafruit.com/datasheets/ST7735R_V0.2.pdf) | $19.95 | **Adafruit** |
| 8Ω or 32Ω speaker | N/A | N/A | EER Checkout Desk |
| 10kΩ resistor | N/A | N/A | EER Checkout Desk |
| 1uF Tantalum Capacitor | N/A | N/A | EER Checkout Desk |
| Switches | N/A | N/A | EER Checkout Desk |
| IRLD120 MOSFET  or IRLD024 MOSFET | [IRLD120 datasheet](https://www.vishay.com/docs/91310/sihld120.pdf)  [IRLD024 datasheet](https://www.vishay.com/docs/91308/sihld24.pdf) | $1.88 | EER Checkout Desk  Or Mouser, **Digikey** |
| 1N914 Diode | [1N914 datasheet](https://www.vishay.com/docs/85622/1n914.pdf) | $0.10 | EER Checkout Desk,  Or **Mouser** |
| ESP8266 | [ESP8266 Datasheet](https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf) | 3.50 | Get it from the TA’s |
| ESP8266 Programmer |  | 10.99 | EER Checkout Desk |
| LM2937-3.3 Voltage Regulator **or** any 3.3V regulator | [LM3927 Datasheet](https://www.ti.com/lit/ds/symlink/lm2937.pdf?ts=1695011187491) | 1.68 | EER Checkout Desk |

# Lab Overview

In this lab your team will create a “smart object” that connects to a MQTT Broker. This Broker will allow communication to a Web Application as shown below in Figure 4.1.

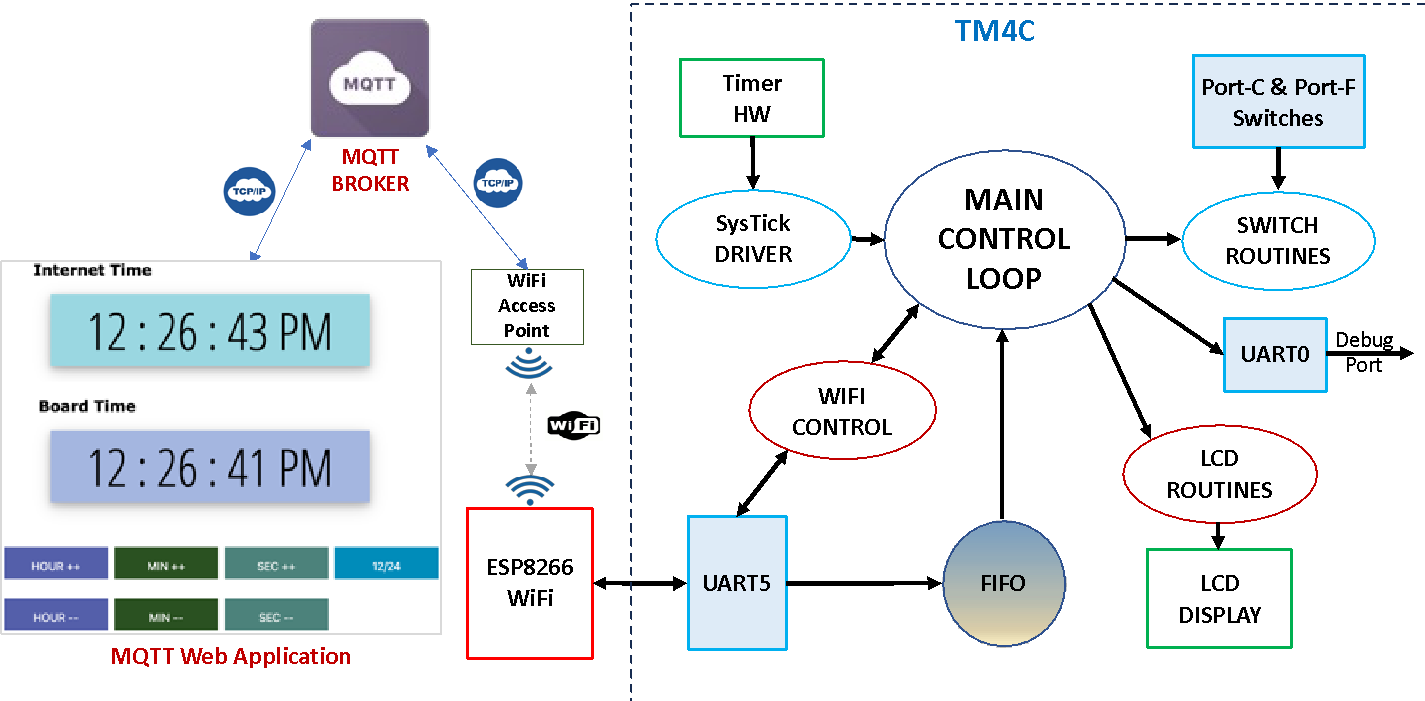


Figure 4.1: Lab 4 Data Flow Diagram

Your system will include:

* A PC or Phone running a MQTT Web App that you will develop using starter code we provide.
* A TM4C123 LaunchPad running your Lab 3 software plus a UART peripheral that talks to the ESP8266 Wi-Fi module.
* An ESP8266 Wi-Fi module (Fig. 4.2) that is configured with starter netcode built using Arduino.
* Existing hardware reused from Lab 3, including the display and any switches.

Your system will be able to:

* Have the phone or laptop running the MQTT Web App, displaying any changes in the TM4C’s state.
* Have the TM4C and its display respond to updates from the MQTT web application.
* Display the time including hour, minute and second as well as 7 control buttons: Hour++, Hour--, Minute++, Minute--, Second++, Second--, Toggle 12/24-hour display.
* A nice to have would be another display showing internet time as shown above in Figure 4.1. You can use this to determine the drift of the crystal on the TM4C.
* Measure an external sensor using the ADC (extra credit). Any external sensor and any interface method is okay (ADC, I2C, or SPI).

You are free to define the “look and feel” of your system if you meet the specifications listed above.

**NOTE: You will need access to a wireless access point, configured with or without security to use the MQTT broker running in the EER. You may also use any online cloud brokers.**

**Wi-Fi** is the name given to devices that communicate wirelessly employing the IEEE 802.11 standard. They typically operate in the 2.4 GHz Industrial-Scientific-Medical (ISM) unlicensed band that is shared with ZigBee (IEEE 802.15.4) and Bluetooth (IEEE 802.15.1) communications. The IEEE 802.11 standard describes how information is represented via radio frequency signals, i.e., the **physical** or **PHY** layer, and how communications are formatted and controlled, i.e., the **media access control** or **MAC** layer. Communication requirements beyond the PHY and MAC, e.g., flow control, error recovery, and routing, are handled at higher levels of the internet stack.

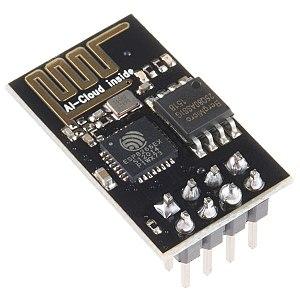


Figure 4.2: The ESP8266 WiFi module.

MQTT ([which stands for nothing](https://www.oasis-open.org/committees/download.php/49028/OASIS_MQTT_TC_minutes_25042013.pdf)) is a message passing mechanism that uses a centralized broker (server) to receive published data and send the data to any number of subscribers as shown below in Figure 4.3.

