ECE 445L Lab 4E Internet of Things

MQTT, Periodic Interrupts, UART, Alarm clock, LCD

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 © 2024.

# Table of Contents

[Table of Contents 1](#_Toc190121262)

[Team Size 2](#_Toc190121263)

[Goals 2](#_Toc190121264)

[Review 2](#_Toc190121265)

[Starter Files 2](#_Toc190121266)

[Required Hardware 2](#_Toc190121267)

[Lab Overview 3](#_Toc190121268)

[Preparation 4](#_Toc190121269)

[Procedure 8](#_Toc190121270)

[Deliverables 12](#_Toc190121271)

[Deliverable 1 12](#_Toc190121272)

[Deliverable 2 12](#_Toc190121273)

[Deliverable 3 12](#_Toc190121274)

[Deliverable 4 12](#_Toc190121275)

[Deliverable 5 (Extra Credit) 13](#_Toc190121276)

[Lab Checkout 13](#_Toc190121277)

[Lab Report 13](#_Toc190121278)

# Team Size

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

# Goals

* Program an ESP8266 01S (ESP8266) to act as a UART-MQTT bridge using the Arduino IDE.
* Interface the ESP8266 with the TM4C using UART.
* Implement a “smart object” that connects to a web application using a MQTT broker.
* Remotely control the clock developed in Lab 3 via the MQTT publish/subscribe interface.

# Review

* Valvano Section 11.4 on IOT
* Chapter 16 of the eBook: [Internet of Things](http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C16_InternetOfThings.htm)
* [Arduino Basics](https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics)
* [MQTT and Lab4](https://github.com/ECE445L/ECE445L-Resources/blob/main/MQTT%20Guide.docx)
* Valvano Section 3.3 on software style
* Valvano Section 3.4 on developing modular software
* Valvano Section 5.7 and 6.2 on periodic interrupts

# Starter Files

* Starter projects:
  + Lab 3 template provided on the GitHub Classroom repo.
  + Lab 4 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-ESP01 | [ESP8266 Datasheet](https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf) | 3.50 | Get it from the TA’s |
| ESP01 breadboard adapter | N/A | N/A | 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

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4.: Lab 4 Data Flow Diagram

Your team will upgrade the Lab 3 alarm clock with Wi-Fi to create a "smart object" with a web interface connected to it via an MQTT Broker. The TM4C lacks Wi-Fi, so we add the ESP8266 01S which will provide connectivity. The architecture is shown in Figure 4.1, with the right-side representing Lab 3 work and the left side showing the web interface and ESP8266.

In specific, you must be able to:

* Run the MQTT Web interface in a browser, displaying updates from the smart object. The web shows the current time (hour, minute, second) of the smart object.
* Have the smart object and its display respond to changes from the MQTT web interface. The web interface can send any of the following commands: Hour++, Hour--, Minute++, Minute--, Second++, Second--, and Toggle 12/24-hour display.
* Optionally, measure and display a calibrated measurement value on the web interface (extra credit). Any sensor and interface method (ADC, I2C, or SPI) is acceptable.

# Preparation

The lab preparation is performed during or before the W/Th lab session.

1. Edit the requirements document in the **Lab04Report.docx** to reflect your design. The requirements document is fluid - we expect it to change as you develop your solution. You should modify the requirements document.
2. Demonstrate that you can flash the ESP8266
   1. Download and install the [Arduino IDE](https://docs.arduino.cc/software/ide/). Then launch the program and navigate to File --> Preferences. In the Additional board manager URL’s, add the following URL:

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

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Adding Board Manager URL

* 1. On the left bar, open the Board Manager and search for “ESP8266”. Install “esp8266 by ESP8266 Community”. After installing, connect the programmer to your computer via USB and select the board as a Generic ESP8266 module.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Installing the board manager

* 1. On the left bar, open the Library Manager. Search for and install both the “PubSubClient by Nick O’Leary” and “Blynk by Volodymyr Shymanskyy” as shown below.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Installing software dependencies

* 1. Now to program the ESP. Open the .INO file provided with the Arduino IDE. Plug the ESP into the programmer and the programmer into the laptop as shown below. Plugging in the chip backwards or misaligned **can damage both** the ESP8266 and programmer. The programmer 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/ESP8266-Firmware-Downloader-Download-ESP-01S/dp/B08F9HTR9Z/ref=sr_1_7_sspa?crid=24QYVHT3JLITD&keywords=DIY%2BMAll%2Besp8266%2Besp-01%2Bprogrammer&qid=1692044267&s=electronics&sprefix=diy%2Bmall%2Besp8266%2Besp-01%2Bprogrammer%2Celectronics%2C209&sr=1-7-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9tdGY&th=1)

A black and yellow circuit board

AI-generated content may be incorrect.

