Wi-Fi Accessibility

Overview:

In this lab you will learn how to program your LaunchPad to communicate with the cloud. You will begin to understand the MQTT protocol and how you can both publish and receive information from the cloud. In the last section of the lab, you will implement a cloud-based version of your binary counter from Lab 5 part 2.

Update PubSubClient Library:

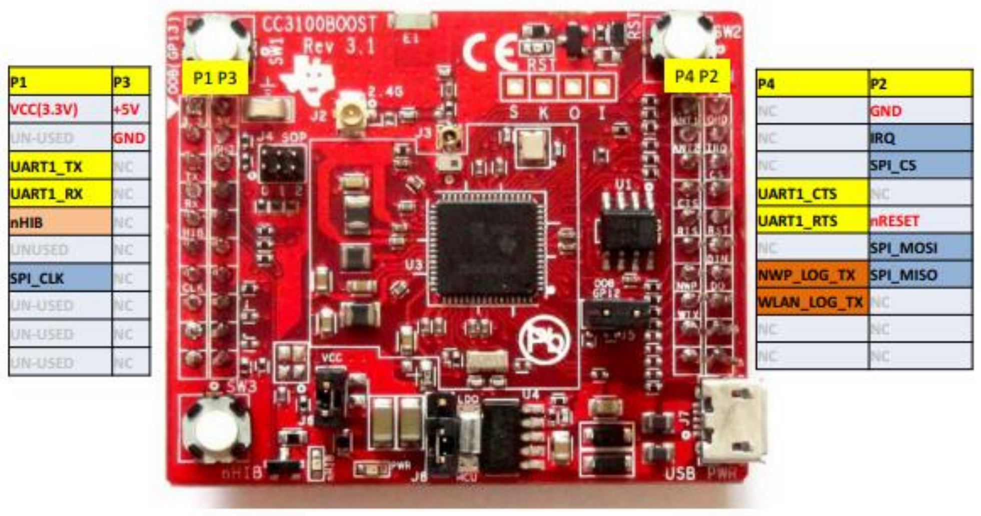
In this lab we will be using a useful library called "PubSubClient.h" to interface with our MQTT broker. Unfortunately, the default "PubSubClient.h" library is slightly out of date. Follow the steps below to update the library.

1. Download the updated library provided to your local machine.
2. Visit the location on your computer where you downloaded Energia. Go to the "libraries" directory and delete the folder "PubSubClient."
3. Finally, move the new PubSubClient to this same "libraries" folder. You should restart Energia.

Attaching the Wi-Fi Booster Pack:

This section will describe the procedure for connecting the Wi-Fi Booster Pack.

1. Unbox the BoosterPack and remove it from its protective anti-static storage bag.
2. On the BoosterPack there are rows of connectors, two on each side as shown in Figure 1.



P1	P3
VCC(3.3V)	+5V
UN-USED	GND
UART1_TX	NC
UART1_RX	NC
nHIB	NC
UNUSED	NC
SPI_CLK	NC
UN-USED	NC
UN-USED	NC
UN-USED	NC

P4	P2
NC	GND
NC	IRQ
NC	SPI_CS
UART1_CTS	NC
UART1_RTS	nRESET
NC	SPI_MOSI
NWP_LOG_TX	SPI_MISO
WLAN_LOG_TX	NC
NC	NC
NC	NC

Outer Row Connectors

Pin No	Signal Name	Direction	Pin No	Signal Name	Direction
P1.1	VCC (3.3 V)	IN	P2.1	GND	IN
P1.2	UNUSED	NA	P2.2	IRQ	OUT
P1.3	UART1_TX	OUT	P2.3	SPI_CS	IN
P1.4	UART1_RX	IN	P2.4	UNUSED	NA
P1.5	nHIB	IN	P2.5	nRESET	IN
P1.6	UNUSED	NA	P2.6	SPI_MOSI	IN
P1.7	SPI_CLK	IN	P2.7	SPI_MISO	OUT
P1.8	UNUSED	NA	P2.8	UNUSED	NA
P1.9	UNUSED	NA	P2.9	UNUSED	NA
P1.10	UNUSED	NA	P2.10	UNUSED	NA

Inner Row Connectors

Pin No	Signal Name	Direction	Pin No	Signal Name	Direction
P3.1	+5 V	IN	P4.1	UNUSED	OUT
P3.2	GND	IN	P4.2	UNUSED	OUT
P3.3	UNUSED	NA	P4.3	UNUSED	NA
P3.4	UNUSED	NA	P4.4	UART1_CTS	IN
P3.5	UNUSED	NA	P4.5	UART1_RTS	OUT
P3.6	UNUSED	NA	P4.6	UNUSED	NA
P3.7	UNUSED	NA	P4.7	NWP_LOG_TX	OUT
P3.8	UNUSED	NA	P4.8	WLAN_LOG_TX	OUT
P3.9	UNUSED	NA	P4.9	UNUSED	IN
P3.10	UNUSED	NA	P4.10	UNUSED	OUT

Figure 1: Pinout Diagram for the CC3100 SimpleLink Wi-Fi and IOT BoosterPack

3. The BoosterPack has the same number of connectors as the MSP432P401R Launchpad. We are going to connect the Wi-Fi module on the **bottom** female header pins of the MSP432P401R Launchpad (see Figure 2).

