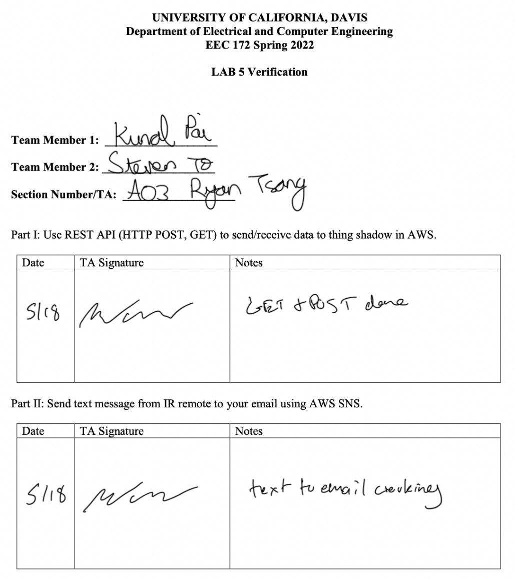
UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering
EEC 172 Spring 2022

LAB 5 Verification

Team Member 1: Kunal Pai
Team Member 2: Steven To
Section Number/TA: A03 Ryan Tsang

Part I: Use REST API (HTTP POST, GET) to send/receive data to thing shadow in AWS.

Date	TA Signature	Notes
5/18	[illegible]	GET & POST done

Part II: Send text message from IR remote to your email using AWS SNS.

Date	TA Signature	Notes
5/18	[illegible]	text to email working

Contribution:

For the first part of the lab, both partners worked on AWS and its setup together. Steven helped with setting up the protocols for the second part of the lab. Kunal worked on integration of Lab 3 into Lab 5, and both partners worked together on sending the post request.

Introduction:

The objective of this lab is to explore and utilize RESTful API to connect to Amazon Web Services to retrieve and send information. Not only will information about an AWS thing be obtained, code from lab 3 will be incorporated to compose and send messages to AWS. These messages can then be emailed to a user through the creation of AWS rules. Most of the code for connecting to AWS will be provided but some parts may require modification to meet the specifications of individual AWS accounts.

Background:

Amazon Web Services is a collection of cloud-based web services for businesses to use without the need to develop hardware infrastructure for their goals. One of the services that will be used in this lab is the Internet of Things which allows multiple hardware devices to be connected to one another. To represent these devices in the cloud, the notion of a device shadow is used to represent the real-life device and its state. This state can be updated using RESTful API calls and can trigger specific actions under certain rules.

RESTful API is a web-based communication protocol used for many services. Typically implemented using HTTP, the API defines the format needed to send a request to a service based on functions like GET (to receive data), POST (to push data), and DELETE (to manage or remove data).

SPI is a serial communication protocol that allows the transmission of data between a controller and one or more peripheral devices. It involves the use of four data lines: MOSI which allows the controller to transmit data to the peripheral, MISO which allows the peripheral to transmit data to the controller, SCLK which transmits the clock signal from controller to peripheral, and CS which allows the controller to select which peripheral to communicate with.

The OLED is a 1.5” display consisting of 128x128 RGB pixels, with each pixel having 16 bits of resolution to allow for high contrast and a wide range of vivid colors. It uses a SSD1351 driver chip that manages the display through SPI. Adafruit also provides the OLED with an API of functions that can be used to display patterns and text (Because it was originally written for Arduino, it needs to be ported to the CC3200 through the modification of low-level functions).

The AT&T S10-S3 Remote Control is a device that produces IR waves that can be picked up by the IR receiver module. These IR waves vary based on a 4-digit device code that can be configured along with channel, volume, and backlight settings.

The Vishay TSOP31336 is a device that can pick up IR signals and produce waveforms depending on the signals. For this to work, the IR receiver has to be configured on a breadboard with a resistor, a capacitor, a voltage source, and ground.

Uniflash is a tool used to overcome the downside of the loss of program data that occurs due to loading the program onto the RAM of the CC3200 board. It also makes the program run independently of Code Composer Studio, which increases portability.

Goals:

Part 1

- To set up an Amazon AWS account
- To obtain a basic understanding of AWS
- To set up a device thing/shadow using the AWS IoT Console
- To obtain the required certificates and keys to get AWS IoT to work
- To establish policies for the device thing to perform specific actions and attach these policies to required certificates
- To convert the keys and certificates for CC3200 to use
- To use Uniflash to flash the keys and certificates onto the CC3200
- To access AWS using RESTful API by modifying certain macro data in main.c and common.h

Part 2

- To create an SNS topic with a subscription that will email text messages
- To develop an IoT rule that will trigger upon a device shadow update and send a SNS email
- To integrate lab 3’s IR remote texting to compose text messages on the OLED and email these messages via AWS

Methodology:

Part 1

To get started with part 1, one of us had to set up an AWS account. From there, we went to the IoT Core module under the Internet of Things service to create a device thing. To do this, we clicked on *Manage -> Things* and created a single thing. We gave this thing a descriptive name and an unnamed shadow, skipping the creation of a certificate and leaving the *Types and Attributes* options blank.

