TI CC3200 Launchpad Wireless Image Reciver

Cole Smith
EEC 172A
University of California - Davis
Davis CA, United States
coles427@gmail.com

***Abstract*—This document outlines a process for building a wireless image receiving program on TI CC3200 launchpad. The projects builds on labs completed in EEC 172A, and makes use of I2C, SPI, UART, and AWS. The CC3200 pulls an image from AWS and displays it on an external OLED screen. Being able to retrieve an image on a remote server and display it on an external display is a useful and common task that is commonly used for displaying advertisements. This project is a dive into recreating how these systems work.**

Keywords—component, formatting, style, styling, insert (*key words****)***

I. Introduction

An embedded system is a computer system that contains a processor memory an input output device. They often use a low power, low performance processing core, and are at the hart of many Internet of Thing (IoT) devices. This project uses a TI CC3200 Launchpad embedded system to receive images hosted on a remote Amazon Web Services (AWS) server. The project builds on topics learned in EEC 172A and combines elements used in previous labs. To achieve the goal of displaying a remote image several protocols and systems are used. An android app is used to process and upload the image to AWS via a Message Queuing Telemetry Transport (MQTT) topic. On AWS the image is stored in a IoT shadow. On power up the CC3200 pulls the latest shadow update from AWS, which contains the image. The CC3200 then, sends the image data to the OLED via SPI where it is displayed. From there the CC3200 waits for input, if a switch is pressed it will receive the latest image from AWS. It also receives accelerometer data via I2C, if it is titled past a certain threshold the image is rotated on redrawn.

II. Image Procsseing and Upload

Most images are not suitable to be displayed on the SSD1351 OLED displayed used by this project. The maximum resolution of the display is only 128px * 128px and it uses 16-bit color. For images to display correctly on the OLED the must first be processed before being downloaded to the CC3200.

A. Image Selection

Images are selected through a simple android app. The app builds on the "*PubSubAvticity"* app included in the AWS android example GitHub repo[1]. A floating button is first added the activity_main.xml. The onClick() function of the button is then set to a new function that invokes an Android photo picker Intent. This opens the default gallery app that is set in Android. The user selects the desired image, and then the program returns to the main program with the Uniform Resource Identifier (URI)

B. Image Cropping

In the main java file for the app, the *"onActivityReturn"* function is overloaded, this detects when the app returns from the gallery. The app then invokes the Image cropper intent with the selected image URI. Using the image cropper library from the *Android-Image-Cropper* GitHub the user selects an area of the image to crop to. Once the selection is confirmed the library resizes the image to 32px*32px and stores the result in a *"Bitmap"* object.

C. Image Processing

First, we need to extract the data for each pixel from the image, this is done using the Android *"copyPixelsToBuffer()"* function. This returns a byteArray containing the color data of each pixel, but it is represented by a 24-bit color. Before uploading the image is converted to 16-bit using the following equation taken from a blog post by Lucas Galand[2]:

$$\begin{aligned} R5 &= (R8 * 249 + 1014) >> 11; \\ G6 &= (G8 * 253 + 505) >> 10; \\ B5 &= (B8 * 249 + 1014) >> 11; \end{aligned} \quad (1)$$

Each 3-byte pixel value is converted into a 2-byte value using (1) and the results are stored in a new byte array. The byte array is then encoded as a base64 string using Android's *"encodeToString()"* function.

D. Upload

Once encoded the image is ready to uploaded to an AWS IoT shadow. The *"PubSubActivity"* is updated so that the host points to my instance on AWS, and same with the region. An IAM role is setup so that the app can connect to AWS Cognito, and download the required certificates. The image is finally pushed to AWS using MQTT via the "$aws/things/CC3200_Thing/shadow/update" update topic.

III. Image Download and Draw on OLED

The CC3200 can retrieve the image stored in the AWS shadow via a HTTP GET request. Once the image is stored locally on the CC3200 it is decoded and then finally drawn on the OLED pixel by pixel

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

A. *Recive Image*

The skeleton code from EEC 172A Lab 4 [3] is used as the base for the image receive code. It is edited to point to the correct host and shadow. The time is updated to the current date so that authentication works correctly. A new function is created for HTTP GET, as is a new HTTP GET header using the HTTP POST equivalents as a template. The HTTP GET function is edited to have the data section HTTP packet removed since it sends no data on a GET. The GET header is edited to change "POST" to "GET" and the rest remains unchanged. Now on startup the CC3200 connects to Wi-Fi and then opens a TLS socket and connects to AWS using the certificates generated in Lab 4 [3]. A HTTP packet is created containing our GET request, that is then sent to AWS. AWS responds by sending the message contained in the IoT shadow. The response is saved to a receive buffer. From the receive buffer the base64 is parsed and stored as a string

B. *Decoding*

To retrieve the image the base64 string received from AWS first needs to be decoded. This is done using base64 library provided by Apple Computer, INC. [4]. A empty sting and the encoded string are passed to the *"base64decode"* function. The functions returns the decoded string in the empty string.

