Week 4:

Date: 09/16/2022

Hours: 14

Description of design efforts:

System Planning:

I revised the system diagrams to reflect our new design. As they have been shown already, I will not describe any unchanged features. As seen in the functional block diagram in Figure 1, we no longer have a second microcontroller; the intermediate step will be handled by the peripheral controller (single MCU) and the laptop themselves. The UART communication link will be kept up by the laptop using a USB-to-UART IC.

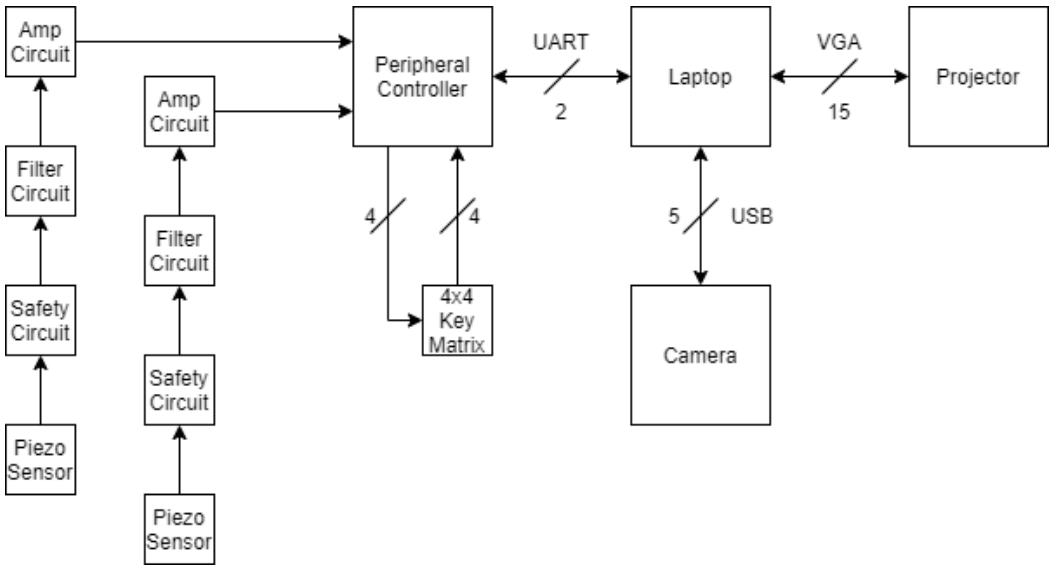

Figure 1. Functional Block Diagram

The changes shown include the removal of the “main controller” and of 1 of the 2 keypad matrices. The intention is to have the least amount of wires not part of a device or the PCB, so we plan to build the PCB with the keypad on it, and attach it to one of the two sides of the table. Here, we will have a “player 1”, as most games do, to exclusively control the gameplay. This prevents needing to have 8 long wires coming to the center of the table from each side. Of course, we will still have wires for each of the contact microphones.

I also devised a finalized solution for the UART packet structure. Communication from the laptop to the microcontroller will be very barebones, requiring only a message to request data

from the micro, and a message to force a reset. Other types of messages may become important later, which can be added as needed. More complex is the other direction of communication. The packet design on the top of Figure 2 is used to handle this.

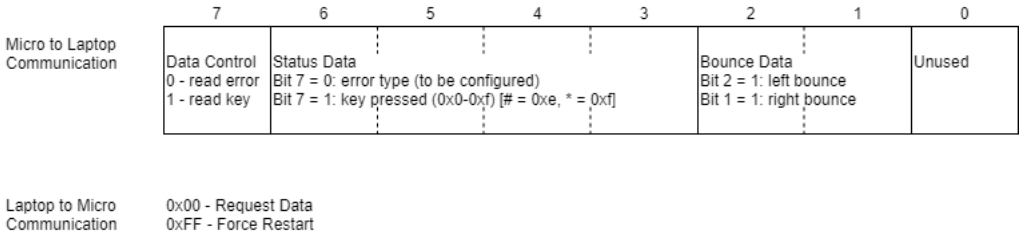

Figure 2. UART Packet Structure

Bit 0 is not (yet) necessary. Bits 1 and 2 are used to share to the laptop whether or not a bounce has occurred, on either the left or right sides (or both, which logically, should be considered an error). Bits 3-6 report which key was pressed, if bit 7 is asserted, and otherwise, report nothing (if no key was pressed), or a particular error code to report to the laptop, which are yet to be defined. These will be also added as necessary.

Hardware:

I finalized the hardware circuitry for each ADC input. We will have 1 of the following circuits, in Figure 3, for each piezoelectric sensor. It incorporates:

1. A Zener diode protection circuit (3.3V maximum output voltage)
2. A bandpass filter (1 kHz – 10 kHz passband)
    - This band was determined through frequency analysis of ball bounce responses on the piezoelectric sensor. Any signal below ~1.6 kHz was found to not be part of the piezo oscillation. As the ADC currently samples at 20 kHz, any frequency above the Nyquist rate (10 kHz) are removed.
3. A 6x gain operational amplifier circuit with power and ground rail connections, further limiting the ADC input voltage between 0 and 3.3V