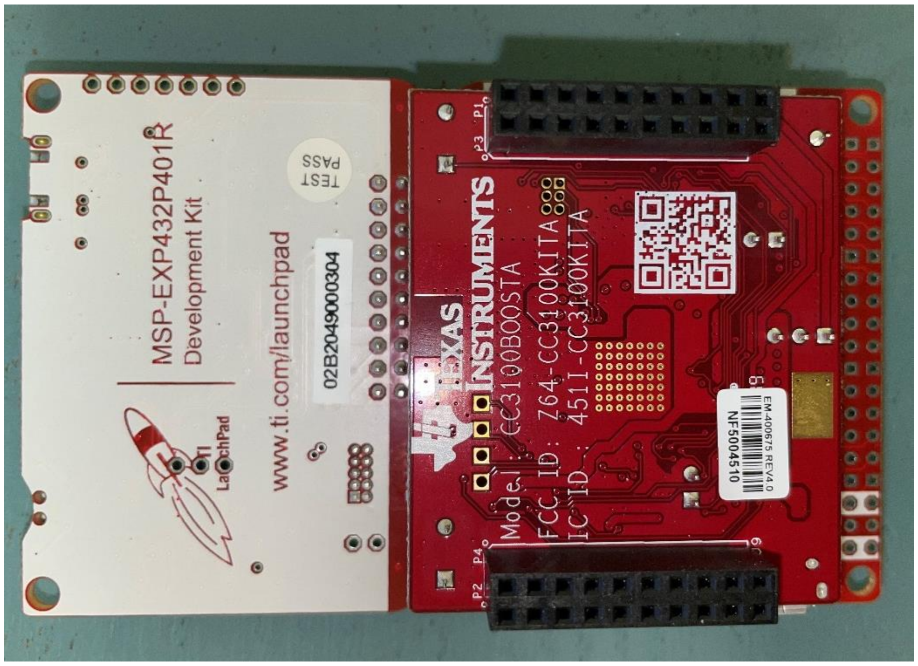


Figure 2: CC3100 BoosterPack stacked underneath MSP432P401R

4. Before connecting the BoosterPack, ensure the Launchpad is **not** connected to power. This will ensure that you will not damage the board if pins are misaligned. As seen in Figure 2, the BoosterPack and Launchpad are properly aligned if the logo on both the back of Launchpad and BoosterPack are both facing upward.

5. The Launchpad and BoosterPack should now be properly connected.

Sending Data to the Cloud through Adafruit IO:
In the first section of this lab, we will design a simple example to get started with sending data to Adafruit IO. Adafruit IO is a free cloud service that uses a cloud interface protocol called MQTT. If you are interested in the nuances of this protocol visit the link below.

MQTT Protocol: https://mqtt.org/

1. First you need to add some libraries to make setting up a MQTT connection easier. Go to the following GitHub page and download the library.