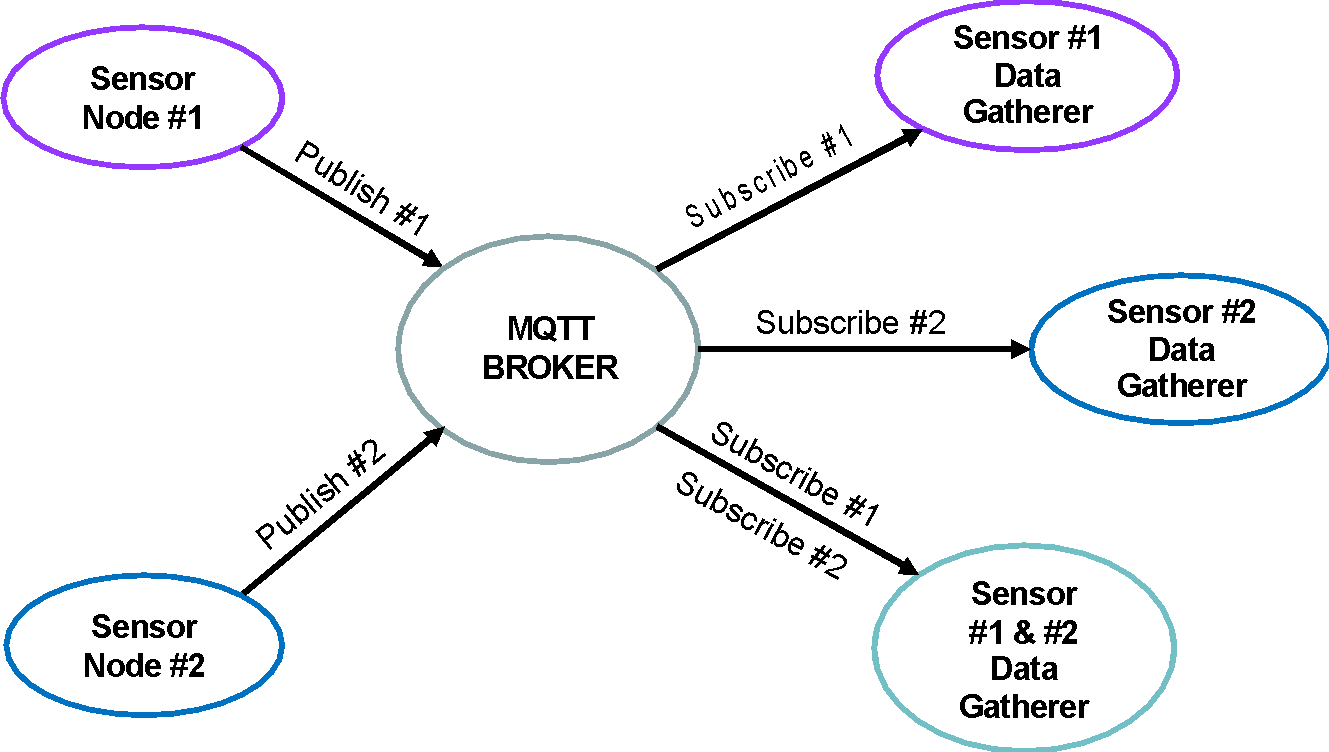


Figure 4.3 MQTT Publish-Subscribe Model

**Key points about MQTT:**

* **All clients** can publish (broadcast) and subscribe (receive).
* Clients **do not have addresses** like in email systems; messages are not sent directly to clients.
* Clients become **subscribers** to “topic(s)” to get messages published to the broker about the “topic”
* Messages are **published to a broker about a particular “topic”**. In this Lab all topics MUST begin with your EID. A topic example is *rg7677/b2w/hour* where:
  + *rg7677* is the student EID,
  + *b2w* is a subtopic indicating that the message is from the LaunchPad board to the web application, and
  + *hour* is a sub-subtopic that is current hour on the board
* The job of an MQTT broker is to **collect and sort messages** based on the “topic”, so that the broker can **distribute messages to subscribers of the “topic”**.
* There is **no direct connection** between a publisher and subscriber.
* MQTT brokers do NOT normally store messages. All messages are sent via TCP which guarantees delivery (vs UDP which does not).
* The publisher dictates the data format. The subscriber must know the format to decipher the data.
* You will be given suggested topics and data formats. You are free to generate your own.
* The only restriction is that all topics must begin with your EID, e.g.,  *rg7677/b2w/hour*
* You have several options for MQTT brokers:
  + McDermott has set up a broker in the EER that you can use, tied to the following config:
    - IP address: 10.159.177.133
    - Port: 9001
    - SSID: Utexas, Utexas-IoT
    - Password: will need one to connect to these Wi-Fi SSIDs
  + You can set up your own broker on a laptop or a server; an option is [MOSQUITTO](https://mosquitto.org/) It is considered one of the best open-source MQTT brokers.
  + An online hosted MQTT broker: EMQX is a free public broker with no sign up that you can use instantly.
    - IP address/domain name: broker.emqx.io
    - Port: 8083 (for websockets, 1883 for TCP port)
    - **The starter code has been set up with to talk to this broker so you can get running instantly.**

Here are some useful MQTT tutorials and projects:

* <http://www.steves-internet-guide.com/mqtt-works/>
* <http://www.steves-internet-guide.com/download/mqtt-study-guide/>
* <http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/>
* <http://www.steves-internet-guide.com/into-mqtt-python-client/>
* <http://www.steves-internet-guide.com/mqtt-websockets/>
* <http://www.steves-internet-guide.com/using-javascript-mqtt-client-websockets/>
* <http://www.steves-internet-guide.com/mosquitto-broker/>
* <http://www.steves-internet-guide.com/install-mosquitto-broker/>
* <http://www.steves-internet-guide.com/mqtt-broker-testing/>
* <http://www.steves-internet-guide.com/monitoring-mqtt-brokers/>
* [MQTT on Arduino Project Hub](https://projecthub.arduino.cc/search?value=MQTT)

# Preparation

Preparation is split into three tasks:

* Hardware setup
* Software review
* Designing the Pub-sub API for the final communication scheme.

FALL 2023: TAs may start the lab prep by walking through the starter code provided and how to bring up each system (TM4C, ESP8266, Web App).

## Hardware Setup

Collect all the hardware needed to perform this lab:

* LaunchPad,
* ST7735,
* ESP8266-ESP01 (you can keep the ESP8266-ESP01 for Lab 10 and your project),
* 8-pin to 8-pin adapter so the ESP8266 can be plugged into the breadboard (please give this adapter back after Lab 10), and
* ESP programmer.

You will need to checkout an ESP8266 programmer from the checkout counter if you intend to change the functionality of the ESP8266 for the Lab or for your project. **NOTE: the orientation of the ESP8266 displayed on the programmer. You will burn up the ESP8266 if you install it in the other orientation.**

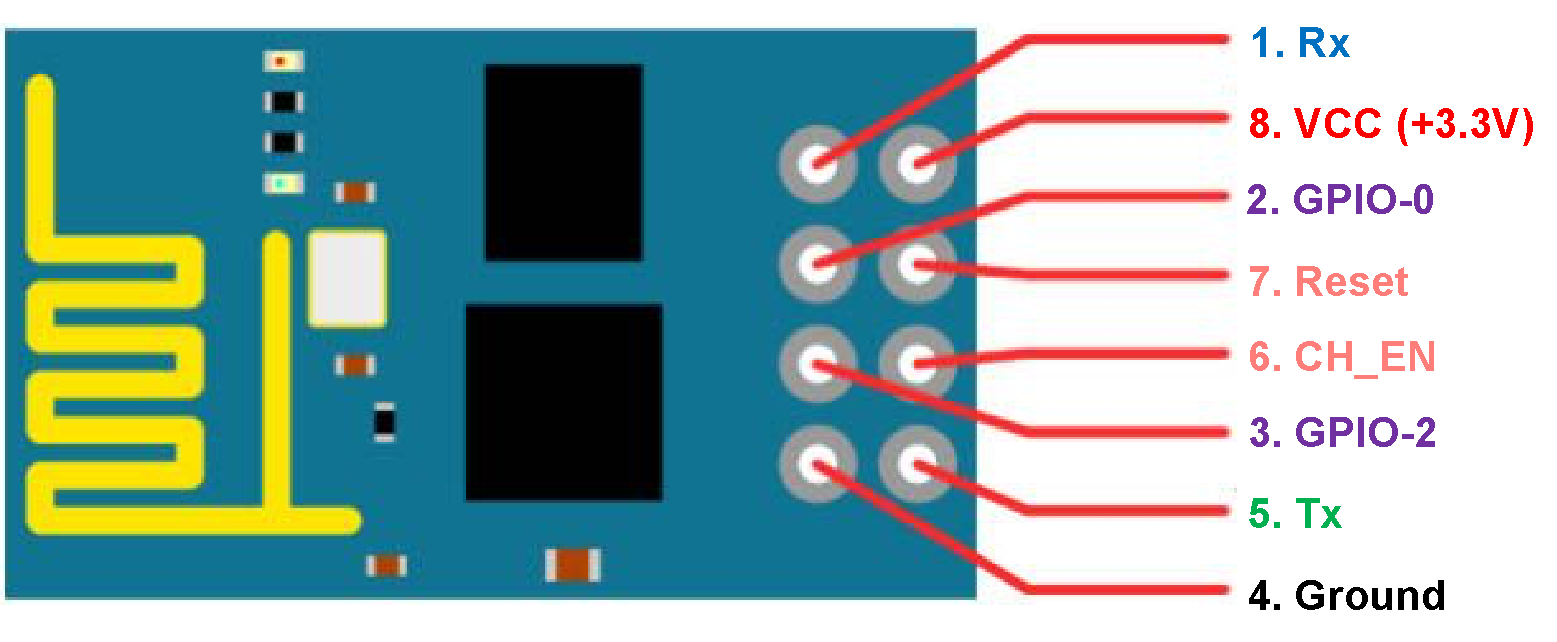
The programmer MUST be returned in working order at the end of the semester. It is very susceptible to electrostatic discharge (ESD). Keep the programmer in the box when not in use. If you destroy it, you can buy it [here](https://www.amazon.com/dp/B08F9HTR9Z/ref=twister_B08F9JQCK1?_encoding=UTF8&th=1).

