Keyboard HEERO

Adrienne Mok, Ashwin Narkar, and Charlotte McGinn

Electrical Engineering 3

Dr. Briggs

Fall 2016

Introduction

Keyboard HEERO, a slight twist to the classic Guitar Hero game, allows players to channel their inner pianist and jam out to their favorite songs. As a stream of notes flows down, the player must hit the corresponding “piano keys” in time with the beat of the music. We chose this project because we thought it would be a fun way to demonstrate some of the fundamental electrical engineering concepts we learned this quarter. Through this project, we aimed to successfully design, build, and test the hardware and software of a game similar to Guitar Hero. In order to do so, we needed to design an appropriate circuit, integrate components with a microcontroller, and process and interpret signals.

Our project comprises of a microcontroller (Teensy 3.2), several pushbuttons, LED strips, a speaker, and a four digit seven-segment display. The LED strips give the player a visual representation of the notes as they fall down and the pushbuttons represent the keys of the piano. A speaker plays our game’s 8-bit song. The four digit seven-segment display shows the player’s score. Since the LED strips require a voltage of 5V to operate, we connected them to a DC power supply. Our microcontroller allows us to read different inputs, such as the current state of the pushbuttons, and send out different outputs, such as the light emitted by the LED strips. A bulk of this project involved programming the microcontroller to allow us to control the system in an appropriate manner.

When building the circuit, we had to apply many of the principles we learned in lecture. To avoid frying our Teensy, we had to make sure our components did not draw

too much current from it. In order to limit the flow of current, we had to apply Ohm's Law (V=IR) to figure out the appropriate resistance value. For example, for our seven-segment display, we needed to use 330Ω resistors so that the LEDs in our display would not burn out. Additionally, we needed to read all of our components' data sheets carefully to avoid exceeding their maximum ratings.