    a. Adafruit MQTT Library: https://github.com/adafruit/Adafruit_MQTT_Library

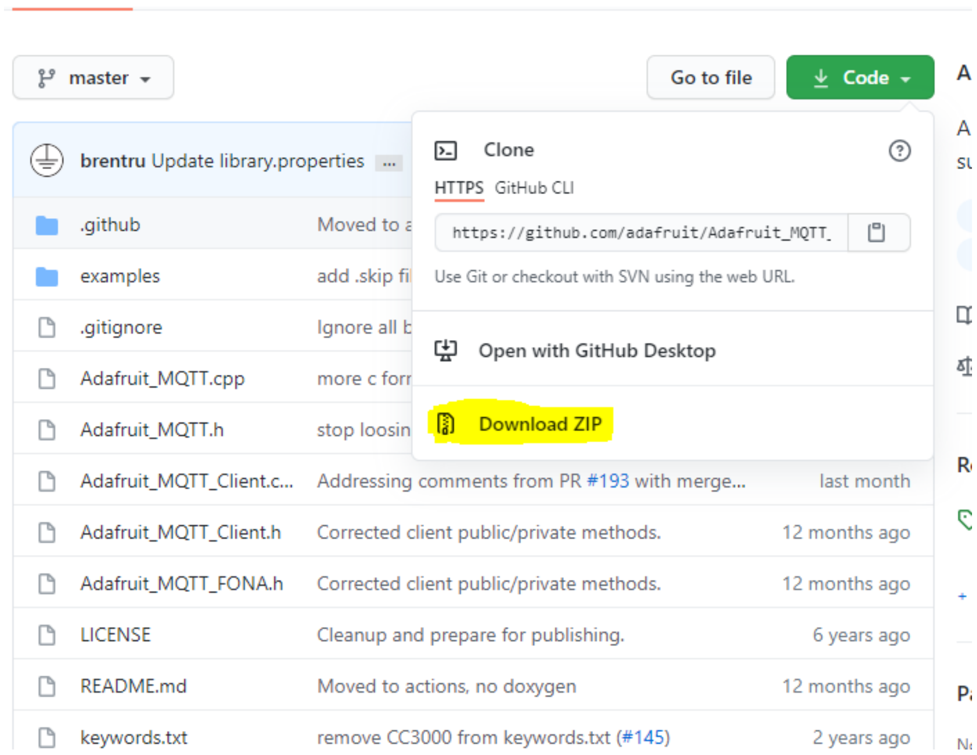
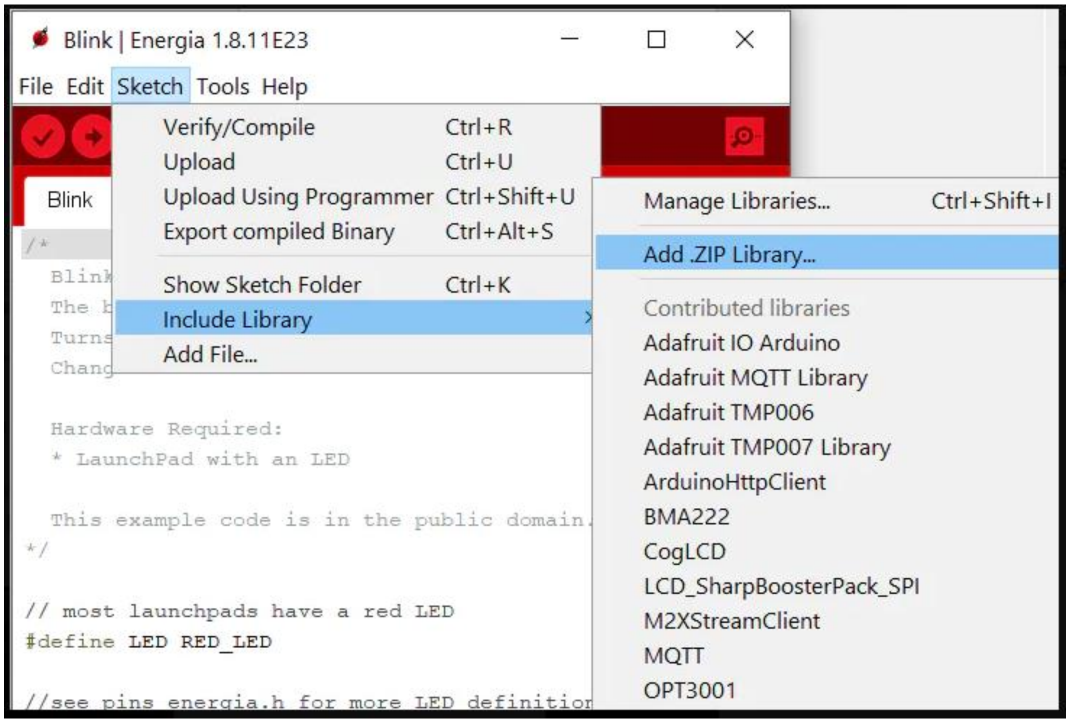

master
Go to file
Code
brentru Update library.properties
.github
examples
.gitignore
Adafruit_MQTT.cpp
Adafruit_MQTT.h
Adafruit_MQTT_Client.c...
Adafruit_MQTT_Client.h
Adafruit_MQTT_FONA.h
LICENSE
README.md
keywords.txt
Addressing comments from PR #193 with merge... last month
Corrected client public/private methods. 12 months ago
Corrected client public/private methods. 12 months ago
Cleanup and prepare for publishing. 6 years ago
Moved to actions, no doxygen 12 months ago
remove CC3000 from keywords.txt (#145) 2 years ago
Clone
HTTPS GitHub CLI
https://github.com/adafruit/Adafruit_MQTT_
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

