Rasul Silva
No partner
ID: 913737619
EEC 172

LAB 6 REPORT

Introduction:

The purpose of this lab was to make an interesting project that could perform some functions on device and then interact with the internet through WiFi to extend its functionality. My project is an arcade style game which utilizes the IR remote and the on-board accelerometer for controls; for all the games played the scores are stored on the internet through AWS.

Brief Description of Tools Used:

Code Composer Studio: This is the IDE we used to program the CC3200

Adafruit OLED: We used this to display images, we connected to this device using SPI

At&t IR Remote: We used this infrared remote to send waveforms to our IR sensor, which we interpreted in the software.

Adafruit OLED: We used this to display images, we connected to this device using SPI

BMA222: This is the on board accelerometer we used to control the movement of the ball for the application program.

TI Pin Mux Tool: We use this software to generate our pin assignment configuration files. This really simplifies the process.

Salae Logic Analyzer (Device and software): We used this to view and analyze our waveforms for both SPI and I2C.

AWS Iot: Amazon web services Iot provided our internet connected cc3200 with useful APIs to do fun things.

Simple Notification Service: Used this to send a text message to my phone after the shadow is updated on the AWS "thing".

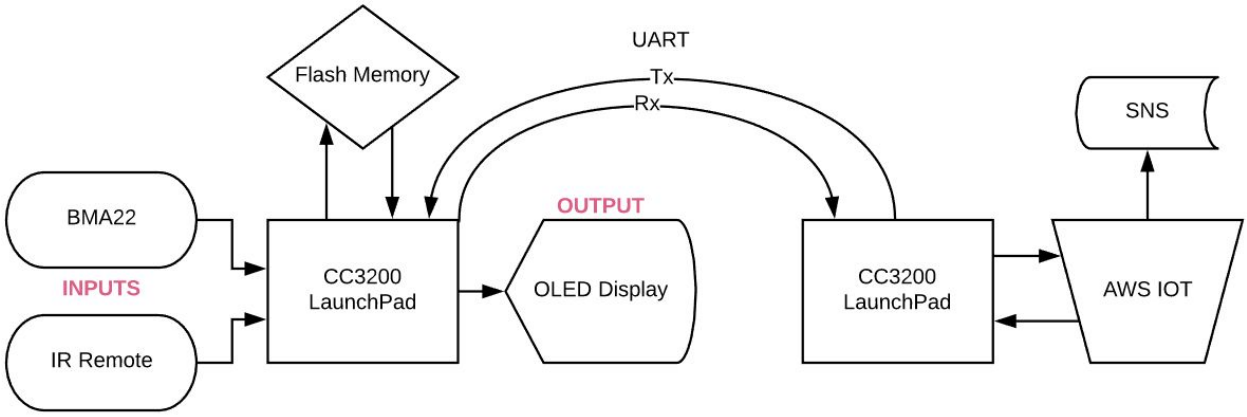
SSLShopper: I used a utility on SSLShopper.com to convert the certificates from .pem and .pem.key to .der.

CCS UniFlash: We used this to flash our certificates onto our device.

Adafruit AMG8833 Thermal Camera: *Not used in in final project* I attempted to use this camera for my initial idea, but I could not get it to work.

Project Diagram:

The following diagram describes the interactions between different elements in the system. At the core we have our TI CC3200 microcontroller. First of all, the CC3200 runs the program straight from its onboard flash. During runtime, the CC3200 takes inputs from the IR remote and the BMA22 on-board accelerometer. The IR remote is used to select an option on the main menu and the accelerometer is used during the game to control the position of the paddle. During all of this, graphics are displayed on the Adafruit OLED. Upon completion of the game, the CC3200 sends the score to the second CC3200 through UART which uses http_post() to interact with our device shadow through AWS IoT before returning back to the start screen.



Tasks:

<u>PART 1:</u> building the game

This game utilizes the accelerometer control from LAB 2 and the IR Remote input from LAB 3, I first combined those programs together; for this I had to rearrange a few things in the pinmux files. After I had both inputs functioning I set the framework for my state machine. There are 3 states: start menu, play, and game over. From the start state we have text printed out with 2 possible paths to follow: ->play or ->quit. In this state I poll the output of the IR remote to determine where to go next. If ->quit is chosen (by pressing 2) then the program terminates. If ->play is selected (by pressing 1) we go to the play state. In the play state a paddle and ball are generated on the screen. I rigged the paddle to move left or right based on the accelerometer output. This allows the user to move the paddle by tilting the CC3200. The ball is bouncing up and down and side to side. I apply a gravity of -4 to the ball, which means that every iteration the ball attempts to increase its downward velocity by 4. If the ball flies into one of the edges I force its velocity value to send it in the other direction. If the ball hits the bottom of the screen, I simply check if the ball is hitting the paddle. If it hits the paddle I send it back up with the same velocity it had before impact, thus simulating a perfectly elastic collision. Also, in the top right of the screen I display a game clock. I made the game clock by incrementing the game clock counter for every iteration of my while loop, because my loop takes so long this runs at approximately 1 tick per second (this is because the graphics take up a lot of time). This serves as the "score" for the game. The longer you last, the higher your score. If the player fails to catch the ball with the paddle, it hits the ground and explodes. This ends the game. We then go to the game over screen. Here the score is displayed for the user and the score is sent to the other board to be posted to the device shadow using http post.