Testing Methodology

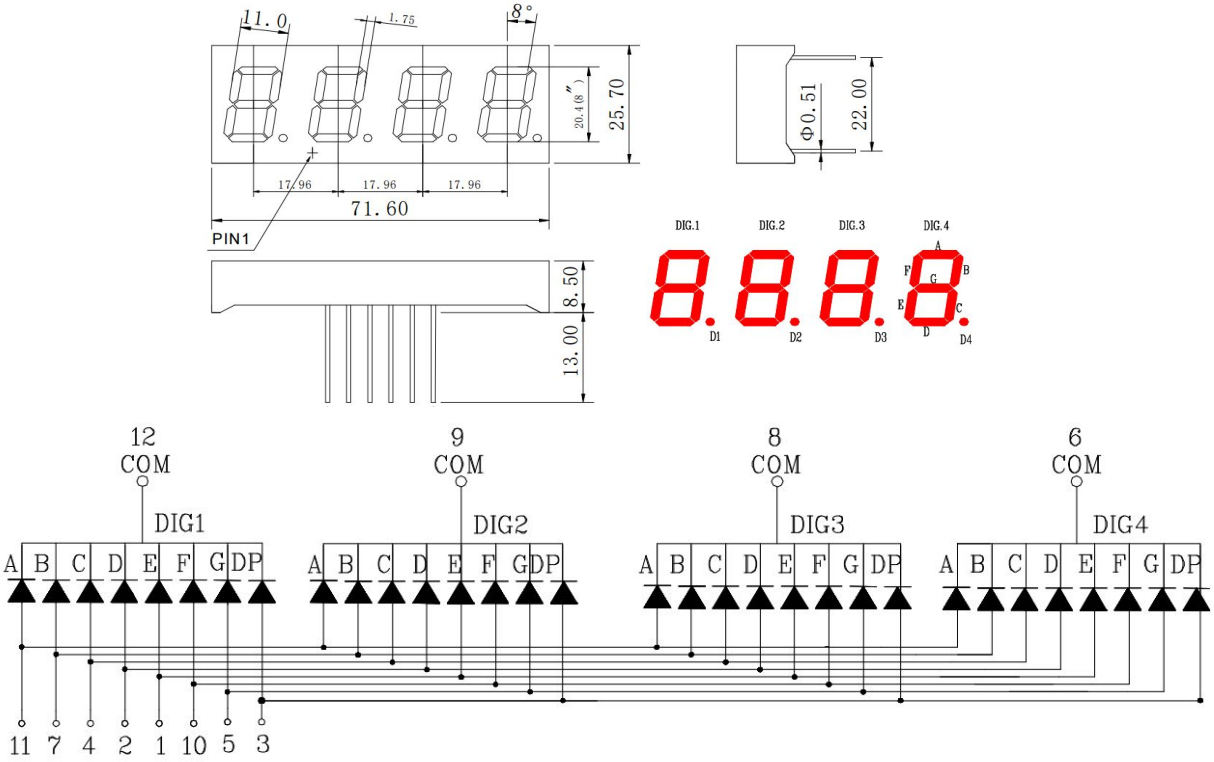
How We Designed the Test

Before placing any of our components in our circuit, we read their appropriate data sheets to ensure they were used properly. By doing so, we were able to identify their maximum ratings, see how they needed to be connected for proper data output, and check how much power they needed. To ensure our components were working properly, we used various tools available to us, such as the digital multimeter, which has several functions useful to us. Since a bulk of our project relies on sending and receiving data through our microcontroller, a lot of our tests involved debugging and manipulating our code, as well as using Arduino's useful Serial Monitor to analyze our data.

How We Conducted the Test

The four digit seven-segment display relied heavily on reading its datasheet found on Sparkfun. The datasheet showed us which pins corresponded to which segments and digits, allowing us to program the display later. **Figure 1** shows some relevant information from the datasheet.

Figure 1: Schematic of Four Digit Seven-Segment Display- Shows pin layout of display



Our addressable LED strips had three connections: one for ground, one for 5V, and one for data output. Since we used four different segments of LED strips, we had to solder wires to a couple of the segments' copper pads. We used the digital multimeter's continuity function to make sure the connections were secure.

To verify that our Teensy 3.2 was working out of the box, we plugged it into the computer and uploaded the generic blink code. The blink code is a standard code that is often uploaded to microcontrollers to make sure all of the pins are connected correctly and that the microcontroller has its basic functionality.

Since we used pull-down resistors for our pushbuttons, their default state was low. Once pressed, their corresponding pin is read as high. To check that button presses were registered, we used Arduino's Serial Monitor to print "Button pressed"

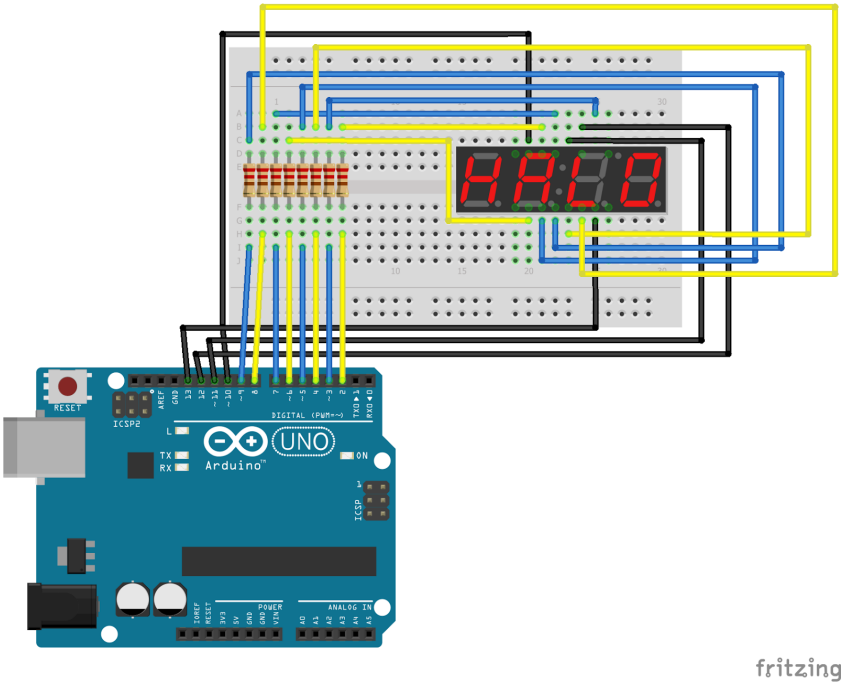
whenever the button was pressed. By doing so, we could see whether or not the button was working in general, as well as whether or not the button was registering extra button presses.