C. *Reconstruction*

Now we have the raw data that makes up the image, it just needs to be formatted properly. The dimensions are not stored in the data therefore it is assumed they retain the 32px * 32px defined earlier in the paper. Therefore, we create a 32*32 matrix to store the image. The string is then read, and each charter is stored in the matrix. Each row represents a 32px row of pixels and the same for columns. Once finished each entry in the matrix stores the color value for its corresponding pixel.

D. *Drawing the Image*

The image matrix is now ready to be drawn on the OLED. The image is sent using SPI commands from the adafruit OLED library [5] provided in lab 2, we use the *"drawPixel"* function. The image could be drawn as is, but it would not take up the full screen so, I chose to map each pixel in the matrix to 4 pixels on the OLED. We iterate through the matrix and draw each element 4 time. Once at x,y , again at x+1,y, again at x,y+1 and finally at x+1,y+1. This draws a full 128px*128px image on the OLED.

Figure 1. Image Drawn on the SSD1351 OLED

IV. Additonal Features

A. *Image Rotation*

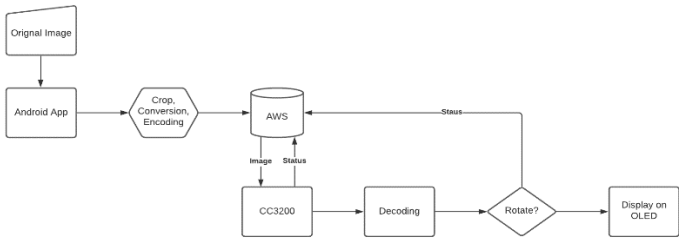
After an image is drawn on the OLED the program returns to the main loop and waits for input. The CC3200's accelerator is polled during each loop via I2C using the code from lab 2 [6]. If the board is tilted past a threshold in either the positive or negative direction along the x-axis or y-axis then the image is rotated in the corresponding direction. The actual rotation of the image is achieved by changing the indices of the image matrix while drawing. For 0°: [x][y], for 90° [31-y][31-x], for 180° [31-x][31-y], and for 270°[y][x].

B. *Status Tracking*

To track the status of the board, an additional shadow was created "resp". Using the code from lab 4 [3], the HTTP POST header is updated to point to the new shadow and several new messages are defined. The main loop is modified such that when any of the following events: switch 3 is pressed and new image is received, image is rotated 0°, image is rotated 90°, image is rotated 180°, or image is rotated 270°, a HTTP POST is sent to AWS containing the corresponding message. The AWS shadow is also updated so that when the "resp" shadow is updated it sends an SNS message through AWS.

V. *Results*

A demonstration video is located at: https://drive.google.com/file/d/1ww4oGNtq2eBeHDaAZ8hT2UGexcbSgnbd/view?usp=sharing. The project works as expected, I was able to successfully upload and receive and image. Draw said image on screen, and then send and receive the updates. The only thing I would like to change is the ability to send a bigger image. Currently it is limited to 32px*32px due to the AWS IoT shadow message size limit. Using another method to send the image it would be possible to send an image at the screens native 128px*128px resolution. Below is a high level block diagram of the project.



References

[1] AWS SDK for Android Samples, 2014, https://github.com/awslabs/aws-sdk-android-samples

[2] Lucas Galand, 24-bit / 16-bit color converter tool for embedded LCD GUIs, 2015, https://www.lucasgaland.com/24-bit-16-bit-color-converter-tool-for-embedded-lcd-guis/

[3] Lab4, SSL_REST_API_AWS.zip, 2021 https://canvas.ucdavis.edu/files/11716054/download?download_frd=1

[4] Apple Computer, INC., base64.c, 2003, https://opensource.apple.com/source/QuickTimeStreamingServer/QuickTimeStreamingServer-452/CommonUtilitiesLib/base64.c

[5] Lab 2, Lab2_starter_files.zip, https://canvas.ucdavis.edu/files/11364975/download?download_frd=1

[6] Lab 3, I2C_IF.c, https://canvas.ucdavis.edu/files/11472499/download?download_frd=1