GRAPHICS:

This was definitely the funnest part of designing this game. I worked really hard to make the game look cool. The main menu has a title at the top and 2 options: “Play” or “Quit”. These texts have a circled 1 and circled 2 next to them respectively. When you select an option, that circle will implode from within in a red pattern. This is achieved by using various sizes of red and black circles displayed in a certain sequence. Next on the play screen, if the ball hits the ground it explodes into a yellow and red fireball with smoke balls coming off of it. The fireball is achieved by drawing circles of increasing sizes and alternating colors. The smoke balls are created by moving a white ball in a certain trajectory. The origin of these events is decided by the touchdown location of the ball, which is saved. Lastly, on the game over screen the score will fade in from the darkness. This style of simple graphics gives the game an old school atari vibe. Trying to make it look nice was definitely the most fun I had with the project.

<u>PART 2:</u> posting to the internet

Connecting to AWS:

For connecting to AWS, I followed the same sequence of steps we used in lab 5. I had to make a “thing” and download certificates for it as “.pem” files. Then I had to convert those to “.DER” files using an online tool called SSLShopper. Then I made a policy to allow for updating the shadow. Then I create an SNS topic to initiate the phone texting. For me the act of connecting to the internet did not change from lab 5. The only change was in the http_post() method and how I build a char array of scores and push it to the shadow. I discuss this later in the “Notable Code Modifications” section.

Updating Device Shadow:

The internet component of my game is rather simple. Upon losing, the gamer is sent to a “game over” screen which displays there score into a value called “str”. This string is then sent to the second board through UART. When the second board receives the UART signal it fires off an interrupt that extracts characters from the UART line 1 by 1. At this point, the received string is pushed to the device shadow similarly to what we did in lab 5. Then the SMS service of AWS sends a text message to all subscribers with the score that was earned. The game is on an infinite loop and scores keep getting added to the shadow. So in this sense it produces an unsorted leaderboard that stores previous scores. It is important to note that the shadow scores get wiped and filled with new values every time the CC3200 is powered off and powered on again.

Notable code modifications:

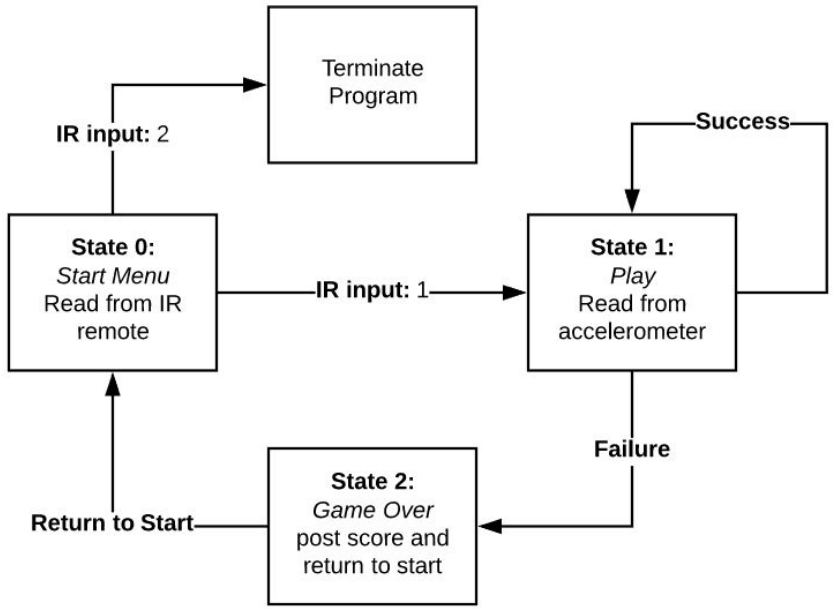
http_post():

I changed http_post() to insert a string into a field called score. I did this by examining the string “DATA1” which was already in the code. I replaced one of the fields with the score field. As for sending this; I change the value of the datalength dynamically based on the size of the string being inserted. I followed the pattern used for all the other string copies. I produce the list of scores by entering chars into an array 1 by 1 separated by commas. Then when a player plays a game he can look at the shadow and see his score added to the list. This serves as a primitive leaderboard. If I had more time I was going to have a extra button on the main menu titled “leaderboard” that would trigger an http_get() call. Then it would parse the gotten string for the leaderboard values and print them out on the screen. It would have also been cool to allow players to enter their names with their scores.

Functions I wrote: (with very brief descriptions)

paintball() - this takes x and y coordinates and a third integer to choose the color of the ball, it invokes the AdafruitGFX.c function fillCircle().
eraseball() - fills the ball at x and y with a black circle.
getgoodchar() - This is the state machine that I use for my IR remote, it chooses which letter of the alphabet we're on based on the previous number, current number, and previous character.
GPIOA2IntHandler()- This is also for the IR remote. When the CC3200 catches a signal from the IR remote this function gets the timer value, and checks a remote counter variable to determine if it should push the time to an array or not. This is stuff we discussed in LAB 3
extractmessage() - upon receiving a UART transmission, this function will pull values from the line one by one using the function MAP_UARTCharGet().