Figure 3: Adafruit MQTT Library download

b. Now, we must add the MQTT Library in Energia. This is done by going to Sketch > Include Library > Add .ZIP library.



```
/*
  Blink
  The b
  Turns
  Chang

  Hardware Required:
  * LaunchPad with an LED

  This example code is in the public domain.
*/

// most launchpads have a red LED
#define LED RED_LED

//see pins_energia.h for more LED definitio
```

Figure 4: Adding a library to the sketch

2. Next you must go and setup an account with Adafruit IO.

    a. Visit this link to create an account with Adafruit IO: https://accounts.adafruit.com/users/sign_up

    b. Once your account is created, you will see your dashboards page. Navigate to the "Feeds" file header and select "View All."

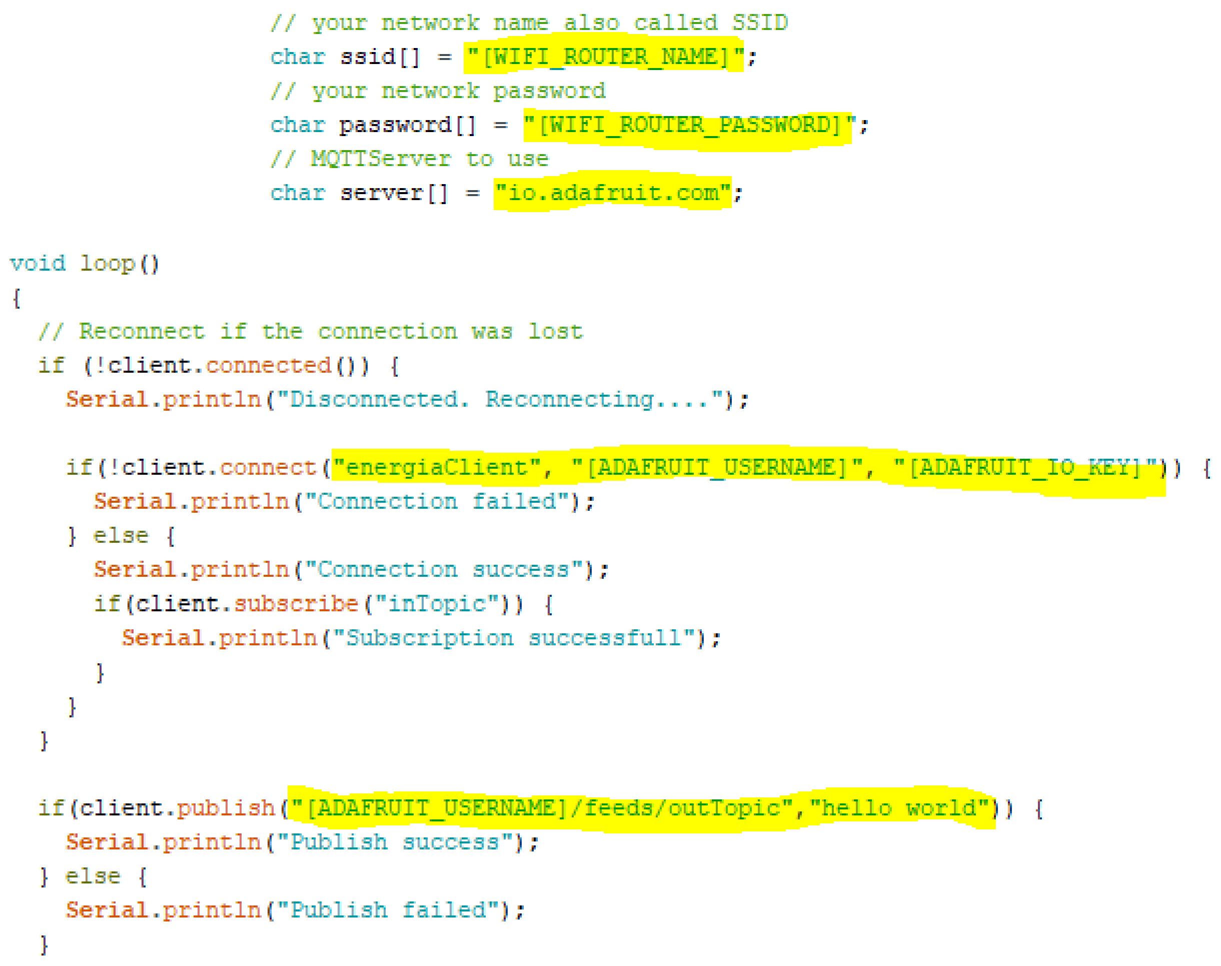
    c. Next, you should create a new feed and name it "outTopic."