The speaker was fairly easy to test, as we only had to play our 8-bit song and check to see if sound was emitted or not. The wiring was straightforward, since there was only one wire connecting to the Teensy's data pin and another wire connecting the speaker to ground.

How We Analyzed the Data

The four digit seven-segment display's datasheet showed us the pins corresponding to the segments and digits [1]. Numbers are displayed by switching the digits' state between low and high repeatedly [2]. Thus, the numbers are refreshed at a high rate, which is virtually imperceptible to the human eye. Even though the numbers are only turned off for a short period of time, in order to protect the segments' LEDs we connected 330Ω resistors between the segment pins and the microcontroller data pins. We applied Ohm's Law to find the appropriate resistor value. **Figure 2** shows the general connections for the display.

Figure 2: Connecting the Seven-Segment Display to the Microcontroller- Shows how the pins of the display are connected to the microcontroller. Note that we used a Teensy instead of the Arduino pictured below



The continuity function works by testing if current can flow in a complete path. When testing for continuity, the position of the leads is not relevant. If a particular component has continuity, the digital multimeter will emit an audible beep to signify a complete path. For all of our solder joints, we checked for that “beep” to assure us our components were connected properly. If no sound was heard, we resoldered the joint to form a better connection.

After uploading the Blink Code to the Teensy, we waited to see a red light being emitted from the board. The Teensy’s built-in LED flashed as expected, leading us to conclude that it was operating normally. We then proceeded to upload our project’s actual code.

For the pushbuttons, Arduino's Serial Monitor helped us see when button presses were being registered. The pushbuttons worked as expected, and the Serial Monitor printed "Button pressed" whenever a button was pressed. However, we did have a problem with button debouncing. Button debouncing is often caused by mechanical irregularities in the switch. This can lead to inaccurate readings, specifically multiple button presses being registered within a short period of time. There are two ways to fix button debouncing, either through software or hardware.

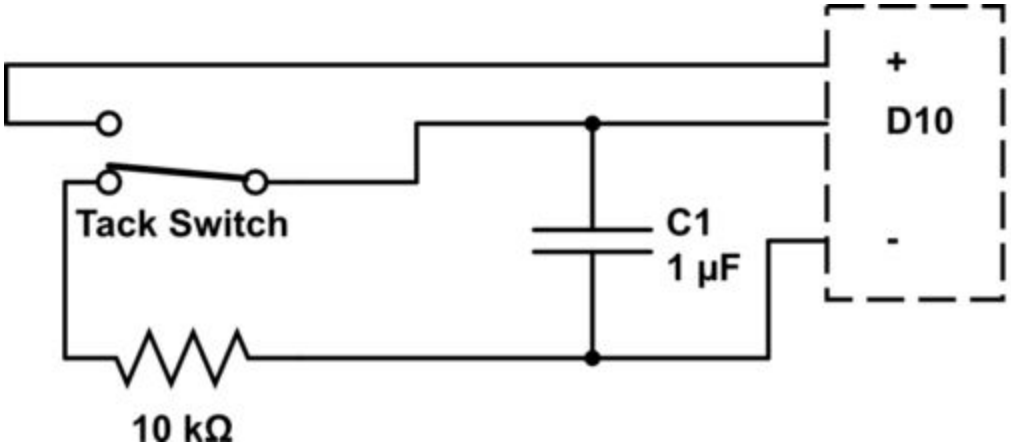
How We Interpreted the Data

The four digit seven-segment display's datasheet, particularly the schematic found inside, was helpful in ensuring the display was properly connected to the Teensy and made it a lot easier to program it. Without the datasheet, it definitely would have taken a lot longer to figure out where everything should be connected. Furthermore, Ohm's Law allowed us to find an appropriate resistor value that would limit current flow to the segments' LEDs while still making sure they were still bright enough to be seen. The digital multimeter's handy continuity function allowed us to quickly identify which solder joints were connected and which ones needed to be resoldered. Without using the continuity function, we would have to rely on visual inspection, which is not as accurate.