Outline of Code:
The following diagram describes the flow of operations of the program. From the start screen the IR remote is used to quit the game or play. Once in the play state, the program will run until the player fails, at which point the program goes to the game over screen. Here the game score is collected and posted to the device shadow, which consequently sends a text message to the users phone.



The second CC3200 simply sets up an internet connection and waits for activity to hit the UART line, at which point it posts the received score to the device shadow, which triggers the text to be sent to the phone.

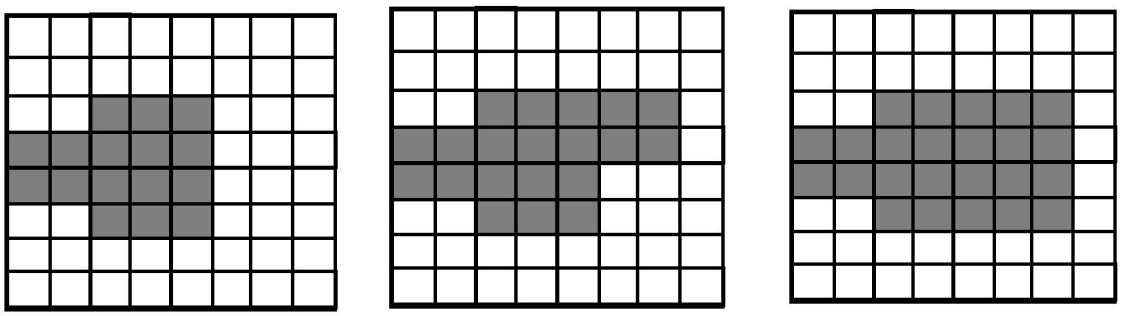
The Pursuit of Thermal Readings: A Story of Failure

My initial final project idea was the following: I was going to use the Adafruit AMG8833 thermal camera to take thermal data from a human hand. The AMG8833 returns an 8 by 8 array of temperature values through I2C that would have been just enough for my project.The idea was that I could take a thermal image of a hand and place it into 1 of 3 categories: rock, paper or scissors. I could use this classification to allow 2 people to play rock paper scissors with each other through UART or through the internet. I am convinced that this could be achieved **without** machine learning. The thermal camera returns an 8x8 array of temperature values; I believe that if the physical conditions of the photos are very controlled (constant hand distance, background temperature, angle of hand, etc.) then you can simply determine which areas contain heat and run that through some logic to determine what it is.

For example: a rock hand might appear as a "square" of high temperature values, a scissor hand might be a square with a protruding line, and paper might be a rectangle

Theoretical expected thermal camera readings: *gray is hot, white is cold*

ROCK SCISSORS PAPER



Unfortunately, I was not able to pursue this project because I was unable to properly read from the correct registers in the thermal camera. By using a logic analyzer I discovered that the CC3200 kept attempting to read from the registers 0xD2 and 0xD3. Regardless of which registers I attempted to read from it kept reverting to these registers. I tried a lot of different things, but by Saturday I had wasted a whole week on a device that was not going to work. At this point, I resorted to my plan B: an arcade style game which posts a score online.

Possible Improvements:

There are some cool features that I would add to this project if I had more time. First, I would add a leaderboard option in the main menu. Upon choosing this option the user would be sent to a screen that displays the scoreboard. This could simply acquired by invoking http_get() then parsing the result for the scores that I want. Also, It would have been fun to figure out how to let 2 players compete against each other through the internet. Perhaps they could play simultaneously and post their scores at a similar time. Then the device will decide who got the high score and let the players know.

Discussion of Challenges For Each Part:

I had much difficulty trying to get the thermal camera to work. That's why I gave up on it on Saturday. I tried many different things to get it to work. I went so far as to trying to understand the Arduino libraries that work with the device. However, I was stopped in my tracks because Arduino code uses a library called "Wire" and I could find the low-level source code for the processes they were running. I also had difficulty getting my UART to work correctly. My problem was that the pin muc config on the receiving side was wrong. I had one of my Pin modes wrong for a receiving pin. This made it impossible to receive data. Luckily, Daniel was able to catch this in lab, or else I would not have seen it myself.

Conclusion:

Overall, this was the a really rewarding lab. In my case I got to use a lot of the techniques and modules from previous labs. I used the following items:

Adafruit OLED (SPI) **UART**
BMA222 (I2C) **Timers**
IR sensor (GPIO) **Interrupts**
At&t remote **AWS & SNS**

I used everything from the quarter except for the DTMF setup. This was definitely a good project to end with because I was able to bring a lot of the stuff I learned into 1 project. Also, I had a lot of fun making the graphics for the game and figuring out how to use the internet properly for this application. This was definitely an awesome last project to cap off a great quarter of embedded systems programming practice.