EEC 172 Lab 6: Open-Ended IoT Project I EECSmart - Smart Toaster System I Section: A03

Manprit Heer
912839204
mkheer@ucdavis.edu

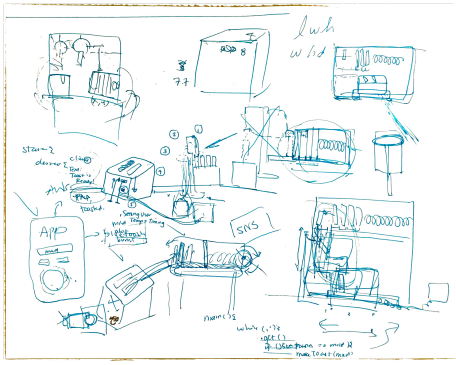
Cathy Hsieh
912753571
cchsieh@ucdavis.edu

OBJECTIVE

The objective of this lab is for us to use the REST API from Lab 5 to a web service to do something interesting and creative. We will use any equipment from the previous labs.

INTRODUCTION

For this lab, my partner and I decided to take it one step up in the creativity scale and create (from scratch) a bread slice-dispensing system that could essentially be utilized in a smart-home setting. In order to achieve this, we first created a sketch of what we wanted our dispenser to look like, and how we would implement the system. During this stage, we listed out the materials and methods that would be required to make the system function the way we want it to.



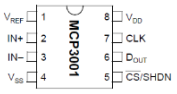
LAB VI | DELIBERATION AND BLUEPRINTING

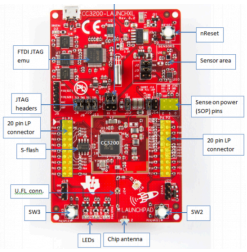
After much discussion and debate, we decided to create a conveyer belt system that would be controlled by the CC3200 Launchpad. With a timer, we are able to control the distance the belt is able to roll forward. Every time the system is triggered, one slice will be dispensed down a slide into the toaster. At first we thought it would be cool to have a spring that would push one slice at a time as it hopped one rung over at a time (controlled by a motor). But then we realized that it would require much time and labor to debug a spring's force and its timing, as well as to controlling the slice's motion from the abrupt push. We decided to step away from this idea and then approached the conveyer belt option with much hesitance due to the period of time we had left to complete the assignment. After much discussion, we realized it would be very fun to do and decided to go for it. While solidifying the sketch, we decided we needed the following items:

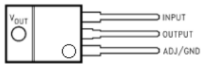
MATERIALS AND METHODS

The equipment needed for this lab are listed as follows:

1. CC3200 LaunchPad (CC3200-LAUNCHXL)
2. USB Micro-B plug to USB-A plug Cable
3. Mobile Device & App with dual-tone multi-frequency
4. LM1086CT-3.3 Low Dropout Voltage Regulator
5. MAX9814 Adafruit Electret Microphone Amplifier
6. MCP3001 A/D Converter (Microchip)
7. Amazon Web Services Account
8. LMioEtool DC Gear Motor 12V/20RPM
9. 3/4" wood screws
10. 2" wood screws
11. 1" multipurpose screws
12. Adafruit OLED Breakout Board[1]
13. Saleae USB Logic Analyzer
14. ULN2003A Darlington Transistor (Driver)
15. Saw/Screw Drill
16. Clamps
17. Ruler
18. Exact-O-Knife
19. Screwdriver/Tool Kit
20. Safety Goggles
21. Hot Glue Gun/Gorilla Glue







[1] https://www.adafruit.com/product/684?gclid=Cj0KCQiAsbrxBRDpARIsAAnnz_PATckp_fWf0mom4VPLEF7osaKOk78D4VWOKufqBx_EhoA3Zs9L6vQaAkC5EALw_wcB

PROCEDURE

Part I. Hardware Installation

Creating the Base

We used a wooden plank as the base of the system, and acrylic glass sheets for the sides. It was created this way so it is easy to see what is going on inside of the system, as well as debugging purposes. We started off by sawing each of the acrylic planks in half and using these halves to create a rectangular box. Then we super-glued and hot-glued each of the sides together. To give some height to the system, we glued four wooden planks to the system after measuring the right amount that would allow for the system to slide the bread down to the toaster without having the bread tumble down due to the slope. After figuring out these rough calculations, the wooden planks came on and allowed the system to stand. In order to secure the planks properly, we decide to use clamps to hold the glue and components together for maximum force and secure adhesion.