### Pinout Connection

The interface between the ESP8266/ST7735 and the LaunchPad is shown below in Figures 4.4, 4.5, 4.6 and Table 4.1. The ESP8266 interface uses GPIO and UART ports on the LaunchPad. The ESP8266 interface uses UART5 running at 9600 bits/sec, 1 stop bit no parity, and no flow control (aka RTS/CTS).

**NOTE: There are two different pinout numbering schemes for the ESP8266 on the web. The schematic in Figure 4.5 below uses the pinout numbering scheme in 4.4(B) below.**

A screenshot of a cell phone

Description automatically generated

(A) (B)

Figure 4.4: ESP 8266 Pinout numbering

Diagram, schematic

Description automatically generated

Figure 4.5: Detailed schematic of the ESP8266 interface. UART5 is used. The +3V3 supply for the ESP8266 is different from the regular LaunchPad +3.3V, see Figure 4.6.

Diagram, schematic

Description automatically generated

Figure 4.6: Detailed schematic of the ST7735 interface. The CARD\_CS pin (pin 5) on the LCD is optional and needed only when the SDC card is being used. This 3.3V supply is the regular LaunchPad 3.3V.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pin | Signal | Direction | Pin | Signal | Direction |
| P1.1 | VCC | POWER | **P2.1** | GND | POWER |
| P1.2 | UNUSED | N/A | **P2.2** | UNUSED | N/A |
| P1.3 | CARD\_CS | OUT | **P2.3** | ESP\_RDY | IN |
| P1.4 | UNUSED | N/A | **P2.4** | SWITCH | IN |
| P1.5 | UNUSED | N/A | **P2.5** | ~RESET | IN |
| **P1.6** | ESP\_U5TX | OUT | **P2.6** | UNUSED | N/A |
| **P1.7** | ESP\_U5RX | IN | **P2.7** | UNUSED | N/A |
| **P1.8** | TFT\_MOSI | OUT | **P2.8** | TFT\_MISO | IN |
| **P1.9** | TFT\_DC | OUT | **P2.9** | TFT\_CS | OUT |
| **P1.10** | TFT\_RST | OUT | **P2.10** | TFT\_SCK | OUT |
|  |  |  |  |  |  |
| **P3.1** | 5V0 | POWER | **P4.1** | LED | OUT |
| **P3.2** | GND | POWER | **P4.2** | LED | OUT |
| **P3.3** | TMP36\_IN | IN | **P4.3** | UNUSED | N/A |
| **P3.4** | UNUSED | N/A | **P4.4** | UNUSED | N/A |
| **P3.5** | UNUSED | N/A | **P4.5** | UNUSED | N/A |
| **P3.6** | UNUSED | N/A | **P4.6** | UNUSED | N/A |
| **P3.7** | ESP\_RSTB | OUT | **P4.7** | UNUSED | N/A |
| **P3.8** | UNUSED | N/A | **P4.8** | UNUSED | N/A |
| **P3.9** | DEBUG | OUT | **P4.9** | UNUSED | N/A |
| **P3.10** | LED | OUT | **P4.10** | SWITCH | IN |

Table 4.1. ESP8266, ST7735 and TMP36 connections to the TM4C123 LaunchPad.

You can use PE3 for profiling the interface. The Port F LEDs and switches are used in the starter code, but you can repurpose these pins as needed for your Lab 4.

### Powering the ESP8266

The ESP8266 draws too much current to be run off the 3.3V regulator on the LaunchPad. To supply 3.3V to the ESP8266, you must build a separate 3.3V regulator using the LM2937ET-3.3 linear regulator and two 4.7 µF tantalum bypass (filter) capacitors (see Figure 9.2 in the book, with Vin=5.0V and both capacitors 4.7 µF).

1. As always, one must turn off the power while connecting or disconnecting wires to the LaunchPad. So, please connect the wires as shown in Figures 4.4 and 4.5 first, before powering.
2. Build and test the 3.3V regulator circuit shown in Figure 4.7 before connecting this 3.3V output to the ESP8266. Place a current meter between the regulator output and the ESP8266 power input.
3. The power to the LaunchPad and ST7735R will be derived from the USB cable connected to the LaunchPad. Note that you will NOT remove the jumper from the LaunchPad (removing the jumper would remove power to the TM4C123, and nobody wants that). The 3.3V output of the regulator is ONLY used to power the ESP8266 (and not connected to the LaunchPad 3.3V).
4. Turn on the LaunchPad and verify 3.3V on the LaunchPad and 3.3V to the ESP8266. Debug the software as needed.

**NOTE: You must connect ALL the grounds together or you WILL have a very bizarre circuit operation.**

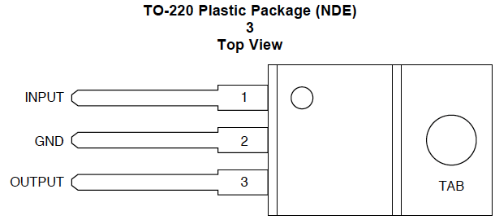
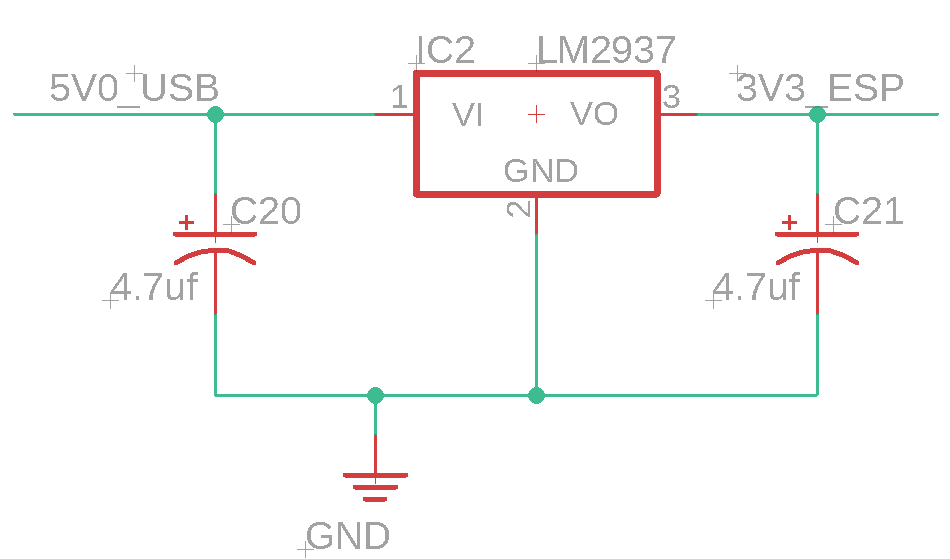


Figure 4.7: Regulator circuit using the LM2937 linear regulator. Please test this circuit before attaching to ESP8266.

## Software Review

Review the Lab 4 starter code provided in the Github classroom repository. There are three sets of software:

* The LaunchPad starter project, in which you must integrate your Lab 3 and setup data parsing;
* The ESP8266 starter project, which performs the intermediate parsing and handshakes with the MQTT broker and LaunchPad (via UART); and
* The MQTT web application starter project which displays the data collected.

