**Lab 5 Report Alex Elkman**

**EEC 172 ­ A02 ­ Spring 2016 Hardik Patel**

**Home Security Camera Monitor with Email Notifications**

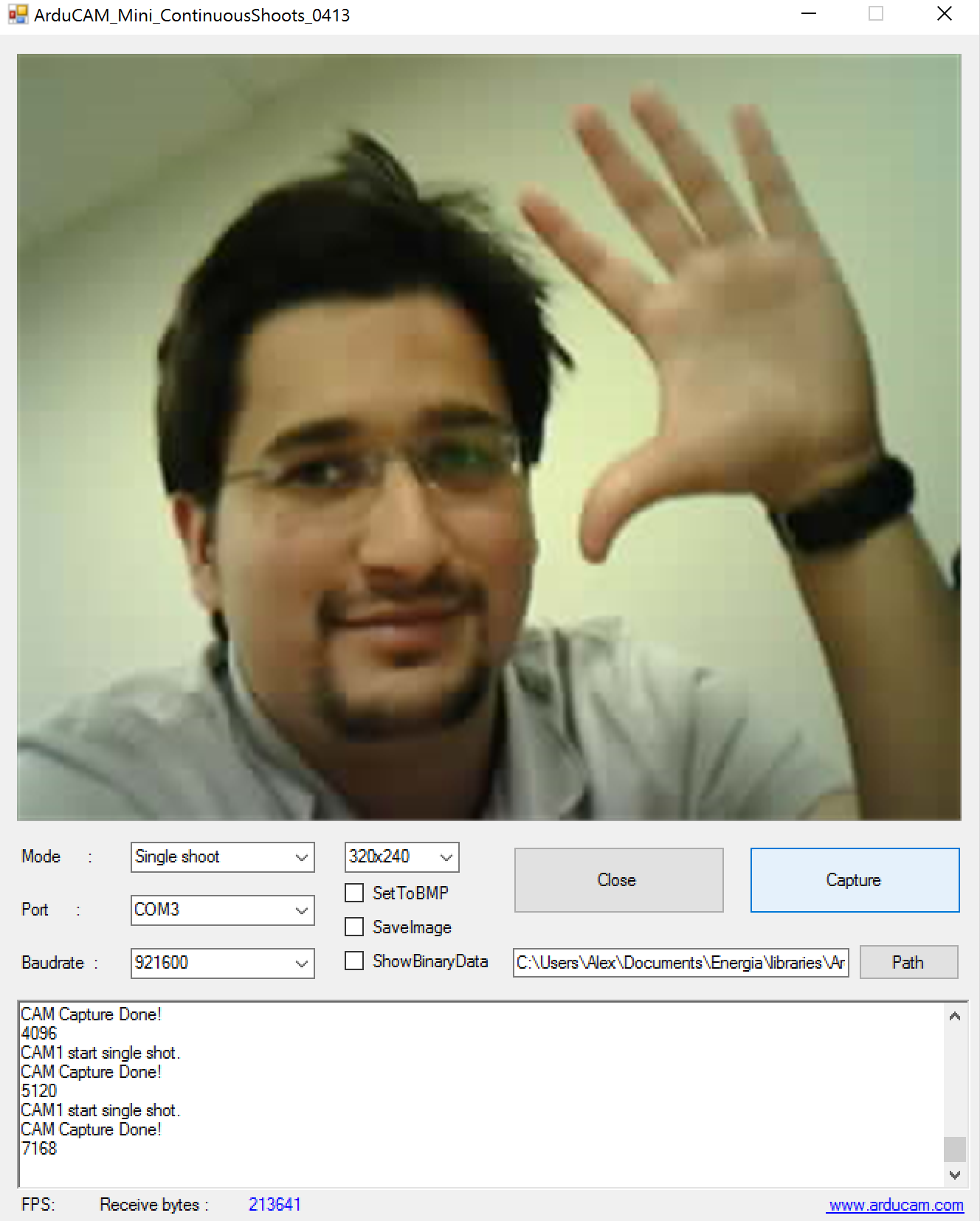
**Objectives**

The purpose of this lab was to use REST API or MQTT to securely connect to a web service and create a practically usable system. Our objective was a home security system, in which we used a camera connected to a an Arduino UNO to interface via I2C with the CC3200 board. The camera takes a picture every few seconds and if a change is detected, it sends a picture of the change over the internet to an email. We used IFTTT (RESTful API) through the CC3200 to send the picture and base64 encoding of the picture to send as the data. We then decoded the Base64 message in the mail through an embedded HTML picture.



Figure 1: Block diagram of security system with web alerts

**Design and Test Procedures**

**Part 1:** *Setting up the camera:* The OV2640 ArduCAM mini comes functional for SPI and I2C interfacing, so either option was viable for use with the CC3200. However, due to the SPI interfacing issues between the CC3200 and the camera, we opted to use the I2C route of using the CC3200 as a slave connected to an Arduino UNO board which interfaced directly with the camera. This facilitated the process of getting the camera working. The camera interfaces via I2C as well as SPI with the Arduino board, using pre made library functions that can be downloaded at the ArduCAM website. We checked the camera with the ArduCAM Host application provided at the same location, which worked flawlessly and captured images onto our computer via Serial ports.