The Blink Code is built into Arduino and takes a few seconds to upload to the microcontroller. Within a short amount of time, we were able to verify that our Teensy was working normally. This feature is also handy to check whether or not a microcontroller has been fried.

The Serial Monitor in Arduino allowed us to check when the button pin was read as high. As mentioned in the previous section, we had issues with button debouncing. Through software we can fix button debouncing by using delays in our code. However, to minimize the inefficiency of delays, we opted for the hardware solution (**Figure 3**). For each pushbutton, we connected a 1 µF capacitor from the pushbutton's respective data pin to ground. Pressing the button switches the pin to high and charges the capacitor. After the switch is released, the capacitor keeps the pin high for a little bit longer, which helps counter button debouncing.

Figure 3: Pushbutton Circuit- A 10k pull-down resistor makes the switch's default state low. A capacitor offers a hardware solution to button debouncing



When we put everything together, we encountered an obstacle such that we could not make the four digit seven-segment display show a score while the user is playing the game for scores with two or more digits. This is because displaying digits on this display requires rapidly flickering the different digits on and off. We could not put the delays required to make the display work in the code because it would mess up the

timing for the game itself. Our solution was to simply display the score at the end of the game rather than throughout, but in the future we would like to display the running score throughout the game by introducing a second Teensy for controlling the display.

Results and Discussion

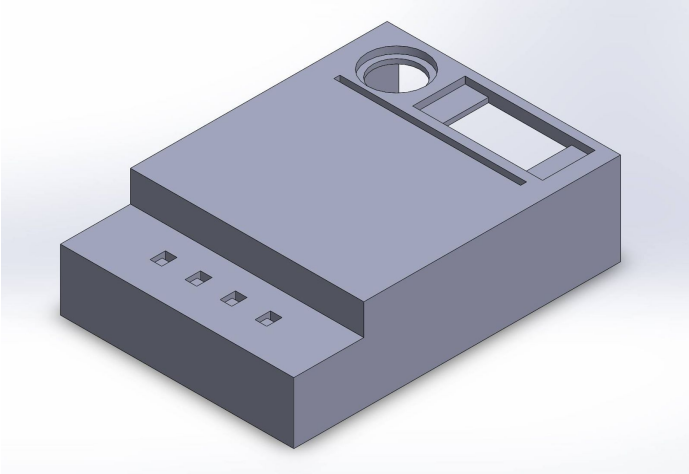
The testing results are mainly the same as what is seen in the “Testing Methodology” section. The testing procedure we employed allowed us to build a working circuit. Since we tested some of the components before and fixed any irregularities, our project was easier to debug later on, and it was faster to identify which part of our project was faulty. After the generic hardware was built, most of our testing involved the actual code we programmed into our microcontroller. Programming the four digit seven-segment display and the pushbuttons was straightforward, since it did not require that much intensive code. However, we had the most trouble with making sure that a particular note from the song corresponded to the appropriate LED at exactly the right time. We had to solve this by altering the delay of the notes, since the song’s note did not all have the same time duration. After tinkering with the code for a while, we were able to make a functioning program.