### LaunchPad Starter Project

The LaunchPad starter project contains the following modules:

* **Lab4EMain.c** is a very basic template containing some basic initialization of the various modules used for Lab 4. Incorporate your Lab3 into this file.
* **MQTT.c** contains the TM4C\_to\_MQTT and MQTT\_to\_TM4C routines. This code is used to bridge the TM4C123 board and the MQTT Web Application via the ESP8266 Wi-Fi board. You will be extending the W2B parser, TM4C\_to\_MQTT, and MQTT\_to\_TM4C routines to send/receive and interpret commands between the TM4C and the MQTT WebApp. We communicate with the ESP via UART5. UART2 is used for PC debugging to see that data is sent to the ESP via serial terminal on your PC.
* **Timers:**
  + **Timer1** is used to check subscriber data from MQTT web app.
  + **Timer5** is used to publish data to the MQTT web app.
  + If you use any of these timers for functionality in your Lab3, you will need to change those timers.
* **Esp files**
  + You are given two separate files to flash to the esp8266: **esp8266\_fifo.c** and **esp8266.c**. These files will need to be merged in order to complete the lab. The file labeled **esp8266\_fifo.c** implements a low level UART interface which is required to send data between the esp8266 and the TM4C. The implementation is interrupt driven and uses a FIFO to buffer data to and from the 8266. Similarly, **esp8266.c** implements the MQTT interface that enables communication between the esp8266 and the MQTT broker. By merging these files, you will create the bridge that allows messages sent over UART by the TM4C to be sent to the MQTT broker.
* **MQTT.c** contains the TM4C\_to\_MQTT and MQTT\_to\_TM4C routines.
* **UART.c**  is an interface to PC for debugging (interrupt driven). Additionally, another UART channel is used to communicate with the esp8266.
* **Timers.c** contains code for the 3 system timers:
  + Timer0 is the 1ms (SysTick) PIT (Programmable Interval Timer).
  + Timer2 is the PIT that checks for subscriber data from MQTT Web App.
  + Timer5 is the PIT that publishes data to the MQTT Web App.

### ESP8266 Starter Project

The ESP8266 contains a single starter file, **ESP\_to\_MQTT\_LAB\_4E.ino**. Observe the following:

* Declaration of variables for authentication, MQTT connection, publish and subscribe topics
* Routines for setting up WiFi that looks for auth data from the TM4C
* Routines for connecting to the MQTT broker and subscribing to a known topic (default is ‘*/w2b*’)
* Callback for receiving messages and sending it back to the TM4C
* Periodic function call to publish any available TM4C data to the MQTT broker.

You will want to modify the high-level data variables that store your relevant topics and topic values, as well as relevant publish calls in the function *tm4c2mqtt*.

### MQTT Web Application Starter Project

The MQTT Web Application starter project contains the following modules:

* **Index.html** is the entrypoint for the web application. Open this in your web browser.
* **clock\_page.js** is a JavaScript that handles the MQTT WebSocket.It talks to the MQTT broker and performs UI and data logic.
* **mqtt\_app.htm** is the html script that displays the clocks and the buttons.
* **mqtt\_monitor.htm** is a simple MQTT Monitor that will help with debugging.

You will want to modify clock\_page.js and mqtt\_app.htm for your application specifications.

## Pub-Sub API

In MQTT, a topic is a string that is a filter a MQTT broker uses for MQTT message delivers. The broker filters all clients according to their subscription and forwards the message to subscribers/clients. A publisher and subscriber can create MQTT topics.

Here are some examples of MQTT topics:

* EER/floor1/embeddedlabroom/temperature - This topic represents the temperature in the embedded lab room, which is part of floor 1, which is part of the EER.
* USA/Texas/Austin/ManorGarage/car/1324/location - A topic structure which can be used to share the location of a specific car, in a specific garage in Austin, TX, USA.

It is up to you to define the topics you feel fit with your Alarm Clock Design from Lab3. MQTT Topics are key in communication between MQTT clients and brokers. Properly defining your MQTT Topics and the API regarding them is a crucial part of this lab. The ESP code has support for basic MODE, Hour, Minute, second topics and has corresponding API. You will extend this to support your additional features. The root topic is your eid.

# Procedure

This lab has a lot of moving parts, so we suggest you perform the lab procedure in a specific order to minimize pain.

First we recommend that students run the starter code for the ESP8266 and Web App to make sure that everything works as expected and to get an understanding of the data flow and code logic. Students should be able to communicate between the ESP8266 serial terminal (on Arduino IDE) and the web app console/monitor, seeing changes when topics are published or subscribed to.

We then encourage students to extend the system to the TM4C, and view communications on PuTTY or the ST7735 display. Some code modifications may be required here. At this point students can split up the work, with one person working on the web interface and another on the TM4C.

When the TM4C facing student has a working unit test that can send messages to the other student’s web interface, then do we encourage students to (1) extend the endpoint API to their desired specifications and (2) begin integrating Lab 3 into Lab 4.

## ESP8266 Procedure

To use the ESP8266 for this class, you will need to install the [Arduino IDE](https://docs.arduino.cc/software/ide-v2) (preferably version 2.0) to flash the .ino file provided in the starter files using the Arduino IDE. There are a couple of software dependencies:

* Adding the [ESP8266](https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/) boards to the board manager
* Adding the following libraries to the library manager:
  + *PubSubClient* by Nick O’Leary
  + *Blynk* by Volodymyr Shymanskyy including dependencies.

See the **hints** section at the end of this lab document for a more in-depth tutorial.

You will want to update the sketch with your group’s EIDs (Line 46) (e.g. mjy358\_abc123). By default, this will be **xxx**, and groups that do not change the EID may result in name collision with other groups, therefore leading to multiple people trying to update the same endpoints at the same time and causing critical sections.

After flashing the device, you can then connect to the ESP via the serial monitor (see Tools > Serial Monitor in the IDE); in the boot process you will be required to provide a comma separated value string (CSV) of a set of credentials in the order shown by Listing 4.1. An example string is provided in Figure 4.8:

**EID, SSID, PASSWORD, MQTT\_BROKER\_IP\_ADDRESS, PORT\_NUMBER,**

Listing 4.1. Sequence input.

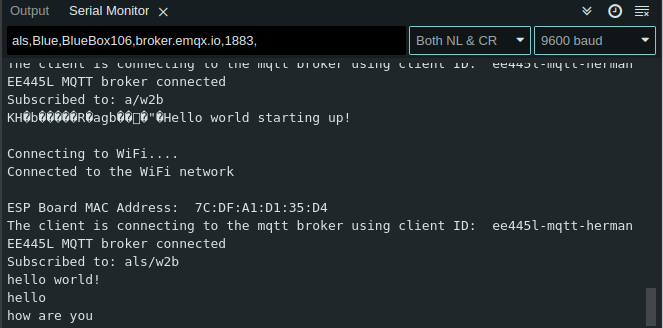


Figure 4.8. Serial Monitor.

The comma at the end is very important and will crash your ESP and restart it if not included.

You can enter the same messages that the TM4C will send to the ESP8266 to debug and confirm functionality of any changes you make to the driver. Here is an example of a typical connection to the EE445L MQTT Broker located in the EER.