**Part 2:** *Setting up the Arduino I2C connection:* We needed to establish a connection between the CC3200 and Arduino UNO in order to transmit the data being stored on the Arduino’s memory over to the CC3200, whereby we could connect to the IFTTT server and relay the JSON message (Bit64 encoded) to our email. The goal here was to send the JSON message with an HTML embedded picture wrapper, which will also decode the Base64 encoded message (HTML has the feature pre-built in). We opted for using an I2C connection between the boards, which may have been one of our pitfalls (will be discussed later in the “Problems” section), so we connected the boards together using 4.5KOhm resistors and a 3.3V source from the CC3200 board. We then modified the “OV2640 Low Power SPI Burst Mode” Arduino sketch which ran the camera to feed the burst data from the Camera’s SPI interface into the Arduino, and then through the I2C connection to the CC3200 as a slave. This was achieved by using the pre-defined “write” function from Wire library in the Arduino IDE which is for sending and receiving I2C data. Inside the SPI interfacing code in the sketch, we initialized this master writing to slave:

while ( (temp != 0xD9) | (temp\_last != 0xFF))

{

//Serial.println(count);

temp\_last = temp;

temp = SPI.transfer(0x00);

//Write image data to buffer if not full

if (i < 256)

buf[i++] = temp;

else

{

//Write 256 bytes image data to file

myCAM.CS\_HIGH();

//outFile.write(buf, 256); //original code used SD card

Wire.beginTransmission(0x5);

for(j=0; j<256; j++){

if(j==64 || j==128 || j==192) //I2C buffer defined to 64 bytes

{ //must process in 64B chunks

Wire.endTransmission(); //hence the end and begin transm

Wire.beginTransmission(0x5);

Wire.write(buf[j]);

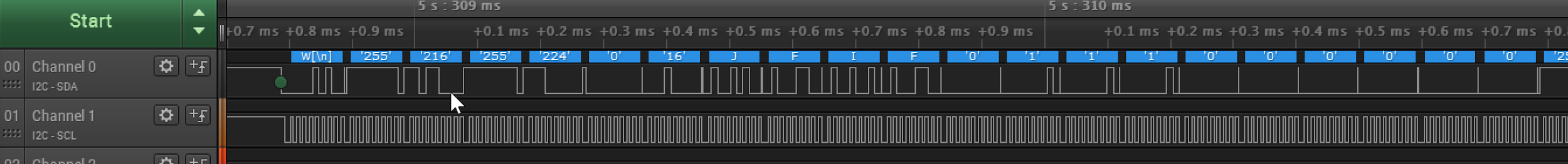
//count+=1;