Figure 4. ESP Orientation

* 1. Verify the proper board is selected in the IDE by going Tools -> Board -> ESP8266 -> Generic 8266 Module. Verify that the programmer is detected by navigating to Tools -> Port and selecting a valid option (You may need to [install CP210X drivers](https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads) and restart the Arduino IDE). Within the Arduino IDE on the top left corner, press upload. If this fails to flash automatically, press the reset button on the programmer, then press D15 button twice to enter the ESP Bootloader and try again.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Configuring for flashing

1. Connect the ESP to a MQTT Broker via Wi-Fi
   1. Setup Wi-Fi sharing on a device, preferably connected to UTexas. In Windows go to Settings -> Network -> Mobile Hotspot -> Edit. Ensure 2.4Ghz is selected. Turn on the hotspot and disable power saving to prevent the hotspot from turning off automatically.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Wi-Fi Sharing

* 1. Now we need to select an MQTT broker. You may use any MQTT broker, but we recommend using McDermott’s if you are on campus. Note that if you connect the ESP to non-utexas Wi-Fi **YOU WILL NOT BE ABLE TO USE McDermott’s MQTT Broker**.
     1. McDermott: IP/Domain: 10.159.177.60, Port 9001 (WebSocket) 1883 (TCP)
     2. EMQX: IP/Domain: broker.emqx.io, Port 8083 (WebSocket) 1883 (TCP)
     3. Selfhost: Setup your own broker, one option is [MOSQUITTO](https://mosquitto.org/)
  2. Read the Arduino Code. Note that you could hardcode an SSID, Wi-Fi Password, EID, and MQTT Broker at the start of the code. Note that Arduino C breaks the normal main() function of C into setup() and loop(). Note that the Setup\_Wifi() function is called from the setup() function – Specifically read how it parses inputs as Comma Separated Values (CSV) to set the SSID, Password, EID, and Broker.
  3. Ensure the Arduino code is flashed, then reset the ESP8266 on the programmer. In the Arduino IDE, navigate to Tools -> Serial Monitor. Ensure the baud rate of the monitor is consistent with the one set in setup(). You will need to send a message using the monitor, specifically the CSV described in the last step to get the ESP to connect to wifi. An example of a valid CSV is

bb37757,HP Deskjet 2624,Desk26241130,10.159.177.60,1883

* 1. You should see a message like the one below if it worked correctly. Additionally, the ESP should show up in the hotspot page discussed earlier.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Wi-Fi Connection ESP Side

1. Demonstrating bidirectional UART and Web Interface communication
   1. The following section requires some background on MQTT topics, consider reading the review documents given above. Review the output in the serial monitor. Note the topics that the ESP is subscribed to and publishing too.
   2. In the web interface starter files open index.htm, the screen is split between two files. mqtt\_monitor.htm on the left and mqtt\_app.htm on the right. When you get to modifying or reading these files, be sure to remember that these files use script tags to ‘import’ code
   3. Pay attention to the MQTT Monitor on the left. This monitor uses the same interface to the broker that your code will use. This makes it a good debugging tool since if it receives/sends data to the broker, your code should also receive data from the broker. From top to bottom the monitor has three main parts. The first is connection settings, **ONLY PRESS CONECT ONCE**, every time connect is pressed it takes the settings given and spawns a thread that interacts with the broker, if you need to reconnect refresh the page first. The second is the interaction with the broker; here you can subscribe to topics or publish to them. The responses from the broker will be recorded in the third part of the monitor. If you have issues, you can download and use alternative monitors.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. MQTT Monitor

* 1. Read the tm4c2mqtt() function in the Arduino IDE, see how it parses text received by the ESP over the Serial port (AKA UART). Note how it places the parsed data into variables of a fixed size. Consider if this limits the length of text it can parse. Consider the CSV format of the input.
  2. Click Connect on the monitor and wait for the status bar at the top of the monitor to say “Connected”. If you publish to the topic the ESP is subscribed to using the MQTT Monitor, you will see it appear in the Serial Monitor as shown below. Similarly, if we subscribe to the “eid/b2w/mode” topic in the MQTT Monitor, we will see the first element of the CSV sent to the Serial Monitor. Here we see the letter “m” appear as it was the first element sent. An example of how this looks is shown below.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Bidirectional Communication

1. Consider how this will be integrated into your lab 3 code
   1. Which pins of the ESP are used for UART? What are the Serial Calls on the ESP doing?
   2. Does the TM4C have UART? How can we periodically send/receive data to another UART Device?
   3. How will you add MQTT functionality to the Web interface? What starter code shows how to subscribe/publish to the MQTT Broker.
   4. What do b2w and w2b mean? What are topics?