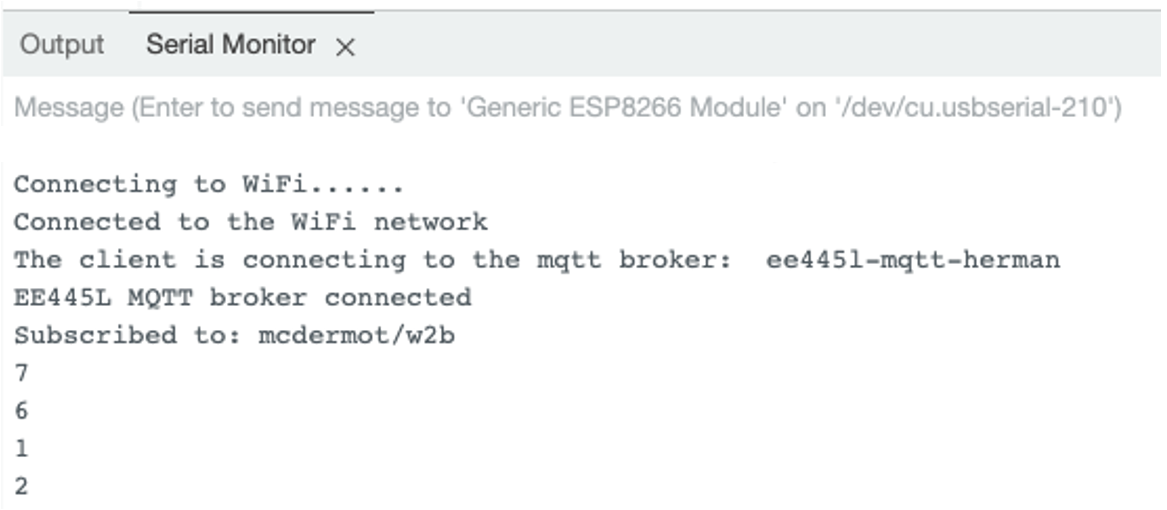


Figure 4.9. Arduino Serial Monitor.

In this example, the input is also a CSV string with the format of *mode,hour,minute,second(,).* In addition to the debug data, the “subscribe data” will also be displayed in serial monitor window (see above). In this example the MQTT Web App sent HOUR++(7), HOUR--(6), MODE(1) and SEC++(2) commands to the TM4C.

For this portion of the procedure, we ask that students:

1. Using the ESP8266 programmer, confirm that the ESP8266 can connect to a MQTT broker by manually sending commands using the serial port in the Arduino IDE.
2. **Come back to this later when you’ve verified that everything works.** You will need to make changes to the ESP8266 MQTT driver if you want to modify subscription topics or publishing topics. Typically, you will want to change the topic names to suit your application. Here is what is in the example code where b2w means board-to-web and w2b means web-to-board.

**// ----------------------------------------------------------------**

**// -------------- Publish topics --------------------------**

**//**

**const char \*pub\_mil = "/b2w/mil";**

**const char \*pub\_hour = "/b2w/hour";**

**const char \*pub\_min = "/b2w/min";**

**const char \*pub\_sec = "/b2w/sec";**

**// ----------------------------------------------------------------**

**// -------------- Subscribe topics ------------------------**

**//**

**char topic\_w2b[20] = "/w2b";**

*Listing 4.2 Pub-Sub topics.*

The ESP8266 Programmer can be used to debug the ESP8266 MQTT driver code using the Arduino IDE Serial.print function:

**Serial.print("\nConnecting to WiFi..");**

**Serial.println("\nConnected to the WiFi network");**

**Serial.print("The client is connecting to the mqtt broker: ");**

**Serial.println(client\_id.c\_str());**

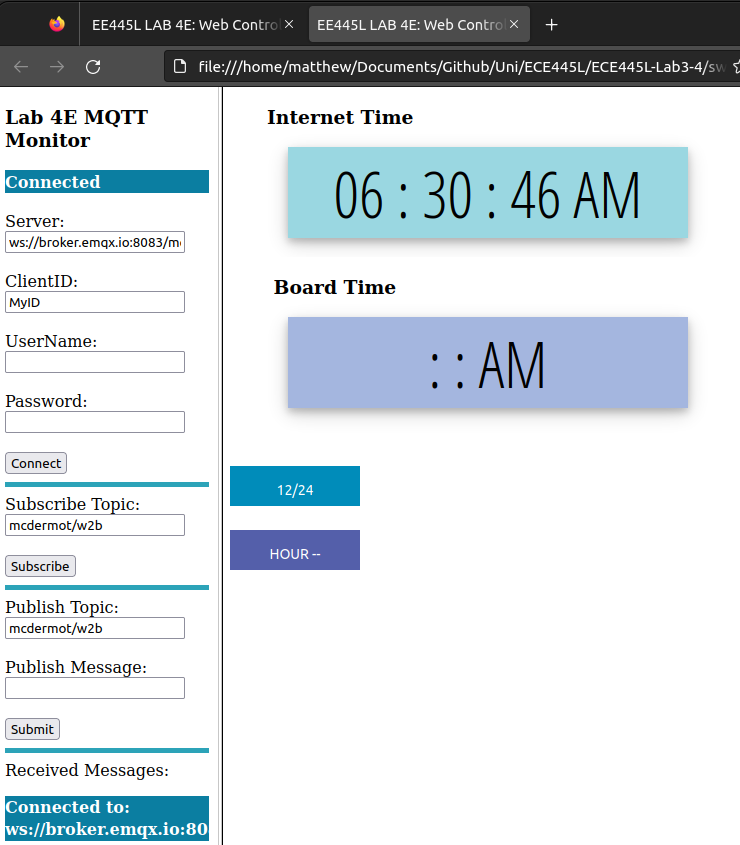
**Serial.println("EE445L MQTT broker connected");**

Listing 4.3 Debug statements.

## Web App Procedure

By default, the web app should connect to the [EMQX](https://www.emqx.com/en/mqtt/public-mqtt5-broker) server regardless of the network you are on (unless it is private and restricted to outgoing traffic).

Opening the index.html in the *sw/web* folder of your starter code on your web browser, you should see Figure 4.10. Pressing connect (even without a username and password) should connect you to the server. For a quick loopback test, you can subscribe and publish to the same topic, with some random message, and you should get a message received response.



*Figure 4.10: MQTT web app interface.*

We also encourage you to download [MQTTX](https://mqttx.app/) (an all-in-one client toolbox that supports MacOS, Windows, and Linux), to also connect and send/receive messages between the ESP and the web app. This is like the Postman equivalent of HTTP and can be very useful for testing your web interface.

The application should be relatively straightforward to set up; just make sure you have the host and port set properly, and you should be able to connect as in Figure 4.11. You can add subscriptions in the center pane and publish to arbitrary topics on the bottom right console.

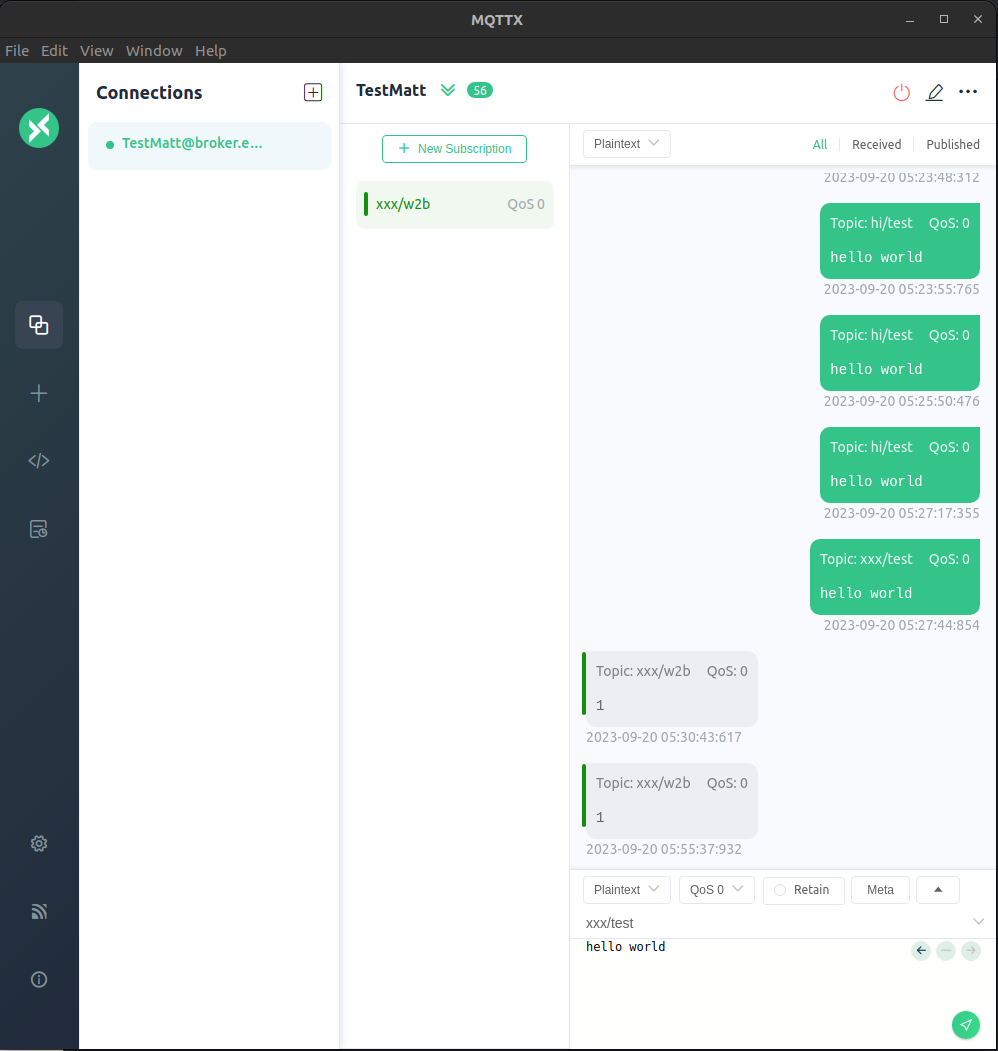
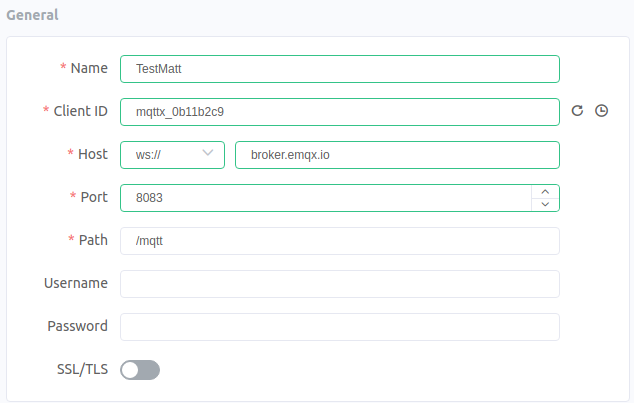