Think of a feed as a data stream. Data can be published from devices to a feed and devices can subscribe to a feed to get the data. So, an easy example is a button. A button is either ON or OFF. If we make a feed to keep track of the state of the button, we can update that feed with a new value if it changes and we can also subscribe to that feed to get the state of the button.

3. We will now send the string "Hello World" to our feed "outTopic." In Energia, open the example "MQTTBasicWiFi.ino" by following the path File > Examples > PubSubClient > MQTTBasicWiFi.

4. Take some time to familiarize yourself with the code in this sketch. In short, we connect to a Wi-Fi network through the CC3100 BoosterPack. We then send data to a MQTT server topic named "outTopic," and conversely, receive data from a topic named "inTopic."

5. Now, you need to specify your Wi-Fi credentials and your unique Adafruit IO key (found under "My Key" directory of your Adafruit account). These parameters are illustrated in the figure below.



```
// your network name also called SSID
char ssid[] = "[WIFI_ROUTER_NAME]";
// your network password
char password[] = "[WIFI_ROUTER_PASSWORD]";
// MQTTServer to use
char server[] = "io.adafruit.com";

void loop()
{
  // Reconnect if the connection was lost
  if (!client.connected()) {
    Serial.println("Disconnected. Reconnecting....");

    if(!client.connect("energiaClient", "[ADAFRUIT_USERNAME]", "[ADAFRUIT_IO_KEY]")) {
      Serial.println("Connection failed");
    } else {
      Serial.println("Connection success");
      if(client.subscribe("inTopic")) {
        Serial.println("Subscription successfull");
      }
    }
  }

  if(client.publish("[ADAFRUIT_USERNAME]/feeds/outTopic","hello world")) {
    Serial.println("Publish success");
  } else {
    Serial.println("Publish failed");
  }
```

Figure 5: Wi-Fi and MQTT server parameters

6. Finally, run your sketch. Visit your “outTopic” feed on the Adafruit site and you should see that you are effectively communicating with the cloud! If you do not see data posted, open your serial terminal to troubleshoot the issue.

*Note: You will likely see warnings after compiling this sketch. You may ignore these as they are an artifact of the header file we added. The sketch should compile and run as expected.

Receiving Data from the Cloud through Adafruit IO:

In this part of the lab, you will learn how to send data from the cloud to your microcontroller. A large benefit of IOT based solutions is that the client (our LaunchPad) can perform simple tasks, such as data collection, fast and effectively while a server on the cloud can perform extensive data analysis remotely. Sending control data back from the cloud is crucial for this relationship to work.

1. For this part of the lab, create a new feed called “button1.” We will have our LaunchPad “subscribe” to this feed and receive data.

2. Now, you must create a user interface on Adafruit IO. This is achieved through the “Dashboards” directory. Go to Dashboards > View All > New Dashboard. Name this dashboard “button1dashboard.”

3. In this new dashboard add a toggle switch. This is achieved by selecting “Create New Block” and selecting the “toggle” block.