}

else{

//count2+=1;

Wire.write(buf[j]); }}}}

Figure 2: I2C Master Signal from Arduino UNO showing the header data from the JPEG image

**Part 3:** *Setting up the IFTTT connection:* Using the “Maker” tool in the IFTTT, we create our thing and get access to a key. Using the key and the server name, the WiFi name and password, we used an energia IDE that was specifically made for CC3200 to send an email to a provided email, from another email, every time the SW2 button was pressed. This was meant to be changed to polling the camera data, and if a change was seen in that data, when compared to the last image, an email would’ve been sent with the email. Since, we weren’t able to get our image data, we couldn’t accomplish the last part. However, we were able to send emails via IFTTT. If we were to hard code an image into the code, it was able to send that via IFTTT to the email.

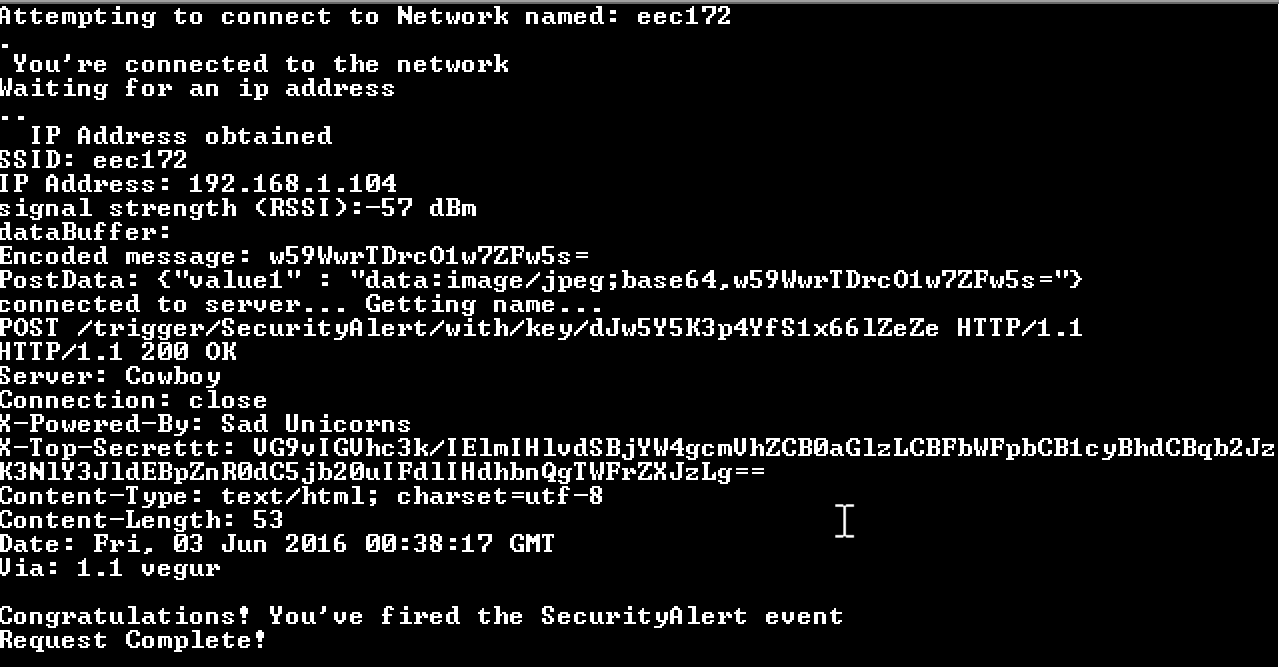


Figure 3: Output from CC3200 with IFTTT connection initialization and JSON message