Figure 4.11. MQTTX interface.

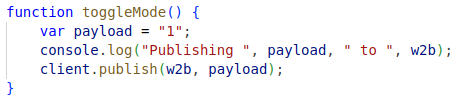
Students who work on this portion of the lab, we suggest making a copy of this code so you can use it to continue debugging ESP8266 and TM4C functionality. With one set of code (the code you will keep), you will need to make the following modifications:

1. Frames are used in this example. [iFrames](https://www.w3schools.com/html/html_iframe.asp) is now required for HTML5. Modify***index.html*** to use iFrames for your Web Page design.
2. Modify the default configuration values in ***mqtt\_monitor.htm*** for your environment.
3. Modify ***mqtt\_app.htm***to display the 7 command buttons. Add in a display for an ADC for extra credit (See w3schools for lots of HTML tutorials on adding buttons or displays, etc).
4. Modify the default configuration values in **clock\_page.js** for your environment.
5. Modify the subscribe and publish variables in **clock\_page.js** to comply with the data protocol that you have decided to use. Modify the following onConnect and onMessageArrived functions for handling more APIs.



Listing 4.4. onConnect and onMessageArrived functions.

Functions to update the ESP (and the TM4C) are called from *mqtt\_app.htm*. An example is one to toggle the MODE:



Listing 4.5. toggle\_mode function.

Add additional functions that are needed for your design.

## LaunchPad Procedure

The next step is to replace the human from the PC sending commands via the serial port with the TM4C. To validate that we get the same connection outputs from the ESP8266, we can use PuTTY (Windows only) to talk to the TM4C through the ICDI USB-UART debug chip via UART0. Reminder that this only work when the USB cable is plugged into the debug microUSB connector. The starter code uses 115200 bits/sec, 1 stop, no parity, and no flow control, which can be set in the connection properties for PuTTY.

You may also output the results on the ST7735 display. There are two #defines in esp8266.h you can select either one or the other, both or neither of the UART0 or the ST7735 LCD for debugging. DEBUG1 activates UART0 debugging and DEBUG3 activates ST7735 debugging. Use the Windows device manager to determine the COM port used to communication communicate with your LaunchPad (in Device Manager, click View -> Show hidden devices to view COM ports).

1. You will need to debug the interface between the ESP8266 and the TM4C. The ESP8266 will assert the RDY flag once it is ready to accept commands from the TM4C. Make sure all debug messages from the ESP8266 are sent to the UART0 debug port.

The following code in the TM4C loops waiting for the flag:

**while (!RDY) { // Wait for ESP8266 to indicate it is ready for programming data**

**#ifdef DEBUG1**

**UART0\_OutString(".");**

**#endif**

**DelayWait10ms(30);**

**}**

Listing 4.6. While loop on flag.

Reminder that the ESP8266 is expecting the following sequence of data to be sent to it after it asserts the RDY flag:

**EID, SSID, PASSWORD, MQTT\_BROKER\_IP\_ADDRESS, PORT\_NUMBER,**

Listing 4.7. Sequence input.

**NOTE!!!! The sequence must be terminated with a comma and a NEWLINE (\n) as shown below:**

**char eid[32] = "your-EID-goes-here";**  **// Your EID goes here**

**char ssid[32] = "EE-IOT-Platform-03";**  **// WAP in the 445L Lab**

**char pass[32] = "”;** **// Get the password from the TA**

**char mqtt\_broker[16] = "10.159.177.113”;** **// EER based broker**

**char mqtt\_port[8] = "1883";** **// TCP/IP MQTT port number**

**UART5\_OutString(eid); // Student EID - Used for individualizing MQTT Topics**

**UART5\_OutChar(',');**

**UART5\_OutString(ssid); // Send WiFi SSID to ESP8266**

**UART5\_OutChar(',');**

**UART5\_OutString(pass); // Send WiFi Password to ESP8266**

**UART5\_OutChar(',');**

**UART5\_OutString(mqtt\_broker); // Send IP address of MQTT Broker**

**UART5\_OutChar(',');**

**UART5\_OutString(mqtt\_port); // Send MQTT port number**

**UART5\_OutChar(','); // Extra comma needed for ESP8266 parser code**

**UART5\_OutChar('\n'); // Send NewLine to indicate EOT**

Listing 4.8. Arduino side UART capture and parsing.

1. Once you can make a connection to the MQTT Broker you will need to confirm that you can subscribe to a topic and publish to a topic. Using the supplied MQTT Monitor (that your lab partner should have already gotten up and running), you will publish data to a topic that the TM4C has subscribed to. The TM4C will in turn republish the data to a different topic than the Monitor has subscribed to. This will confirm that the roundtrip from the TM4C -> Broker -> Monitor -> Broker -> TM4C is functioning.
2. The next step is to set up the topics and data format between the TM4C and the WebApp. All topics MUST begin with your EID. A topic example is rg7677/b2w/hour where rg7677 is your EID and b2w is a subtopic indicating that the message is from the LaunchPad board to the Web Application and the hour is a sub-subtopic that is current hour on the board.

The topic implementation for the example code is:

**// ----------------------------------------------------------------**

**// -------------- Publish topics --------------------------**

**//**

**const char \*pub\_mode = "<your\_eid>/b2w/mode";**

**const char \*pub\_hour = "<your\_eid>/b2w/hour";**

**const char \*pub\_min = "<your\_eid>/b2w/min";**

**const char \*pub\_sec = "<your\_eid>/b2w/sec";**

**// ----------------------------------------------------------------**

**// -------------- Subscribe topics ------------------------**

**//**

**char topic\_w2b[20] = "<your\_eid>/w2b";**

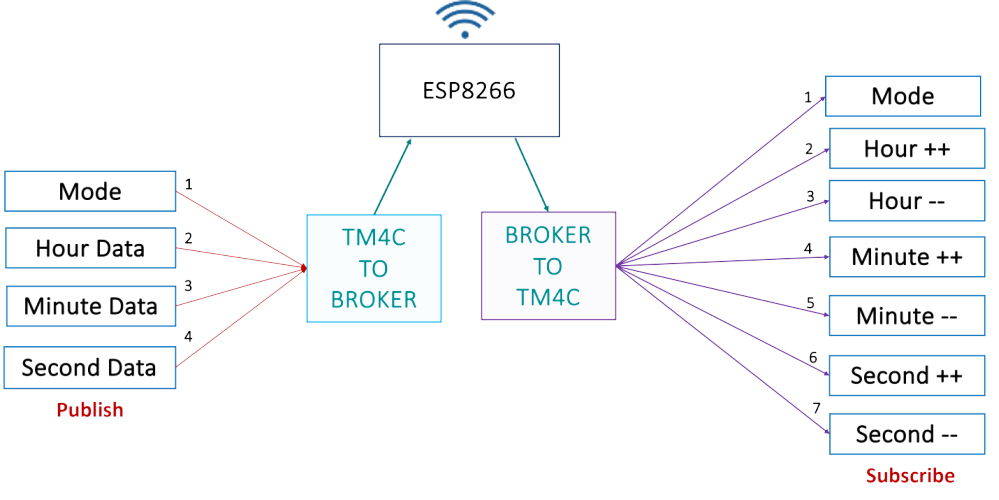
Listing 4.9. Publish and subscribe topics.

