Week 4

UART:

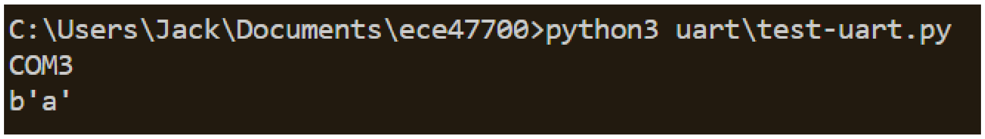
Micro → Laptop

For our project to function we need a full duplex UART channel between the laptop and the micro. Because last week I tested from the laptop to the micro, this week I wanted to test from the micro to the laptop. This process was very similar however. I again used the a simple python script to read the char sent.

```
for( ;; ) {
        putchar('a');
        //val = getchar();
    }
```

```
import serial

ser = serial.Serial('COM3', baudrate=115200)
print(ser.name)
i = 1
s = ser.read()
print(s)
ser.close()
```



```
C:\Users\Jack\Documents\ece47700>python3 uart\test-uart.py
COM3
b'a'
```

Terminal Output Showing the read char value of ‘a’

Rewrite Using C++

In order to interact with the OpenCV code, it is simpler to instead read the serial communication using C++. For this test, I used a Linux machine, which is where game logic code will run in our final design. The below C++ can communicate (read and write) with the MCU. I followed an example found here: https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c-cpp/

It is worth noting that user group and file permissions for the tty device had to be changed in order for this code to work

```
#include <stdio.h>
#include <string.h>

#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <unistd.h>

int main(int argc, char ** argv){
    struct termios tty;
    tty.c_cflag &= ~PARENB; //Parity Bit
    tty.c_cflag &= ~CSTOPB; //Stop Bit
    tty.c_cflag &= ~CSIZE; //Clear Size
    tty.c_cflag |= CS8; //Set Size
    tty.c_cflag &= ~CRTSCTS; //Disable Hardware Flow Control
    tty.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)
    tty.c_lflag &= ~ICANON; //Don't wait for \n
    tty.c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
    tty.c_cc[VTIME] = 0;    // Wait for up to 1s (10 deciseconds), returning as soon as any data is received.
    tty.c_cc[VMIN] = 1;
    cfsetispeed(&tty, B115200);
    cfsetospeed(&tty, B115200);

    int serial_port = open("/dev/ttyUSB0", O_RDWR);
    if (serial_port < 0) {
        printf("Error %i from open: %s\n", errno, strerror(errno));
    }

    if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
        printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
    }
    if(tcgetattr(serial_port, &tty) != 0) {
        printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
    }
    //char read_buf [256];
    //int n = read(serial_port, &read_buf, sizeof(read_buf));
    //printf("%c\n",read_buf[0]);
    //char sendByte;
    //sendByte = 'a';
    //write(serial_port, &sendByte, 1);
    close(serial_port);
}
```

In addition to sending and receiving UART messages, this week I outlined a structure for the uart Message:

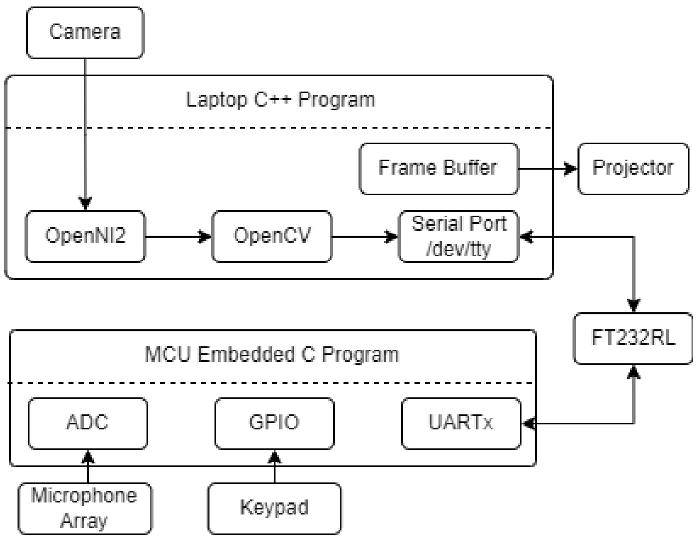
- The first bit will determine if a button was pressed
- If a button was pressed the next four bits will determine what button was pressed
- The next two bits determine what side of the table the ball bounced on:
  - 10: right
  - 01: left
  - 00: no bounce

The code to decipher this message on the computer is listed below:

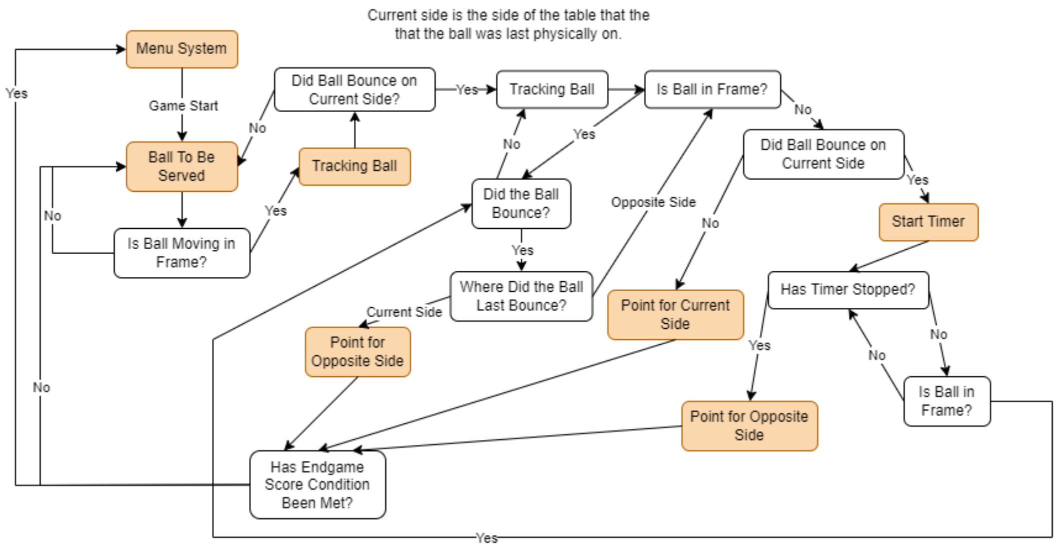
```
// DECODE MESSAGE AND SET GLOBAL VARIABLES
int decode(unsigned char message){
    if(!(message & BUTTON_PRESS_MASK)){
        curr_press = NONE;
    }else{
        curr_press = (message & BUTTON_MASK) >> 3;
    }
    unsigned char bounce_message = (message & BOUNCE_MASK);
    if((bounce_message == 0)||(bounce_message == 3)){
        curr_bounce = NONE;
    }else{
        curr_bounce = (message & BOUNCE_MASK) >> 1;
    }
    return 0;
}
```

Overview

Event Loop:



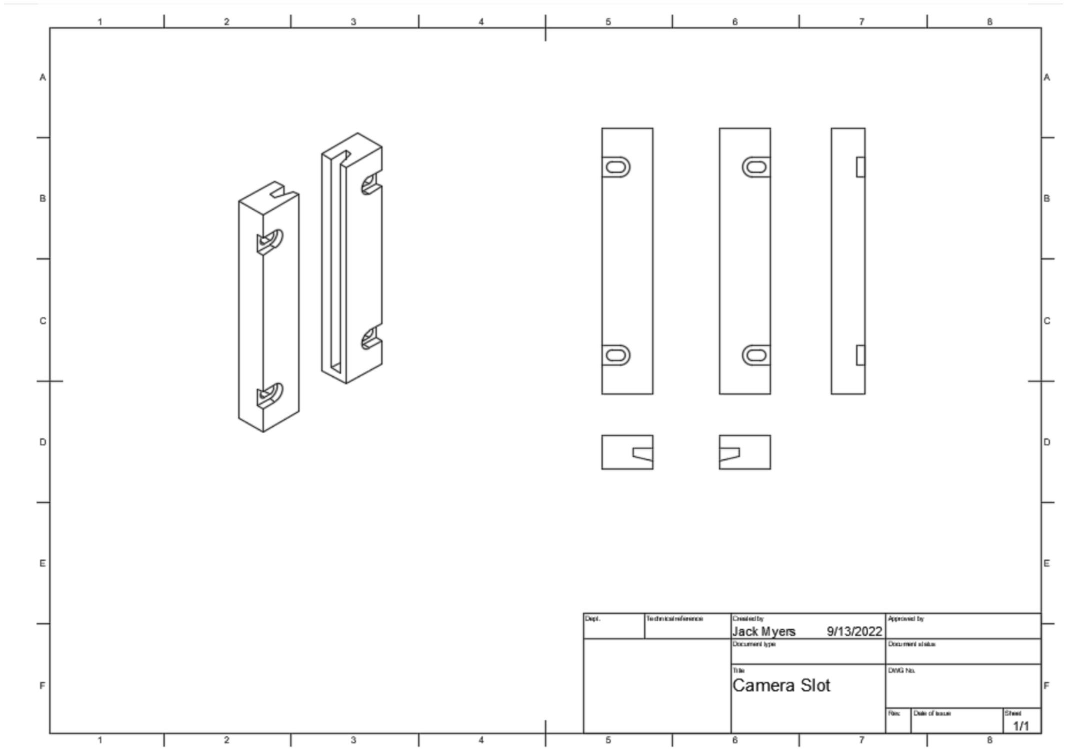
State Machine:



One thing to note for this state diagram is that it will not function correctly on the edge case that the ball is hit across the table, hit by a player and then hit on their side. In a game of Table Tennis this would be a point for the opposite player, but with this state machine will continue the game.