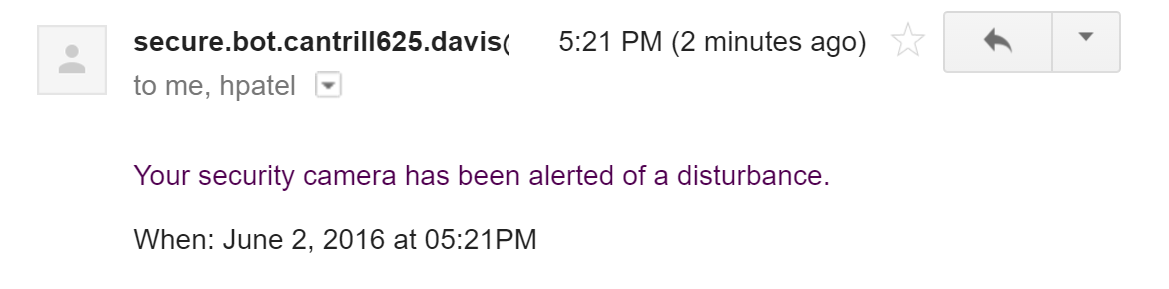


Figure 4: Email message received via IFTTT from dummy email account

**Part 4:** *Reading the data from the I2C connection into the CC3200:* The next challenge was to read the data in from the I2C line into the CC3200 board which functioned as a slave. The first challenge was making sure we assigned the CC3200 a unique slave identifier address to the CC3200, since the camera was also functioning as a slave to the Arduino UNO. This was achieved by using the CC3200 pre-defined I2C libraries and properly initializing the connection to utilize an interrupt handler. The interrupt handler’s purpose was singular- to receive and get the data from the I2C line if it was meant for the slave. This was used by a simple blocking command:

data = I2CSlaveDataGet(I2CA0\_BASE);

We then parsed the data into a messageBuffer, which was fed into the Base64 encoder function. The libraries for C++ Base64 encoding were found online and credit goes to it’s creator, show in the reference section. The encoder spits out the decoded image into the JSON string which gets sent with the HTML wrapper through the email server. This way the user would be alerted any time there is a disturbance in their home, or perhaps as a pet or child monitor so they can get constant updates if there is movement going on in their home environment.

**Part 5:** *Image comparison algorithm:* The IFTTT “Security Alert” was to be triggered only when the new image, taken in 5 second intervals, is significantly different than the older image. This was to be achieved by doing a simple averaging function of the binary image data and comparing the prior image’s data to the current’s, and only send the alert if the differences between the two have passed a certain threshold percentage value. Unfortunately, we weren’t able to get to this part of the project because of issues with the I2C connection fully working, and as such the picture data was not fully sent over from the UNO to the CC3200. In addition, we wanted to implement an algorithm that uses the newest picture as the base image, if it didn’t trigger the alter. In this way, we allow the camera to adapt to changing lighting conditions in the environment over slow, prolonged times (like the sun rise and fall).

**Problems:**

The major problems we encountered were with the I2C connection between the CC3200 and the UNO. The issue, we believe, was to do with a difference in the SDA and SCK value voltages coming out of the two devices, which caused an impossible state for choosing pull-up resistors. The UNO uses a 5V signal for both the signals, while the CC3200 needs 3.3V. We found out at the last minute that we could use a bidirectional level converter for this exact problem, at which point in time it was unfortunately too late to remedy the issue. We regret that we were not able to get a finished product ready in time by the deadline the project was due. The transfer would usually get about 500 bytes into the picture and then freeze both the master and slave. Even more interestingly, if we disconnected the master and then reconnected it, we encountered the rest of the missing data appearing on the slave’s terminal, indicating there there was some sort of interrupt error causing a freeze with the data, or perhaps due to inconsistent voltages causing issues.