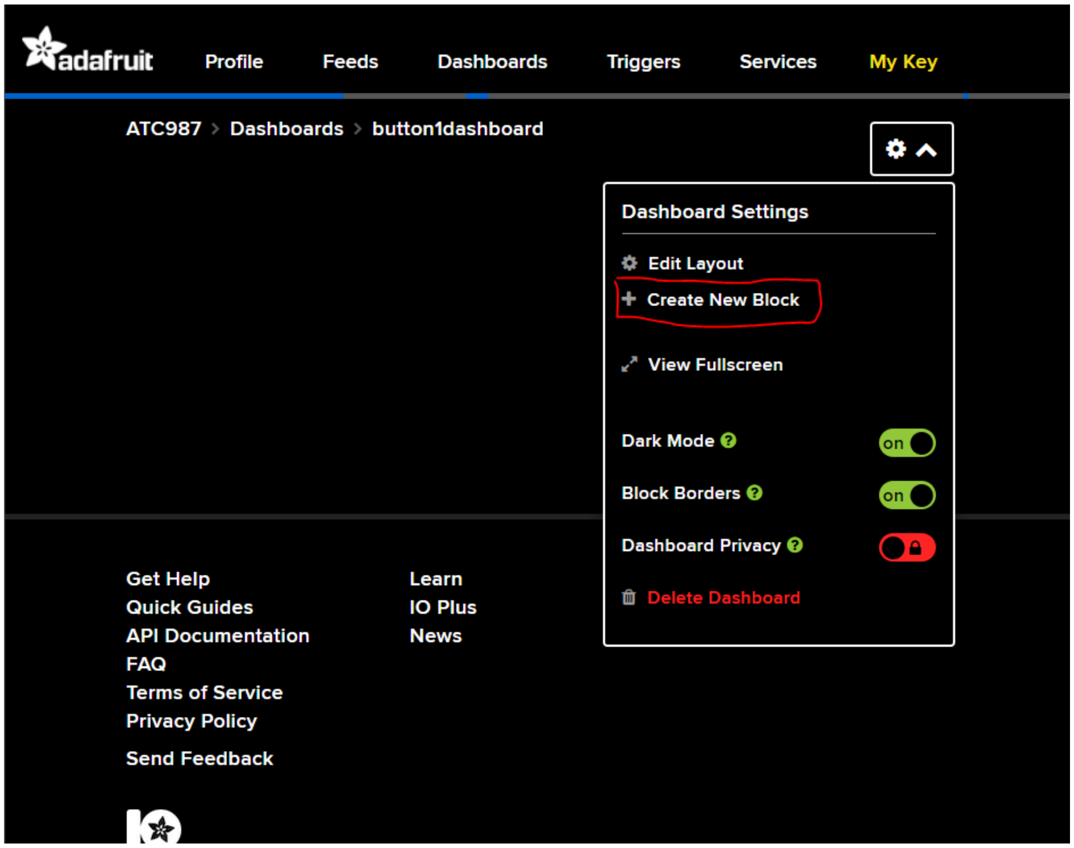
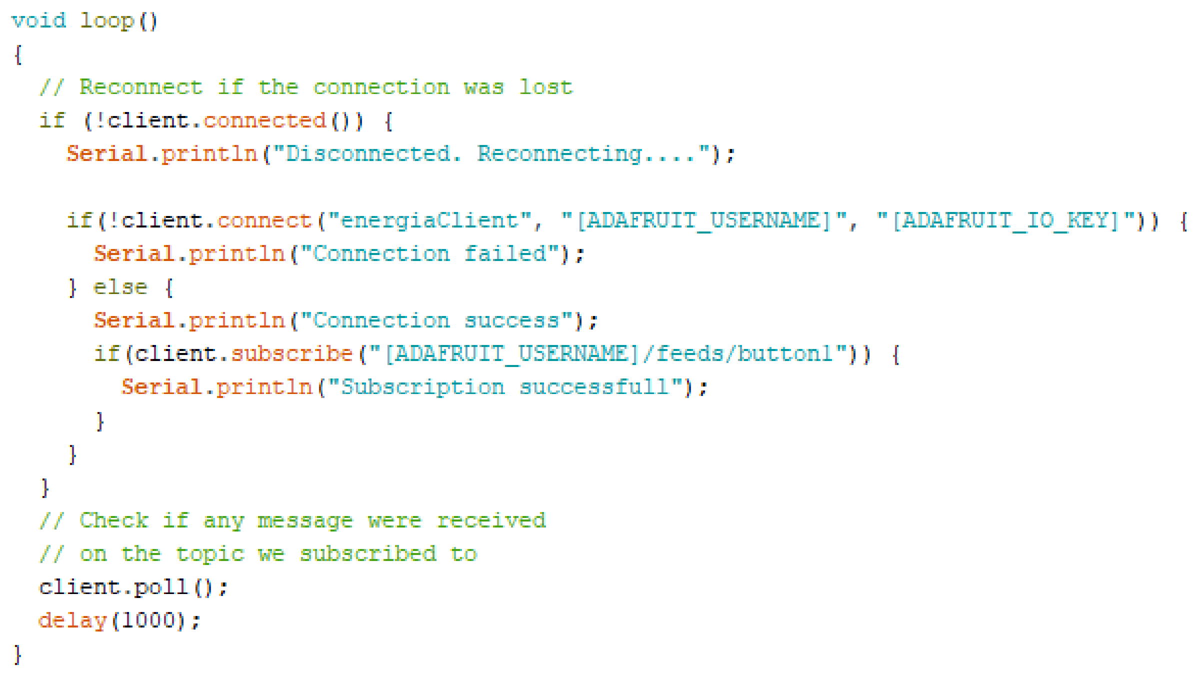


Figure 6: Creating a Toggle Block on Adafruit IO

4. You will be asked to link a feed to the toggle button. Select the feed you created in part 1, “button1.”

5. Leave everything else as default and click create block. Now your dashboard has an added toggle switch. You can use this switch to publish data directly to your “button1.” Test this functionality out yourself.