Once the thing was created, we went to the *Device Shadow* tab and clicked on *classic shadow* for the device. Checking the shadow details, we saved information about the AWS endpoint which can be found in the device shadow URL.

After saving this information, we generated the required keys and certificates for IoT permissions by going to *Secure -> Certificates* and creating a new certificate using the *auto-generate* method and selecting *active* for the certificate status. From there, we saved the public and private keys and root certificates to a folder and activated the certificate.

To give our thing special privileges to perform specific actions, we navigated to *Secure -> Policies* and created a new policy with a descriptive name that allows us to get and update the thing's shadow by setting the policy resource as the thing's ARN. We then attached this policy to the previously created certificate by going to *Secure -> Certificates*, clicking on the desired certificate, and going to *Actions -> Attach policy* to select the policy to attach.

Because the private key and root certificates use the .pem format, we had to convert them to .der for the CC3200 to use. To perform this conversion, we used openSSL and provided a series of commands that perform the conversions. From there, we flashed the converted keys and certificates to the CC3200 under "User Files" using Uniflash. The files flashed were the root certificate, client certificate, and the private key.

Before we could access AWS using RESTful API, we had to adjust some defined macros in a few files. Since RESTful API requires Internet connectivity through an Access Point, we had to change the macros of common.h to match the configurations of the wifi network we used. Another change we needed to make was to adjust the date macros in main.c to match the current time so that the credentials can be verified as valid. Lastly, we had to change information about the AWS endpoint path and the thing name in the macros of main.c to reflect our device thing.

Once this was all set up, we tested out the GET and POST requests to ensure that the requests were being fulfilled properly and that information was being retrieved and updated accordingly.

Part 2

To get started with part 2, we created a new SNS topic by going to the Simple Notification Service module under the Application Integration service and creating a new topic. From there, we created a subscription with the protocol as *EMAIL* for text messages. We provided one of our email addresses as an endpoint to receive text messages and confirmed the subscription.

Next, we set up an IoT rule that allows a SNS email to be sent upon a triggered action. To do this, we went to *Act -> Rules* in the IoT module and created a new rule. We made this rule listen to the topic '$aws/things/*thingName*/shadow/update/accepted', report on the attribute 'state.desired', and leave the condition blank. To wrap things up, we attached an action that will send an SNS push notification when the rule is triggered, set the message format as 'RAW', and created an IAM role to allow the IoT to access the SNS service securely.

Once we enabled this rule, we pushed an update to our device shadow which resulted in an email notification. With the SNS set up, we integrated our code from lab 3 to send composed messages via email using AWS. Firstly, we initialized all the clocks and the peripheral for SPI communication. Then, we defined global strings of the input sequence for every button. For example, “abc” for button 2, “def” for button 3 and so on. After that, we changed the function that decodes the input. We added a variable that stores the previous input and another that marks the index of the input sequence string. If the previous input and the current input are the same, the index increases by 1, which outputs the next character in the input sequence. To prevent out of bounds errors, we used a modulo arithmetic function so that it remains within the bounds of the string sequence. For the colors, we created a 2D array of unsigned integers using the color macros defined in the Adafruit files, and cycle through them when 1 is pressed, similar to the logic behind the button presses. The current color is stored in another variable and applied whenever printing of the character is supposed to happen. If a zero is pressed, since it is a space, we just concatenated a whitespace to the resultant string, and for delete, we moved back an index and changed the character to a null escape sequence (“\0”). We also have a for loop running through, and if another button is not pressed within it, the character is locked in and the user can enter the next character. Instead of having the MUTE key send the messages to a remote board using UART, we allowed it to make the necessary preparations before making a post request using the composed message as data to be sent as an email.

Discussion:

The main difficulty of this lab was the inability for wireless LAN to connect before using RESTful API. Because it was unable to connect, our program ended up hanging due to an infinite loop. The fix to this issue was changing the wifi network main.c was connected to as defined by the header macros in “common.h”. Rather than connecting to the lab wifi, we connected to an open wifi hotspot we had set up which allowed wireless LAN to connect

Conclusion:

In this lab, we were exposed to some of the many features that come with AWS. Through step-by-step instructions, we learned how to set up what was necessary for the lab, from setting up the device and its shadow to assigning it with rules that will trigger actions based on certain requests. We also learned about the importance of a valid AP connection, proper date and time, and correct formatting to ensure that the requests made using RESTful API are satisfied appropriately. Lastly, we learned how to slightly modify our code from lab 3 to utilize RESTful API to meet the specifications of the lab 5.