In addition, we spent a lot of time first trying to interface the CC3200 directly with the ArduCAM which proved fruitless. The issue we encountered was that once the SPI signal was correctly established and sent to the camera, the signal we got back when trying to echo was incorrect. The data the CC3200 received was unique and dependent on the DATA field we went in, however, it was not the correct data at all. Even though the logic analyzer data looked correct for the Read command (see picture below), the data appearing on the terminal from the echo was not correct. We believe this may have been a hardware compatibility issue, but we couldn’t identify exactly the cause of the problem and moved on to using the UNO to interface with the camera instead.

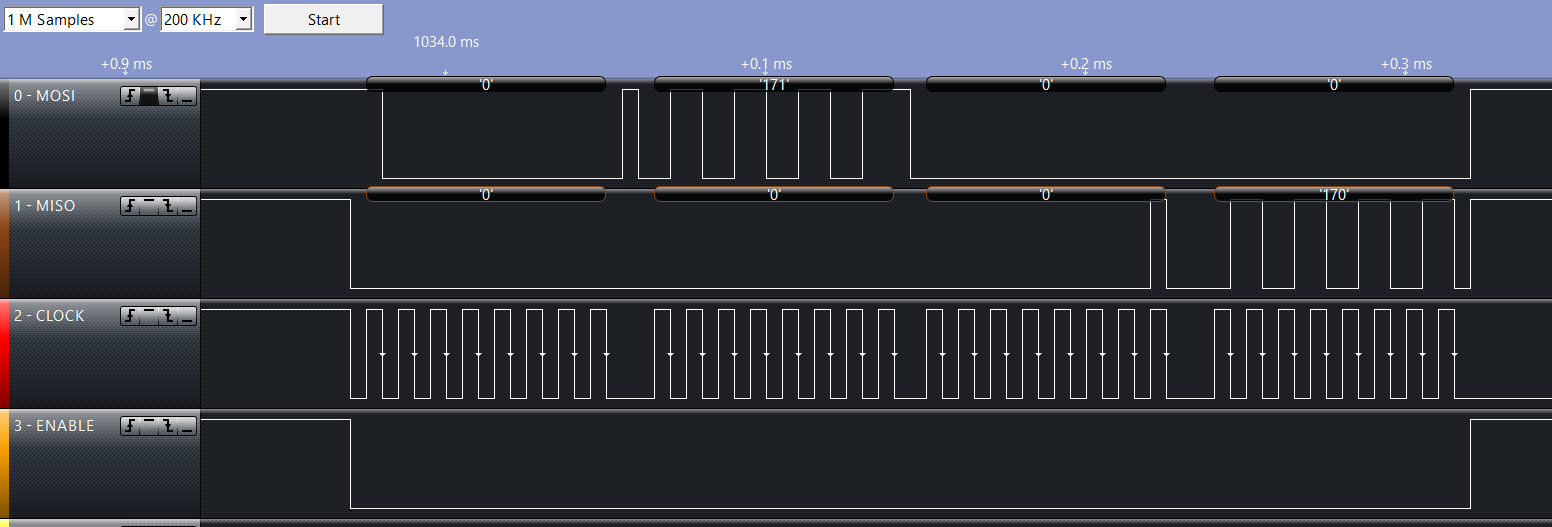


Figure 5: SPI signal from CC3200 to the ArduCAM Mini, first 16 bits are a Write, the latter a Read

**Conclusion**

In conclusion, we learned a lot about the difficulties present when trying to handle with large amounts of data being transferred over serial connections. Difficulties with transferring the data may arise due to buffer sizes, synchronization between both signals, and even more so, correct voltage levels between them. It was much more difficult than we had anticipated, and in retrospect was probably not the best choice for a project with such a short deadline. In any case, we learned much about interfacing with the camera and also about how to embed images over text using Base64 encoding and HTML.

**References**

1. Base64 C++ Encoder, by “Frank4DD”, 02/23/2009. <http://base64.sourceforge.net/b64.c>
2. ArduCAM Mini Test Sketches. <http://www.arducam.com/download/>