Assembly and Construction

The final aspect of our project after testing it involved constructing and assembling the casing. Initially, we wanted to 3D print a casing for our project. We developed a CAD model of the design (**Figure 4**). However, the 3D printing would

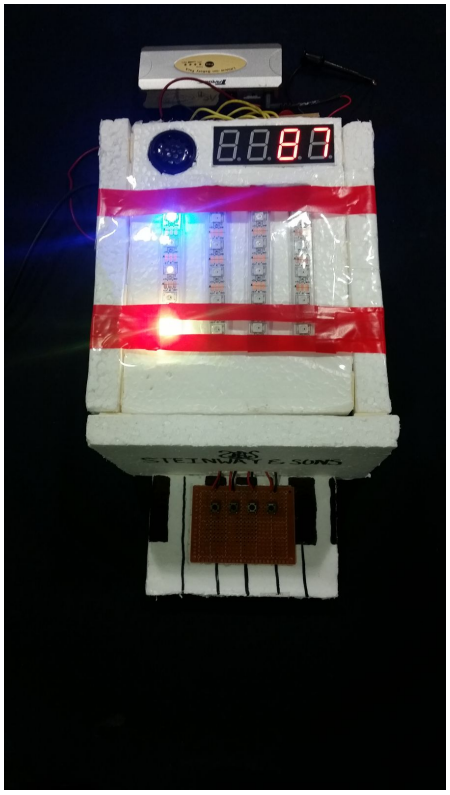
involve too many parts and overhangs that are difficult to print a 3D model for. We decided to create a casing out of styrofoam insulation and mounting tape.

Figure 4: CAD Model of the casing designed in Solidworks: Shows the holes cut out for the speakers and the 7 segment display. The holes for buttons are also cut out.



The casing was still based off the CAD model. The final styrofoam model required slits to be cut out for the wires from the LED strings and buttons. The buttons were then soldered onto a perfboard and connected to longer wires which went inside the casing where the breadboard and teensy would be placed. The 7 segment display was connected to the breadboard via female header pins which were soldered to another perfboard, with more wires that to the breadboard. Wires were also soldered onto the speaker. The perfboard for the buttons was hot glued to the casing and the rest of it was taped. The LED strips had adhesive backing so these were directly pressed onto the casing. The casing was then decorated to look like a piano (**Figure 5**).

Figure 5: Final Casing and Assembly: The final decorated styrofoam casing fully assembled and decorated. Note the hollow back end for the battery and breadboard.



Conclusion and Future Work

Overall, our project met our general expectations. We were able to create a fun project that performed its expected function while building upon the electrical engineering and computer science skills we have learned in our courses. The project gave us a greater insight as to how project design works from the initial conception of the idea, circuit design, testing and debugging, and hopefully achieving the desired finish project. Furthermore, we were able to work in a team-environment and collaborate with each other with our ideas.

In the future, we would like to add a greater variety of songs to our game. Our current project only has the option of one song. Ideally, the player would be able to choose from a number of different songs based on their interests. To select a song, we could have the player use the pushbuttons to scroll through the different options and eventually pick the song they want. The hardware would not have to be altered at all. Most of the changes would lie in the code. Since this is the case, testing to see that this added feature works would be as simple as uploading the code to the Teensy, analyzing data from the Serial Monitor if necessary, and altering the code if needed.

We would also like to have the four digit seven-segment display show the running total for the project, as was mentioned earlier. In order to do this, we would introduce a second Teensy microcontroller to control the display. We would use a serial connection between the two Teensys to send the data from the first to the second. This solution would require powering a second Teensy, rewiring the display, wiring the transmitting and receiving pins of the two Teensys, and modifying the code.

Illustration Credits

1. http://cdn.sparkfun.com/datasheets/Components/LED/1LEDREDCC.pdf
2. https://www.hackster.io/meljr/sparkfun-com-11408-4-digit-7-segment-display-hookup-guide-4b4d9e
3. http://electronics.stackexchange.com/questions/64770/is-it-possible-to-use-just-a-capacitor-to-debounce-a-button

References

1. http://cdn.sparkfun.com/datasheets/Components/LED/1LEDREDCC.pdf
2. https://www.hackster.io/meljr/sparkfun-com-11408-4-digit-7-segment-display-hookup-guide-4b4d9e