The TM4C publishes four values once a second: HOUR, MIN, SEC & MODE. The values are sent to the ESP8266 in the following format: mode, hour, min, sec,

The ESP8266 converts the CSV stream into 4 topics as shown above and sends them to the Broker. The broker sends them to the Web Application which displays the time as shown below in Figure 4.12.

The Web Application publishes a single command to the TM4C since there isn’t a need to stack commands.

You can keep the same publish-subscribe topic format for this lab or modify it to be more efficient. For example, setup the TM4C to send out the number of seconds since midnight and have the Web App convert it to Hour:Min:Sec format while keeping the 12/24-hour mode active.

Figure 4.12. TM4C pub-sub command dataflow diagram.

## Merge Lab 3 Code

Finally, you will need to merge your Lab 3 code with the program you just got up and running.

1. Merge your Lab 3 code with the starter code.

You will find these five lines around line 35 in esp8266.c. Modify the lines of code for the environment you are working in.

**char eid[32] = ""; // Your EID goes here**

**char ssid[32] = ""; // SSID for the WAP you are using**

**char pass[32] = ""; // Password for the access to the WAP**

**char mqtt\_broker[32] = ""; // IP Address to your Broker**

**char mqtt\_port[8] = "1883"; // Port for TCP/IP access to the Broker**

Listing 4.10. Auth fields.

1. Modify the following routine in MQTT.c.

**void Parser(void) {**

**uint8\_t cmd\_num = atoi(w2b\_cmd); // Need to convert ASCII command to integer**

**// ---- Command #1: Toggle MODE select**

**//**

**if(cmd\_num == 0x1) {**

**if (MODE == 0x1) MODE =0x0;**

**else if (MODE == 0x0) MODE = 0x1;**

**}**

Listing 4.11. Command parser.

Modify the example code to parse commands from the Web App based on the data protocol that you have decided to use for the lab.

1. The TM4C\_to\_MQTT routine in MQTT.c needs to be modified before it can be used.

**void TM4C\_to\_MQTT (void) {**

**char msp[24] = "";**

**sprintf(msp, "%d", MODE);**

**sprintf(msp + strlen(msp),","); // Append data in CSV format**

**...**

**}**

Listing 4.12. Output to ESP8266.

Modify the example code above to generate the correct data stream to the Web App based on the data protocol that you have decided to use for the lab. This

1. Add your Lab 2 **Dump.c** to the project and modify the **MQTT\_to\_TM4C** routine to capture incoming data like this:

**LED = pin\_int;**

**Dump\_Capture(LED);**  **// add this line**

**PortF\_Output(LED<<2);**  **// Blue LED**

Listing 4.13. Capture of incoming value.

1. Modify **SendInformation** to measure jitter like this:

**uint32\_t thisF;**

**Jitter\_Measure();**  **// add this line**

**thisF = PortF\_Input();**

Listing 4.14. Measure jitter.

1. Add calls to the corresponding initialization functions before interrupts are enabled.
2. **(Extra Credit)** Interface a sensor to the ADC in TM4C to measure the signal. Calibrate the sensor so it reads data in a human understandable fashion. One possibility uses a potentiometer to measure distance or angle. This measurement will be sent to Web App and be displayed along with the time.

## Deliverable 1

Take screenshots in the debugger showing incoming data dumps, and jitter measurements, collected using your Lab 2 dump.c code.

## Deliverable 2

Take at least three screenshots on your phone illustrating the features of your system.

## Deliverable 3

Measure the voltage and current on the 3V3 and 5V0 lines.

1. (In Lab) Disconnect the USB cable to the PC. If you have access to the lab bench supply, you can adjust the voltage output to +5V and connect it to VBUS on the LaunchPad. Verify 3.3V on the LaunchPad and 3.3V on the ESP8266. There should be between 40 and 200 mA of current on the 5V line, depending on what hardware you have connected and which software you are running. Take current measurements (both on 5V and on 3.3V to ESP8266) with
   1. just the clock;
   2. alarm on; and
   3. during Wi-Fi communication.
2. (Outside of Lab) Disconnect the USB cable to the PC. If you do NOT have access to the lab bench supply, you will remove the 2-pin jumper from the LaunchPad and connect a current meter between the pins. Reconnect the USB cable to the PC. Verify 3.3V on the LaunchPad and 3.3V on the ESP8266. There should be between 40 and 200 mA of current on the 5V line, depending on what hardware you have connected and which software you are running. Take current measurements (both on 5V and on 3.3V to ESP8266) with
   1. just the clock;
   2. alarm on; and
   3. during Wi-Fi communication.

If you only have one current meter you can perform the experiment twice repeating the same conditions.

1. Turn off power and remove the current meter(s). Connect the regulator output to the ESP8266 supply. Make sure the 2-pin jumper on the LaunchPad is conn and verify the regulator still works using a voltage measurement on the ESP8266 3.3V supply.

Document how you measured currents and report the current values (5V total and 3.3V to ESP8266) for (1) just the clock; (2) alarm on; and (3) during Wi-Fi.

# Lab Checkout

Demonstrate that your system can control the Clock on the TM4C display data using the MQTT Web Application. Demonstrate that your system can read the sensor (extra credit) and time on the TM4C. Students should be able to display understanding of the data flow through the system and between the web interface.

# Lab Report

## Deliverables

1. Objectives (Summary of lab)
2. Hardware Design
   1. Create a schematic or figure showing all external components connected to the TM4C123 board on KiCad. You do not need to show hardware components on the TM4C123 LaunchPad board.
3. Software Design
   1. System call graph including all endpoints that you added for this lab.
4. Measurement Data
   1. Deliverable 1: Screenshots of dump and jitter measurements.
   2. Deliverable 2: Screenshots of features of your system from Lab 3 working on Lab 4.
   3. Deliverable 3: Current measurements with the clock, alarm, and Wi-Fi on at various states.

# Analysis and Discussion Questions

Give short 1 or 2 sentence answers to these questions. Read this tutorial on the IoT before answering the latter questions.

1. Explain the differences between frames and iFrames.
2. Explain what console.log is used for? Show how it is used on your Web App Page.
3. Explain how the functions are called from *mqtt\_app.htm* and executed in *clock\_page.js*?
4. In the client server paradigm, explain the sequence of internet communications sent from client to server and from server to client as the client saves data on the server. Assume the client already is connected to the WiFi AP and the client knows the IP address of the server.
5. What is the purpose of the DNS?
6. What is the difference between UDP and TCP communication? More specifically when should we use UDP and when should we use TCP?

# Extra Credit

You may (for a +5% bonus) interface a sensor to the TM4C and have it report on the web interface. The type of sensor and interface is not important.

# Hints

There are more advanced monitors (such as MQTT\_Explorer) available if this homebrew tool is not adequate to debug an issue:

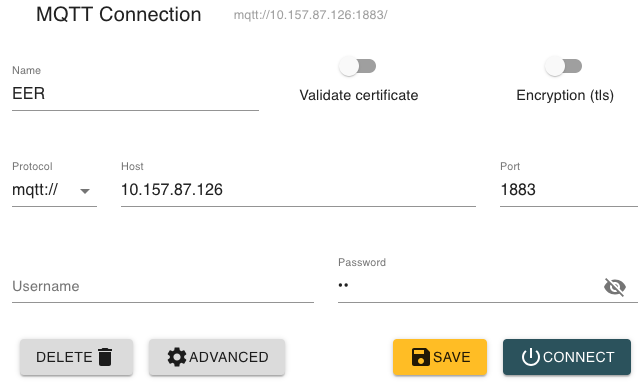


Figure 4.16. MQTT explorer.