# Procedure

The lab procedure is done before checkout on the M/T lab section.

To finish lab 4, you have four main tasks: 1) Integrate the ESP8266 into your Lab 3 Hardware 2) Write an ESP8266 Driver for the TM4C in Keil, 3) Finish constructing the web interface, 4) Integrate your lab3 and lab4 code.

1. Integrate the ESP8266 into your Lab 3 Hardware
   1. The ESP8266 programmer not only has everything the chip needs but also has an easy way to access the chip’s UART port. Unfortunately, it is not practical to include the programmer in your system. To replace it, we need to handle connections for Power, Reset, Enable, and Ready ourselves. The following wire diagram shows how to connect the ESP to UART5 of the TM4C.

A diagram of a wifi connection

AI-generated content may be incorrect.

Figure 4. ESP Wiring diagram

* 1. As noted in the previous figure the ESP8266 requires its own separate power supply. To understand why, consider the following excerpts from the datasheet of the ESP, TM4C Launchpad.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. ESP Power Use vs Available Power

* 1. Fortunately, providing power to the ESP is simple. We need to add an additional regulator to the design, making more 3V3 from VBUS. Note that you usually should not short the output of two regulators together. As always, you must connect ALL the grounds together or you WILL have very bizarre circuit operation.

A diagram of a circuit

AI-generated content may be incorrect.

Figure 4. ESP Power Rail

1. Write an ESP8266 Driver in Keil
   1. The ESP8266 only has 8 pins, with 4 being VCC/GND/RST/EN, it's no surprise that the Serial Monitor uses the same UART interface that the TM4C will. Recall that whenever the ESP's Serial is written to or read from, UART TX/RX is triggered. This is good since we can test any protocol between the ESP and TM4C manually with the debugger. The downside is that in the final ESP code we use, we may want to either omit the debug messages from the output **OR** make out TM4C driver understand and ignore the debug messages the ESP generates.
   2. Before beginning implementation, note that the specific communication protocol your web interface and smart object use on top of MQTT is entirely up to you, and you should consider what you want to send or receive in any given topic. (I.E. What will the TM4C specifically sent to the web interface to indicate it is in military/regular time. How will the web interface command the TM4C to increment the hour/min/sec via the W2B topic? Consider the adhering to the parser limitations mentioned earlier or commit to editing the Arduino code.)
   3. When the ESP8266 and TM4C are wired together, you can snoop on their communications using the ESP programmer. If you connect the RX pin of the programmer (Normally connected to the TX pin of the ESP) to the UART lines of the circuit, you can get a real-time readout of what is happening. Note however, you cannot normally use the TX pin in the same fashion as each transmitter has its own pull up.
   4. MQTT.c and esp8266\_base.c are starter code for the TM4C’s ESP driver. esp8266\_base.c should contain functions to reset and connect the ESP to Wi-Fi. MQTT.c contains functions that check for data from or send data to the ESP. The MQTT.c functions should be called periodically.
      1. When updating the esp8266\_base.c functions recall what the ESP will do during setup (Literally setup() from the Arduino IDE). Consider what the ESP does to signal when it has finished connecting to Wi-Fi. Consider why the TM4C might want to avoid sending/parsing data to/from the ESP until setup is done.
      2. When considering how often to call the MQTT functions, consider why we might need to check for incoming data frequently from the ESP (Hint UART fifo size). Consider why we might not want to send data to the ESP too frequently.
   5. The minimum data flowing through the ESP, not counting the extra credit is described by the following diagram.

****

Figure 4. TM4C Publish-Subscribe Command Dataflow Diagram

* 1. Additionally, you may find it helpful to use UART0 for debugging purposes. The TM4C can communicate with your laptop via UART0 in a similar way to the ESP’s Serial functions. You may find it helpful to output both what the TM4C sends and receives from the ESP on UART5 to UART0.