LAB VI | CLAMPING WOOD WITH ACRYLIC SHEET

Compiling the Conveyer Belt

Creating the conveyer belt was a difficult process. For one, we did not know what material would be a good and effective substitute for the heavy-duty rubber belt material, which would have been costly for this operation. We traveled to couple different stores, and got some advice. We were able to finally settle by making our own belt, one that would not be too malleable, nor stiff. To do this, we purchased poster paper and black, industrial tape. After cutting the poster paper to the right size of our belt, we tape one side of the belt with tape so that it causes enough friction to keep the belt in place when the motor turns it so that it does not shift and slip excessively in the system.

In order to create the rotating tubes for the belt system, we decided to buy PVC pipes and use those as the main rotators. Due to the long size of the pipes, we had to use a saw machine to cut the pipes accordingly and we continued to use proper measurements in the process for it to fit the system in our diagram.

One problem with the conveyer belt included the fact that it did not have enough tension to smoothen the cycle around the tubes. To fix this, we tightened the belt, which was then able to straighten the belt and make the process more smooth.

LAB VI | RESIZING PVC PIPES + SANDING FOR SMOOTHNESS

Motor Installation and Troubleshooting

The first struggle of installing a motor we had never worked with included the lack of research we did for our first purchase. We had bought a toy fan motor kit on Amazon, and it did not have nearly enough power to power the system we were trying to build. We then do some research to find a stronger motor, one that is 12V. This was sufficient for our purposes, and after many attempts, we realized attaching this particular motor to our system would be harder than we thought. We first tried to attach it directly to the pipe, but the wooden connecter had given up after a couple rotations and all of the hot glue came un-done.

After many smaller attempts, we decide to utilize a couple different tools that could solve our slipping problem. This included the use of tightening clamps and rubber tubes, those that shrink upon applying heat. In order to attach the motor, we had to use a piece of wood and nails to clamp down the motor and attached this whole subsystem to one of the wooden legs using an L-hook. One issue we encountered here was the question of connecting the motor directly to the pipe while making sure it would turn freely to do a perfect, uninterrupted 360-degree revolution. To fix this, we used our saw drill to create a long

LAB VI | CREATING HOLES FOR NAILS AND MOTOR INSTALLATION

entryway for the motor-pipe connection to slide in. This change also helped with debugging and quick removing purposes. These tools, along with the magic of hot glue, allowed for us to finally connect the motor and, when we connected the belt and tightened it for tension, allowed for the system to run smoothly.

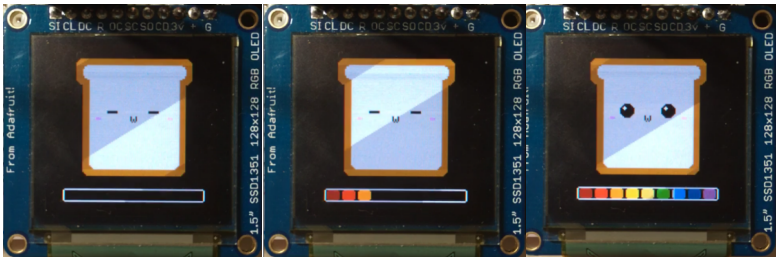
We then install the sliding compartment of the system, which would then allow for the bread to slide down into the toaster system. This was easy, as all we had to do was grab one of the plastic sheets we had purchased and cut the edges to create a restricting path for the bread slice to follow and fall off of. The constraints made from this sheet was to ensure that the slice doesn't fall outside of the system when it free-fell off of the conveyer belt.