6. Now we want our LaunchPad to subscribe to the “button1” topic. Create a new sketch in Energia and copy and paste the code from the last part to get us started. Make the following changes to your loop function.



```
void loop()
{
  // Reconnect if the connection was lost
  if (!client.connected()) {
    Serial.println("Disconnected. Reconnecting....");

    if(!client.connect("energiaClient", "[ADAFRUIT_USERNAME]", "[ADAFRUIT_IO_KEY]")) {
      Serial.println("Connection failed");
    } else {
      Serial.println("Connection success");
      if(client.subscribe("[ADAFRUIT_USERNAME]/feeds/button1")) {
        Serial.println("Subscription successfull");
      }
    }
  }
  // Check if any message were received
  // on the topic we subscribed to
  client.poll();
  delay(1000);
}
```

Figure 7: Subscribing to an Adafruit IO feed

7. Run your Energia sketch. After toggling the switch on your Adafruit IO dashboard, you should see the data reflected on both the “button1” feed and your devices serial terminal!

*Note: If you are successfully subscribing to the feed but you do not receive messages back from your feed, ensure that the “client.poll()” function is present. This function is crucial for the callback function to execute.

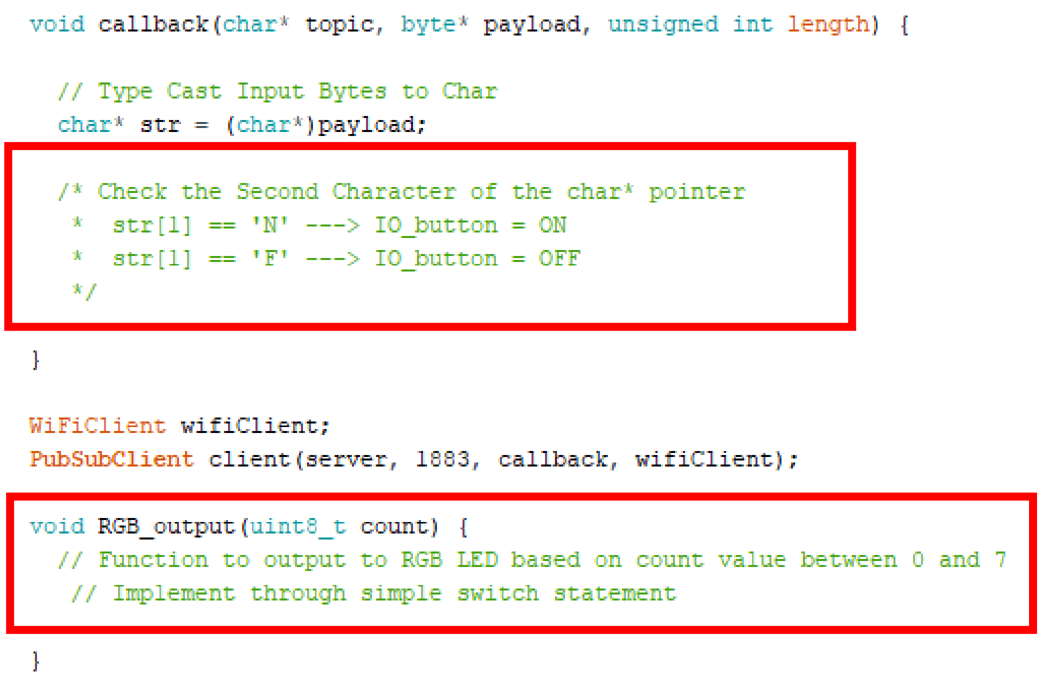
Implement a RGB Counter through the cloud:

In this final part of the lab, you will implement a cloud-based version of your binary counter from Lab 5 part 2. The RGB counter on your LaunchPad should increment each time either the SW1 on the LaunchPad or the toggle switch from Adafruit IO are pressed.

The approach for the binary counter from Lab 5 has an infinite loop that waits for the user to press the button. This approach will not work for this example because you also need to poll the user input from the cloud. Consequently, you must poll both the user switch and cloud switch simultaneously. Like Lab 5, the binary counter should increment when both cloud and mechanical buttons are released.

Design Guidelines

The first modification you must make for this lab is to implement the logic for the MQTT subscription callback function. Adafruit IO sends callback data in the form of an array of bytes (8-bits). Each byte in this array represents a specific ASCII character. To get you started, we convert the array of bytes to an array of chars using a method called "type casting." It is your job to implement the control logic for this function below. You should create a global variable that encodes the state of the IOT button so that your main loop knows its current state. Lastly, remember to include your RGB output function from Lab 5.



```
void callback(char* topic, byte* payload, unsigned int length) {

  // Type Cast Input Bytes to Char
  char* str = (char*)payload;

  /* Check the Second Character of the char* pointer
   *  str[1] == 'N' ---> IO_button = ON
   *  str[1] == 'F' ---> IO_button = OFF
   */

}

WiFiClient wifiClient;
PubSubClient client(server, 1883, callback, wifiClient);

void RGB_output(uint8_t count) {
  // Function to output to RGB LED based on count value between 0 and 7
   // Implement through simple switch statement

}
```

