A10G Cloud (S24)

A **group** submission to be submitted on Github Classroom & Gradescope.

**Github Classroom assignment:** <https://classroom.github.com/a/SWcHbWxo>

Remember to read the [ESE5160 S24 Assignment README](https://docs.google.com/document/d/1pPXQByy8eTxTJ--3vO8KpTjMk5yBHF8wQXoLJ55w5a8/edit) before starting!

If you need to use a late day, you must submit using [this form](https://docs.google.com/forms/d/e/1FAIpQLSd2hfFc7tIAqP-B1GouC5sP6Zbl59p7JXJa_yGTR60CJHRU3A/viewform).

# Table of Contents

[**Table of Contents 1**](#_q6d2hz4sk96g)

[**Learning Objectives 1**](#_oqo0md4nmgwo)

[**0. Starter Manual 2**](#_v6w6qnpmj61b)

[**1. OTAFU 2**](#_3j6q2dhnjhlb)

[**2. Node-RED Design 2**](#_ywfvmelvfvtx)

[**3. Node-RED Implementation 3**](#_7p8q3eo22wsa)

[**4. Bidirectional Cloud Communication 3**](#_m78u609idmu)

[**5. Percepio Analysis 4**](#_a04oticbzbe5)

[**Rubric 5**](#_psy3hy66jdwd)

# Learning Objectives

* Learn how to set up a virtual machine to act as our file host, Node-RED host, and MQTT broker.
* Use the VM file hosting to complete the over-the-air update capability on the MCU, adding any additional debugging handling.
* Design and implement a compelling and intuitive user interface for control of your Internet-connected device.
* Complete the communication loop between MCU and the cloud using MQTT.

# 

# 0. Starter Manual

Follow the **A10 Cloud Manual** in the Google Drive to set up your system. This will guide you on how to set up a virtual machine on Microsoft Azure, then use this as an HTTP host for firmware binary and status files. It will also show you how to set up a Node-RED instance and an MQTT broker communication server.

Continue programming your firmware in the same IoTStarter Project used in A09G.

# 1. OTAFU

The starter project has a CLI command, “fw”, that downloads a file from a server. Modify the code on this CLI command to download a file set in your server that is a firmware update. Modify the code in a way that you can make a complete OTAFU: The device downloads an image from the internet, saves it into SD card, adds any flags that your bootloader needs, and then resets the MCU to perform a FW update. You can even add text files to the server that have metadata for the firmware image - like version, intended hardware, and other information.

Note: You can reset the MCU with the command **system\_reset()**.

**Submission:**

* A short video/GIF showing that your device can update its firmware via the Internet. Show both the device behavior and the CLI readout in the video.
* Code submitted to the Github repository.

# 2. Node-RED Design

Before doing any coding, design the MQTT topics and program flow your system will use. In your README.md, explaining the following:

* General program flow of your system. What communication does the device need to send to the cloud? What does the cloud do with this information? What does it send back?
* List all the topics your system will use. Describe for each one the information that is sent on each topic (is it an integer? A Boolean? An array of integers? A JSON string?)
* Describe for each topic in your system: Who generates the data? Who subscribes to each topic?
* Present how you would divide your MCU Application code into threads, how the threads would communicate, etc.

**Submission:**

* README.md section explaining your Node-RED design (the questions above). A reader should be able to understand how your system will work at a high level with this document.
* Use text and diagrams as needed to explain your approach.

# 3. Node-RED Implementation

We can implement and simulate what happens on the Node-RED part of the system without programming our microcontroller devices. We can leverage the use of a web-browser based MQTT server to simulate input and output to/from the Node-RED Design. This will allow us to “divide and conquer” – divide the design into smaller, more achievable tasks.

In this part of the assignment, implement your complete Node-RED system and test how it works by using the MQTT broker you set up on your Azure virtual machine.

Unit test your complete Node-RED system – implement the complete workflow, and use MQTT messages, sent by a client broker on your Azure VM, to test that the inputs work. Be sure to test everything!

Make the frontend to show to the user all the important information. This includes graphs, dials, buttons, etc. In short – all that you need to put in the frontend to allow a user to see your system in function! What you implement should be complete – that is, it should have all the logic that your application needs as well as all the user interface required, and everything should be tested to work.

Ensure you’re using your Azure VM MQTT broker, NOT HiveMQ. You’ll get a 0 if you’re not using your own MQTT broker.

**Submission:**

* A video of your Node-RED setup working and explain how it operates.
* Screenshots of your backend and frontend of Node-RED.
* Commit your Node-RED source code to your Github repository in the Node-RED folder.
* Submit a link to your Node-RED UI (the public URL).

# 4. Bidirectional Cloud Communication

Now that you have your own VM with Node-RED and MQTT, it’s time to set up communication between your MCU and the cloud. For starters, we’ll use your debug LED and debug button.

Behavior:

* Node-RED UI Switch: controls the state of the debug LED on your MCU HW (on or off)
* MCU Debug Button: turns on a Node-RED UI LED when the HW button is pressed (and turns off the LED when it is not pressed)

**WifiHandler.c** already sets up your MQTT connection and has examples of the functions **mqtt\_subscribe** & **mqtt\_publish**.

Ensure you’re using your Azure VM MQTT broker, NOT HiveMQ. You’ll get a 0 if you’re not using your own MQTT broker.

**Submission:**

* A gif/video of this behavior working.
* Commit your Node-RED source code to your Github repository in the Node-RED folder.
* Submit a link to your Node-RED UI (the public URL).
* Commit your firmware to the Github repository.

# 5. Percepio Analysis

Percepio gives us an easy way to analyze how our tasks are functioning in our firmware repository. Using Percepio, capture a trace of the firmware when it is running all tasks in your system. This should be somewhat complex by now.

**Submission:**

* A screenshot of Percepio capturing the tasks that are running.
* In your README.md, describe your insights from reviewing the Percepio Tracelyzer analysis.

# 

# Rubric

While the rubric attempts to capture all assignments details, points assigned may vary based on submission quality and teaching team review. Please ensure you read the assignment carefully so as not to miss details and lose points. Poor readability / formatting can lose you points on any assignment.

**For all questions, 0 points will be awarded if the submission is non-existent, very poorly done, or doesn’t compile (for firmware assignments).**

| **Max Points** | **Question** | **How to achieve full credit** |
| --- | --- | --- |
| 15 | 1. OTAFU | Video shows clearly that an OTAFU was done on the system. |
| 15 | 2. Node-RED Design | A document explaining the topics of the system and the general program flow is provided. Each topic proposed mentions what data is transmitted in it and how it is used by the system |
| 15 | 3. Node-RED Implementation | A Node-RED system is provided (Small explanatory video + pictures + code + URL). |
| 10 | 4. Bidirectional Cloud Communication | Successful bidirectional communication with the cloud is demonstrated in video form. Firmware and Node-RED UI JSON code is submitted. |
| 5 | 5. Percepio Analysis | Percepio has been used correctly to analyze the FreeRTOS Thread behavior. Insights have been written in the README.md. |
| 60 |  | Total Achievable Points |