LAB VI | INSTALLING THE SLICE PATH - FROM SYSTEM TO TOASTER

Part II. Software Components

Establishing the Logic Flow

In terms of logic flow, we were able to write a C program that combines the different components we have worked with over the past couple labs. We implemented the dual-tone frequency system that acts as a trigger for the system. We also send an SNS text to the subscribed phone number when the process is ready, which indicates that the toast is ready. The OLED system is also incorporated by creating a progress bar (with a fun toast cartoon) that helps the user see the length of time left until their toast is ready. In order to ensure a secure logic flow, our code makes sure that the functions are called in the correct order.



LAB VI | DYNAMIC TOASTER PROGRESS ON OLED

Amazon Web Services

We implemented a front-end dynamic iOS application that is registered to a topic on our IoT Core MQTT system. We tried to publish from the app to AWS, but there was simple not enough time and documentation to make this possible. So instead, we were able to send a message from the web service to the app, which then prints out if the toast is ready based on the change in state.

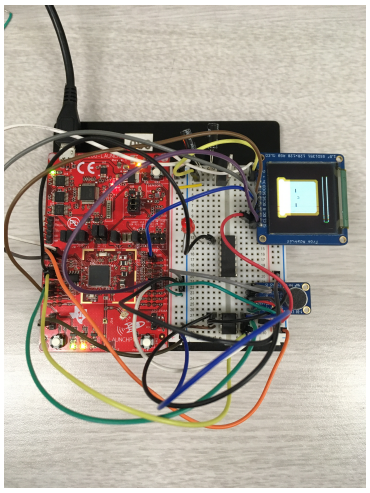
LAB VI | IOS APP, CONNECTED TO IOT & REGISTERED TO TOPIC

Wi-Fi and Authentication

The secure connection for iOS required a couple of different processes. As we already had our certificate/keys in place from the previous lab, we have to convert the files with OpenSSL into a .p12 file compatible with the iOS protocol. After this conversion we were required to use other services on AWS to authenticate users and create policies for the IAM Users. After giving permission for "FullAccessIoTCore" to the authenticated AND unauthenticated users, this allows for the AWS to be able to publish and register to topics created for the application. AWS's Identity Pool ID string was then inserted to initiate a data configuration with AWS's MQTT protocol. With the cocoa pod "AWSIoT", we were then able to establish a connection with AWS with the library provided.

```
class ViewController: UIViewController {
    var JSONPayload = "{\"state\":{\"desired\":{\"newpost\": \"published from Device\"}}}"

    var JSONPayload2 = "{\"message\": \"published from Device\"}"

   // Initialising AWS IoT And IoT DataManager

    @IBOutlet var mqttLabel: UILabel!

    override func viewDidLoad() {
        getAWSClientID { (clientId, nil) in

        }
        AWSIoT.register(with: configuration!, forKey: "kAWSIoT")  // Same configuration var as above
        let iotEndPoint = AWSEndpoint(urlString: "wss://a3s6tp2bmfdnpi-ats.iot.us-west-2.amazonaws.com/mqtt") // Access from AWS IoT Core
            --> Settings
        let iotDataConfiguration = AWSServiceConfiguration(region: .USWest2,     // Use AWS typedef .Region
                                                           endpoint: iotEndPoint,
                                                           credentialsProvider: credentials)  // credentials is the same var as created
                                                              above

        AWSIoTDataManager.register(with: iotDataConfiguration!, forKey: "kDataManager")

        super.viewDidLoad()

        // Do any additional setup after loading the view, typically from a nib.
        connectToAWSIoT(clientId: clientID )


        publishMessage(message: JSONPayload2, topic: "myTopic/3")
        dataManager.publishString(JSONPayload, onTopic: "$aws/things/CC3200_Thing/shadow/update", qoS:.messageDeliveryAttemptedAtLeastOnce)
        let gotshadow =  dataManager.updateShadow("CC3200_Thing", jsonString: JSONPayload)
        print("Shadow Status: ")
        print(gotshadow)
        dataManager.updateShadow("CC3200_Thing", jsonString: JSONPayload) //new
        dataManager.getShadow("aws:iot:us-west-2:185005176321:thing/CC3200_Thing")

    }
```