1. Finish constructing the web interface
   1. The web app has several files
      1. **clock\_page.js** is a JavaScript that handles the MQTT WebSocket I/F (Port 9001)
      2. **index.html**  is a html file that instantiates the other .htm files
      3. **mqtt\_app.htm** is the html script that displays the clocks and the buttons.
      4. **mqtt\_monitor.htm** is a simple MQTT Monitor that will help with debug.
   2. Start with mqtt\_app.htm, here you will add the buttons for the interface. Two buttons are already given. Critically, you should look at the class and onclick functions of the buttons. The class can be used to get the buttons from the DOM, and the onclick function will be called from clock\_page.js when the button is clicked.

A group of text on a white background

AI-generated content may be incorrect.

Figure 4. .htm buttons

* 1. Add functions and edit clock\_page.js as needed. Note that the Board\_Time() function calls itself every second, and has code used to update the displayed time from global variables. You can see this by opening inspect element when the index.html page is open in your browser, then setting the global variable hour = 12 in the console. The implication of this is that when a button is pressed from the htm you should publish data, and when data is received from MQTT you should set global variables.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Browser Dev Console (Inspect Element)

* 1. Once the web app is working, update ***index.html*** from Frames to iFrames. Frames are used in this example. [iFrames](https://www.w3schools.com/html/html_iframe.asp) is now required for HTML5. Consider the difference between these two types.

1. Integrate your lab 3 and lab 4 code.
   1. Considering the variety of alarm clocks students make in Lab 3, you probably know your code best. In general, we would recommend doing this piece by piece. This is because you may need to rewrite portions of the code, mainly as the MQTT code may use pins/timers you used in lab 3. It gets hard to keep track of what needs to be updated if you copy everything into the project at once. Additionally you may exceed stack or memory limits, and I find the incremental process helps you solve this problem.
   2. If you are unfamiliar, you may need to add your files to the project on the right-hand side project window -> right click on src -> Add existing files -> Select the lab 3 source files you added to lab 4. You will need to merge the Lab 3 and Lab 4 mains, but you should be mindful of the order to initialize and use pins.
   3. As part of Deliverable 3 you will need to add Dump\_Capture() and Jitter\_Measure() from Lab 2 to MQTT.c

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Adding project Files

# Deliverables

Deliverables are completed for the lab report, but checkout Q&A may be about deliverables.

## Deliverable 1

Draw the electrical circuit you used to create the alarm clock turned Smart Object. You can copy your work from the lab 3 schematic into the lab 4 schematic if both windows are open. You do not need to show hardware components on the TM4C123 LaunchPad board. Your schematic should include at least the TM4C, The ST7735, the LDO, and the ESP.

## Deliverable 2

Take screenshots in the debugger showing incoming data dumps, and jitter measurements. This data should be collected using your Lab 2 dump.c code. Specifically, you should use:

1. **Dump\_Capture**() to record some of the outgoing data (Such as what value of second is being sent) in the **TM4C\_to\_MQTT**() function calls.
2. **Jitter\_Measure**() to record the variance in timing of the **MQTT\_to\_TM4C**() function calls.

## Deliverable 3

Take at least three pictures of the final alarm clock demonstrating the features of your system. These pictures should show that:

1. The clock and the web interface show the same time after initialization. Use the MQTT monitor to show what data is coming in for the seconds topic.
2. The unique features that you demonstrated in lab 3 are still present in the updated lab.

## Deliverable 4

Disconnect the TM4C’s USB cable from the PC. Using the bench supply, adjust the output voltage to +5V and the current limit to 500mA; then connect it to VBUS on the LaunchPad. Verify the 3.3V rails on the LaunchPad and ESP8266 using a multimeter. There should be between 40 and 250 mA of current on the 5V line, depending on what hardware you have connected and which software you are running. Take current measurements (Displayed on the bench power supply) with

* 1. the clock idling without the ESP active
  2. the clock with the alarm on
  3. the clock idling with the ESP active

## Deliverable 5 (Extra Credit)

Interface any sensor to the TM4C and have its **calibrated** output show on the web interface. A valid solution will have a code that calibrates the sensors to measured results and reports these results in a human readable format to the end user. This measurement must be sent to Web App and be displayed alongside the time. Examples include a distance sensor reporting inches, a digital protractor measuring degrees, or a temperature sensor reporting degrees Celsius.

# Lab Checkout

The lab checkout is performed during the M/T lab session.

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

The lab report shall be submitted by the Friday after the second lab section.

You should complete the Lab04EReport.docx file with your data and answers then submit the completed file to Canvas.