## Browser Debugging

Use the WEB browser to receive console messages from the HTML/JS code. Embed console.log messages in your code: console.log("Connection Lost: " + responseObject.errorMessage);

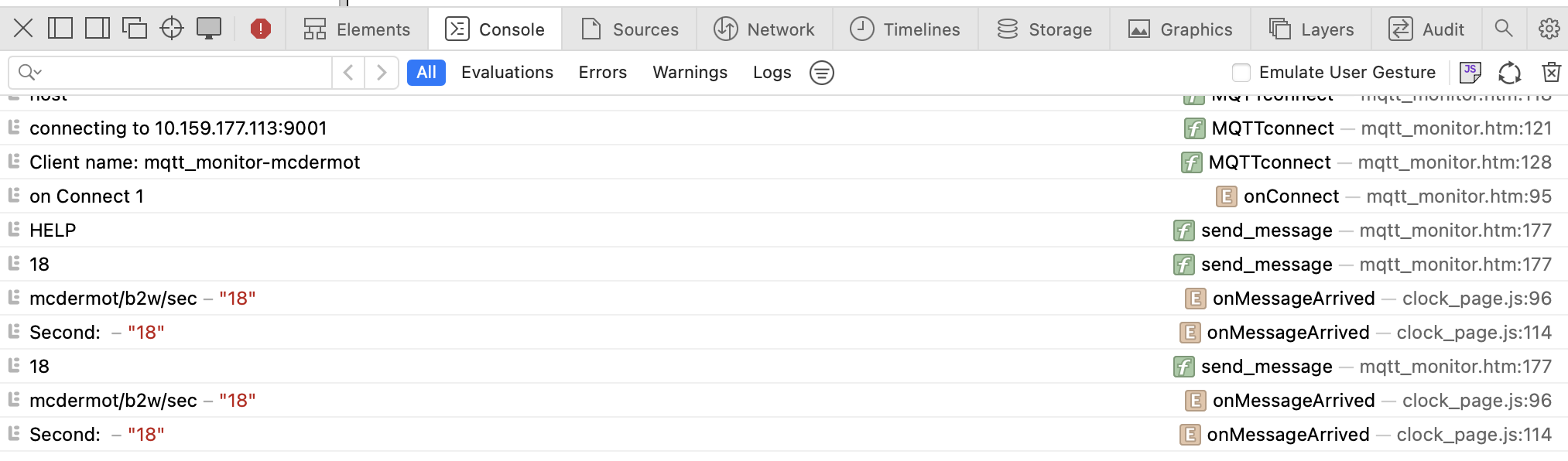


Figure 4.17. Browser console.

## Internet Traffic Analysis & Debug

If the MQTT monitors are not able help debug an issue, the next step is to look at the internet traffic between the MQTT Broker and the Web App or the TM4C. You will need to use a TCP/IP analysis tool such as WireShark. You will need to know the IP address of the Broker the computer that is running the Web App. An example of a typical MQTT packet is shown below.

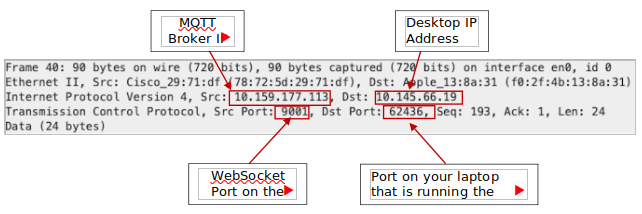


Figure 4.18. Wireshark frame.

The packet details are shown below. The subscribed data for this packet is “mcdermot/b2w/mode 0”.

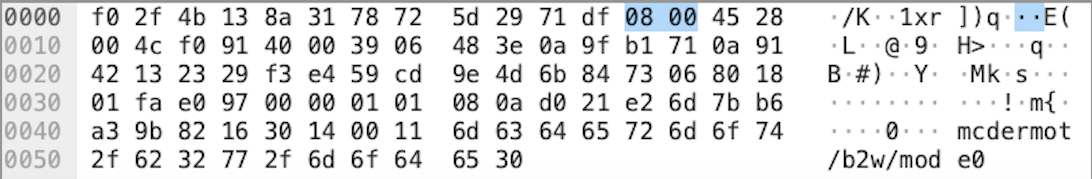


Figure 4.19. Hex packet data.

This corresponds to the subscription shown in the previous page.



Figure 4.20 Associated topic.

Refer to these three web pages for more details on how to use WireShark.

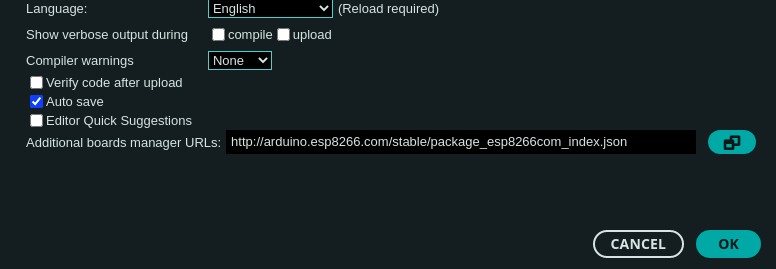
* Installation - instructions are [here](https://www.wireshark.org/docs/wsug_html_chunked/ChIntroPlatforms.html)
* WireShark - User Guide is [here](https://www.wireshark.org/download/docs/Wireshark%20User%27s%20Guide.pdf)
* WireShark - YouTube tutorial is [here](https://www.youtube.com/watch?v=TkCSr30UojM)

## Arduino Setup

You will need the following dependencies to program the ESP8266:

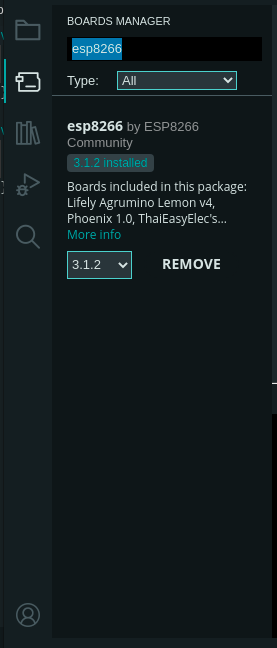
1. To program the ESP8266 via the programmer, you must have board support in the Arduino IDE. In the Arduino IDE, navigate to File-->Preferences. In the Additional board manager URL’s, add the following URL:

<http://arduino.esp8266.com/stable/package_esp8266com_index.json>



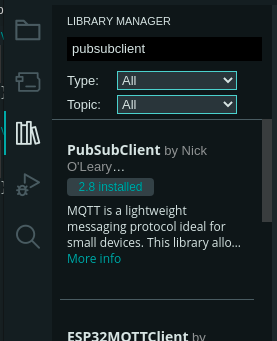
*Figure 4.8: Adding Board Manager URL*

1. Open the Board Manager and search for “ESP8266”. If the previous step worked as it should, you should see the following package from ESP8266 Community. After installing, connect the programmer to your computer via USB and select the board as a Generic ESP8266 module.



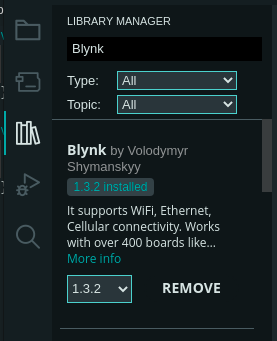
*Figure 4.9: Installing the ESP8266 Board Packages*

1. Install the MQTT Drivers: In the library manager tab, search for “pubsubclient”, install the package called PubSubClient by Nick O’Leary.



*Figure 4.10: Installing MQTT Library*

1. Install the Blynk Driver: In the library manager tab, search for “Blynk”. Install the library titled Blynk by Volodymyr Shymanskyy.



*Figure 4.11: Installing Blynk Library*

1. To program the ESP, plug in the ESP into the programmer such that it is over the bent pins. Plug in the programmer, press the reset button on the programmer, then press D15 button twice to enter the ESP Bootloader. Within the Arduino IDE, press upload. If you are unable to connect to the board, your pins may be loose. If the board fails to program, try rebuilding and flashing this board as a NodeMCU board.