LAB VI | IOS APP CONNECTIONS



LAB VI | LAUNCHPAD SYSTEM

SUMMARY

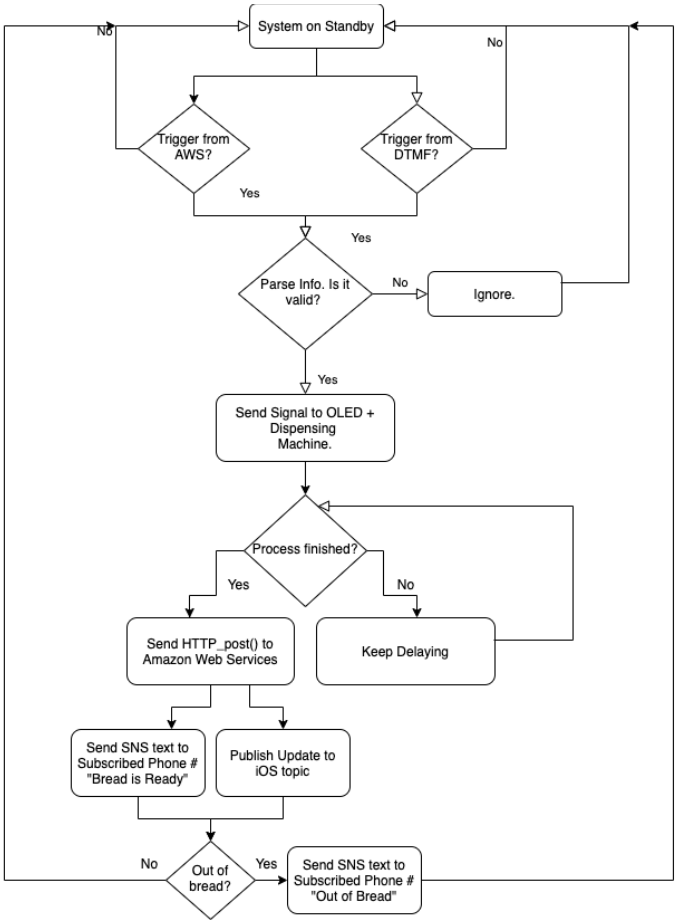
After working on this lab for countless days and nights, we were able to make the bread-slice dispenser work smoothly and efficiently. One aspect we were not aware of until implementation was the amount of bugs and errors that could be found throughout our hardware setup. For example, we did not realize the importance of the precision of the motor placement, as well as rigid structure that binds the motor with the PVC pipe. This took the majority of our time, but was quite rewarding after the countless troubleshooting strategies we were able to come up with.

After this lab, it was simple to see how quickly we are able to think of and implement embedded systems such as our own. With the right amount of electrical and coding knowledge, anyone is able to put two parts together and control them through the internet. After this lab, we were able to connect Amazon Web Services with various physical devices, which helped us realize the unlimited power that comes with embedded systems. Troubleshooting in this lab, as well as the many others, also established a workflow for the way we start, as well as the multiple instances of design change and debugging, until we were able to get to the final product. It is definitely not a linear path to a working system, but it is surely a rewarding one.

GPIO Pin #	Use
7	MOSI (SI)
5	SCK (CL)
58	DC
18	RESET (R)
62	OLEDCS (OC)
57	UART RX
55	UART TX
61	Motor
4	Open
3	CS for ADC

LAB VI | CC3200 PIN CONFIGURATION

LAB VI | HARDWARE SYSTEM



LAB VI | STATE DIAGRAM