Week 2:

Date: 09/02/2022

Hours: 14

Description of design efforts:

System Planning:

Much of the work I completed this week dealt with basic configuration of the microcontrollers we've picked out, the STM32F091RCT6 from ECE 362. We felt that this was our best option, as we're all familiar with this chip, and we don't expect to need any impressive processing power handled by them, as we already have the Raspberry Pi 4B to handle graphical display and ball tracing.

We decided to use 2 microcontrollers for our project, as we will be handling multiple interrupts from multiple sources – contact microphones (x2), keypad matrices (x2), and the RPi. As our output from the contact microphones needs to be filtered and ran through the ADC, and we need constant polling of the keypad lines, we thought it was best that a single microcontroller, the "interface controller", handle these processes, and report any necessary data to the "main controller", or the other microcontroller that is responsible for handling game logic, score tracking, and RPi interfacing (which is then responsible for reporting the microcontroller's instructions to the camera and projector). To handle this communication, I set up a 2-way UART protocol between the 2 microcontrollers, as seen in Figure 1.

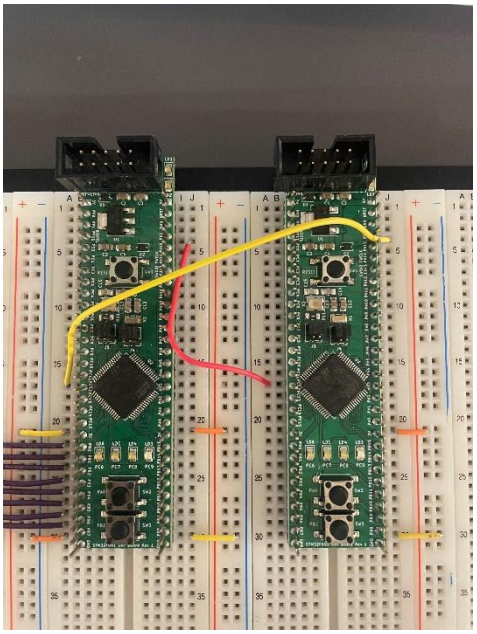

Figure 1. UART Connection

Microcontroller Programming:

With basic UART working, I began to interface with a keypad matrix. At first, I wanted to use the Adafruit NeoTrellis, an I^2C 4x4 keypad matrix with LEDs on each button. I spent some time setting up a basic I^2C interface, but unfortunately found that this device is not well documented – while there are Python and Arduino libraries for this device, I was unable to find any I^2C commands to interface with the device directly. It did not seem worth my time to scan through the library code to decipher the I^2C commands.

Instead, I opted to use the 4x4 keypad matrix we used in ECE 362, as I remember how to interact with the rows and columns to read each button. I set up a timer to run at 1 kHz to control which row to set high, then I read each column to see if any of them have gone high, indicating a button press. Then, I loop through each row, and continue this process forever. At first, the code was set up to always wait for a key press, which will not be acceptable, as this interface controller also needs to handle another keypad and 2 contact microphones. The whole microcontroller system with UART and the keypad is shown below in Figure 2.

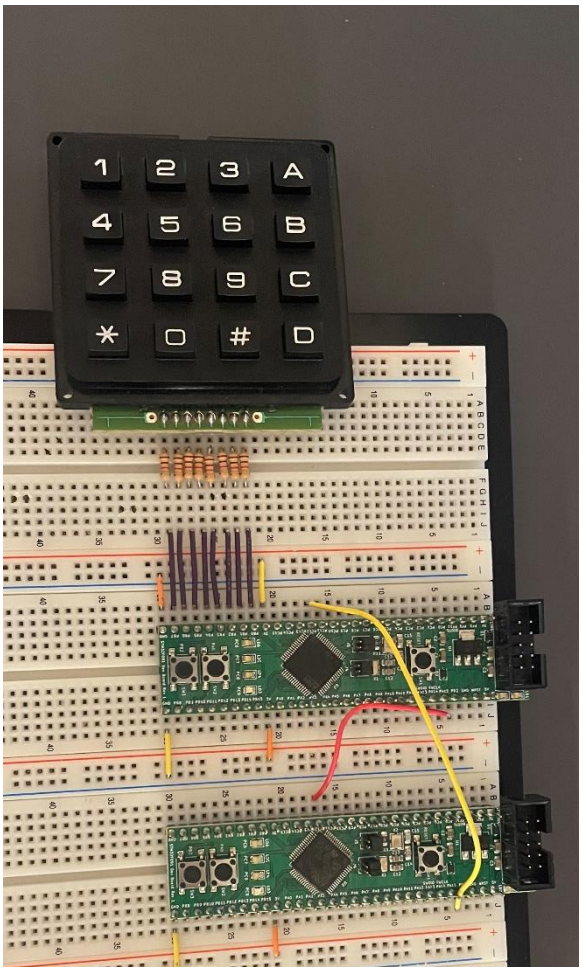

Figure 2. Keypad Wiring

Now, with working UART and a working keypad, I designed a basic “check” to ensure that we both read the correct key from the first microcontroller, and that we send that data and receive the correct key data on the second microcontroller. For now, I simply set a breakpoint in the UART ISR on the second microcontroller to read the single character that is sent. Of course, in the future, we’ll have more complex data to send. My intention is to use an enumeration that is identical on both microcontrollers to send a particular “event” to the main controller. For example, if I were to send the hex value 0x01, this may correspond to the user pressing a particular button or may mean that the ball has bounced on the left side of the table.

Website Design:

I spent the rest of my time this week setting up the website. I used the template as I don’t have thorough HTML and CSS experience, but we have discussed creating our own design in the future. In essence, I ensured all the files existed and were updated to fit our current information, i.e., our team members, team name, and PSSCs.