Camera Mount:

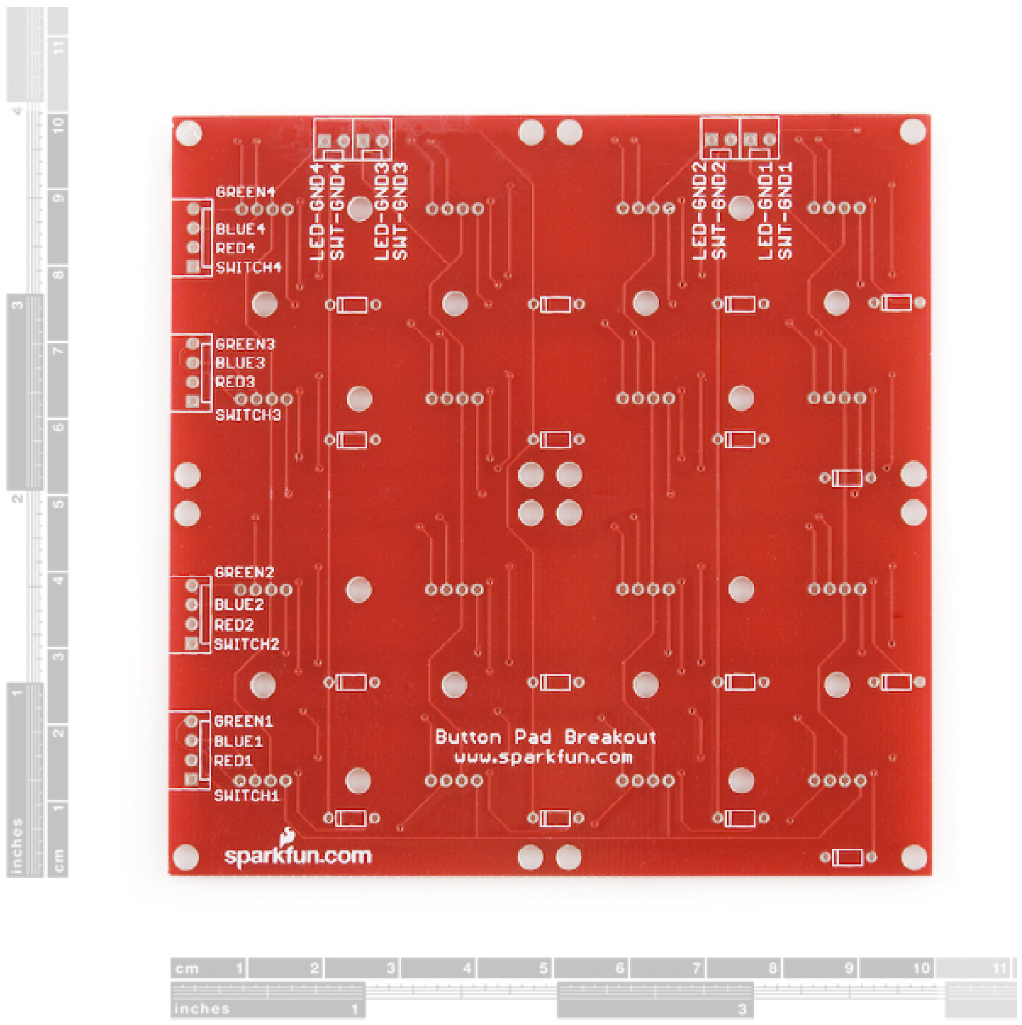
Created a CAD drawing to easily mount the camera to whatever surface we choose



The mount uses M4 screws, the same as the projector. We will attempt to 3D print this in later weeks.

Keypad:

This week we decided that we would need a new keypad with diodes to prevent ghosting. I found an alternative keypad from sparkfun:



Keypad PCB with pads for diodes

Next Week:

Game Logic:

Next week I plan to implement the game logic code above with two new conditionals to account for edge cases such as hitting the net or the table being hit multiple times

The code will use global variables and "fake" messages to make conclusions about what the score is.

Ball Tracking:

Additionally I plan to tweak the parameters of blob detection to come to a final conclusion if it is an accurate and fast algorithm for our application

Keypad:

I plan to solder diodes to the new keypad PCB we have acquired, and find contact pads or physical buttons to complete the keypad and eliminate ghosting.

Overall:

Finally, we plan to get a more permanant setup, with the projector and camera mounted somewhere stable where we can start to record and gather data of the ping pong ball on the table.