Joseph Suarez

Aug 16, 2013

EE47

Custom Tactile Interface MP3 Player

**Design Progression:**

Originally, I planned to develop a nine-button interfaced MP3 player for convenient, tactile use. The user would control the player from his/her pocket, using three fingers to easily manipulate the 3X3 button grid of buttons. Of course, this would involve a slight learning curve, but after a single day the interface would become second nature. A picture of this primary design is included in the “joseph15 Final Project” directory with the name “Original Plan.”

After considering the relative uselessness of two of the buttons (fine volume controls), I revised my project plan to include only seven buttons. As these buttons no longer fit in a 3X3 grid, I reordered them into the shape of a hexagon. This design allows the user to center a finger on the middle button (play) and easily navigate the rest of the interface by moving the finger in any direction. However, while the button layout improved, the case was still excessively complicated and therefore would have required 3d printing–a time consuming and expensive procedure. A diagram of this revised design is also included in the aforementioned directory with the name “Revised Plan.”

My third and final design made use of a laser cutter to produce an elegant acrylic case without the need for 3d printing. In addition, the ability to etch images onto the case pieces allowed me to both label the buttons and add a personal touch. However, the case design did not come easily: three iterations were required to produce a usable result. An image of the two rejected prototypes is included in the same directory with the name “Early Prototypes.”

**Execution:**

Although I dropped one planned feature due to time constraints (in the final design, the battery pack must be removed to charge the batteries), the project execution went smoothly overall. The case is meant to represent only an approximate scale model, of course: given integrated components, the player could be made pocket size. Despite last minute hustle, the final programming and assembling saw no impassible difficulties.

**Point of View:**

Scaled down to pocket size, my MP3 player presents the ideal tool for mobile listeners, especially runners. It solves the simple and annoying problem that arises from using an iPhone as an MP3 player: songs cannot be changed and shuffle/loop settings cannot be adjusted without being able to see the device. My design allows all settings to be changed via a simple click from within the user’s pocket. In addition, the buttons will not accidentally depress, as they are flush with the rest of the device. Overall, it suits its intended purpose perfectly, effectively providing music to the mobile listener without the unnecessary hassle inherent in any screen-based device.

**Process:**

I allocated time and resources efficiently, for the most part. While I removed the battery-charging feature, this cut actually improves convenience. Keeping one set of batteries charging while using the other ensures constant battery life. No other significant cuts were made, but several improvements manifested during the overall design, manufacture, and assembly process (discussed in the Design Progression section of this write-up).

**Documentation:**

Aside from the barebones, my MP3 player contains no convoluted features that would require any sort of scrutiny in order to comprehend. My custom device is a product of UI design and execution, not complex circuitry. The knotted appearance of my mp3 player is simply a result of my attempt to conserve space; the wiring could be easier accomplished on a two dimensional board. The code is also quite simple, focusing mainly on the manipulation of the current song index. A copy of the code has been included as “MP3 Code.” In addition, photos of a Verplank diagram and a state diagram along with a video of the MP3 player have been included. As the state diagram is incredibly simple, a more complex decision tree has also been included.

**Additional Notes and Discussion of Challenges:**

Along the course of the development process, a few particular problems presented themselves. These include writing non-blocking code, wiring circuitry in three dimensions, and isolating error sources. To address the coding issue, the “delay” command could not be used to debounce buttons. Attempted use caused skipping in the track, which in turn caused screeching painful to the ear. The solution is quite simple: the millis() function returns the current time. Hence, the time of the last button press may be saved in a variable. When a button is “pressed” again, the current time is checked against the old time to determine if the button is simply bouncing without user interference. As for wiring circuitry in three dimensions, I found that gluing the components to both each other and the clamp produced a firm enough hold to allow for easy soldering. Finally, during the process of the lab, a wiring mistake half-fried my SD card. When the problem surfaced, I ran SineTest, Blink, and CardInfo. As CardInfo was the only test program that failed, I determined that only the SD card needed replacement. Overall, problems that at first seemed illogical soon became comprehendible via debugging.