Figure 8: Callback Function

In the final part of this lab, you need to poll both the mechanical and IOT button simultaneously. For the first while loop, you should wait until either the mechanical or the IOT button are pressed. In contrast, the final loop should wait until both the mechanical and the IOT button are not pressed. Within each polling loop, you should ensure the connection to Adafruit IO and the "button1" subscription is active. The logic flowchart for the loop function is described in the figure to the right.

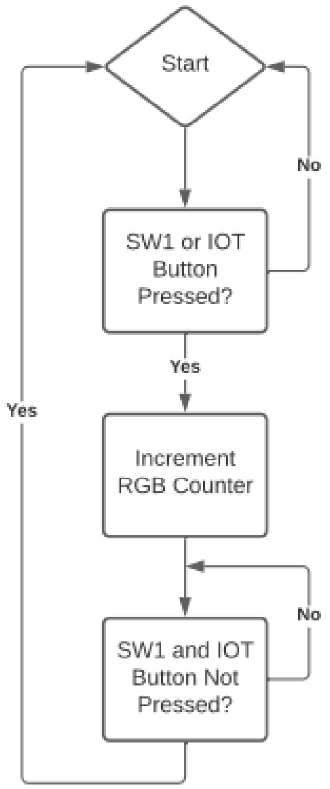
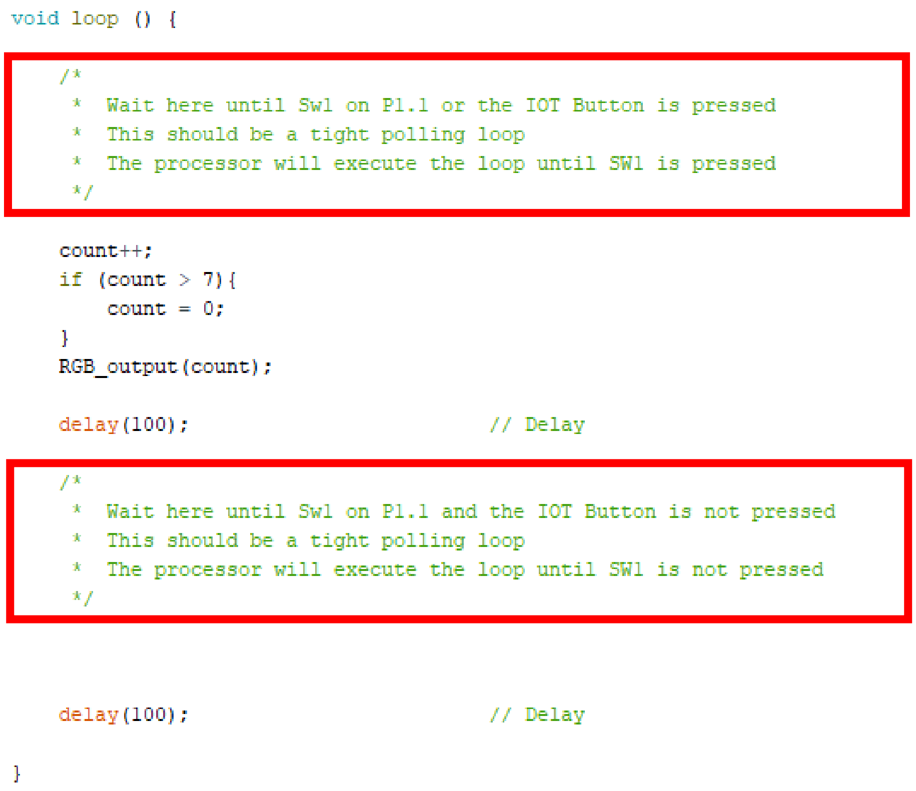


Figure 9: Main Loop Logic Diagram



```
void loop () {

    /*
     *  Wait here until Sw1 on P1.1 or the IOT Button is pressed
     *  This should be a tight polling loop
     *  The processor will execute the loop until SW1 is pressed
     */

    count++;
    if (count > 7){
        count = 0;
    }
    RGB_output(count);

    delay(100);                     // Delay

    /*
     *  Wait here until Sw1 on P1.1 and the IOT Button is not pressed
     *  This should be a tight polling loop
     *  The processor will execute the loop until SW1 is not pressed
     */


    delay(100);                     // Delay

}
```

Figure 10: Main Loop