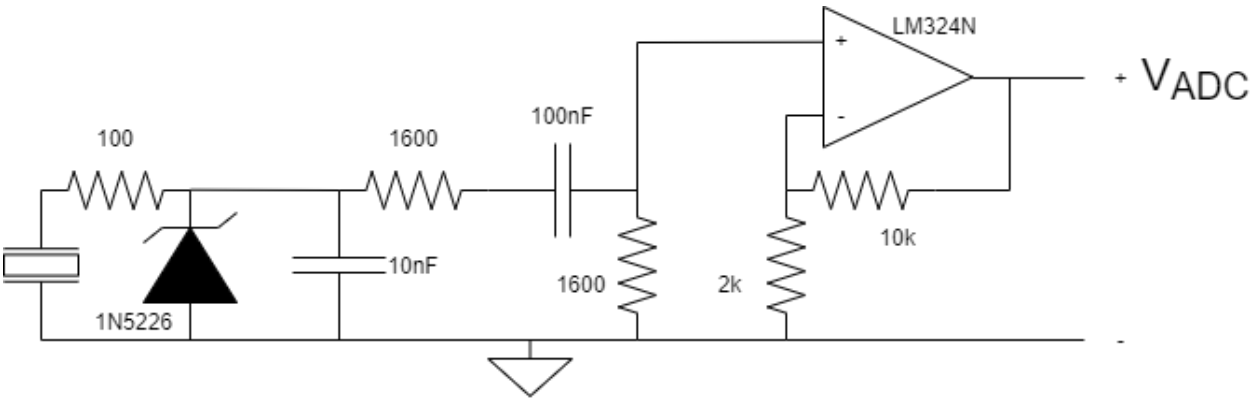

Figure 3. Piezoelectric Sensor Processing Circuit

An example of the circuit filtering a bounce is shown below on the oscilloscope, in Figure 4. Here, a bounce outputs a non-negative voltage, oscillating between 0 and 3.3V. From testing, it appears that the circuit caps the voltage at around 2 V. A typical hit from a table tennis paddle will see a spike maxing between 1 and 2V, depending on the strength of the hit and the distance from the sensor.

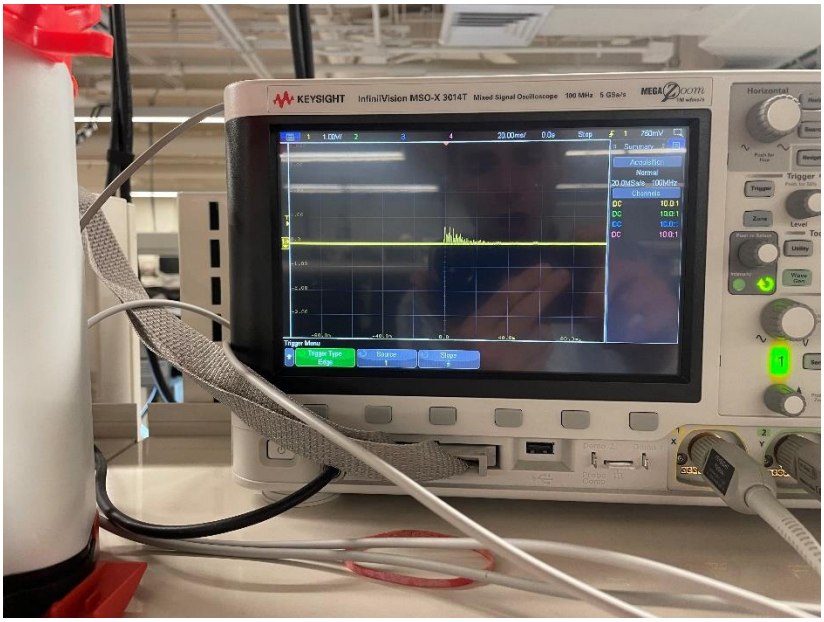

Figure 4. Oscilloscope Output of Ping-Pong Ball Bounce

Microcontroller Programming:

Jack and I confirmed we have working UART from the microcontroller to the laptop. Now, we just need to set up the particular packet structure on both sides to send and process particular information for the game logic handling on the laptop that I designed this week.

Most of my work this week has been dedicated towards designing the hardware and software for the ADC. With the hardware complete, I designed a basic calibration algorithm that creates a "threshold" for the ADC to detect a bounce. If a value is ever read above this, we determine it to be a bounce. It is not robust and needs some planning and more memory intensive algorithms to solidify. That said, it is not weak, either. All bounces I tested that would actually be caused by a game of ping pong (i.e. have somewhat of a velocity) triggered the threshold. However, with the new amplifier strength, wiggling the wires from the piezo to the microcontroller *can* cause a threshold cross, if done vigorously enough. To fix this, we will ensure the wires are tightly bound to the table.

I also ensured that multiple ADC channels work properly, though I have not yet tested multiple contact microphones at once. The most difficult task to come is creating thresholds that block as much signal as possible from bounces on the other side of the table but are not too stringent such that we miss bounces on the correct side of the table.

Next Week:

The microcontroller code is nearly complete. The only things that need to be changed or added are listed below:

1. Solidify packet structure to be sent out, only upon request from the laptop
2. Design a robust calibration sequence to handle bounce detection
    a. Requires single sensor and multi-sensor testing
    b. Needs to filter out all bounces from the opposite side of the table, without missing any bounces
3. Test the limits of the ADC sample rate – answer the following questions: How many piezoelectric sensors can we read without skipping samples? How quickly can we sample without the possibility of "running over" interrupts?
4. Write a software reset algorithm (this is likely a register value that can be written, I just need to find the documentation for it)

With these tasks complete, I will begin to assist with other work, relating to either PCB design or software development of ball tracking.