

■■■■

The goal of my project was to create a keyboard controller. The original inspiration for my project was an old (and by old I mean ancient) laptop that I had disassembled several years ago. I had kept the keyboard thinking I might find some use for it. And this was just that chance.

So I started out using a multi-meter and I had fully mapped out the keyboard matrix for each key.

But then came the process of actually using the Z16 to implement a matrix encoder. I quickly found that there was not a really good way of creating a solid electrical connection to the three FFC (flexible flat connectors) that the keyboard used, as I had not in my wisdom remembered to de-solder the connectors from the dead laptop's motherboard, which had long since gone to the electronics recycler. At first I thought about trying to solder to the individual pins of the FFC cable, but I was too afraid of ruining the part, thereby ruining all hope of completing the project. What ended up somewhat working was using a IDE connector as a 40 pin IDE connector had the same wire spacing as the cable, however I could only get one side to actually create a reliable connection.

So my hopes of using the laptop keyboard were dashed.

Fortunately as part of my research for USB-enabling my ghetto keyboard, I had taken apart a fully working Apple keyboard I had rescued from the trash. The Apple keyboard was using some untraceable Cypress chip.

However with this keyboard I not only had the keys and the matrix, but I still had the circuit board FFC connectors. So I desoldered them from the circuit board, and attached them to, you guessed it, the ever purposeful IDE cable, which I used to extend the cables to a solderless breadboard.

From there I ran individual jumper wires to the pins on the Z16 development board. Finally I had a solid electrical connection to both sides of a keyboard matrix, and I could finally write code.

And now that I had a full electrical connection, I could use the power of the microcontroller to map out the matrix, instead of using the multimeter. I discovered that the Apple keyboard had 19 "columns" on one cable and 8 "rows" on the other cable with an additional 2 lines for the capslock led. I also found out that the Apple matrix was much much more organized, with keys actually in rows and columns, whereas the laptop keyboard was all over the place.

Starting out I elected to use the B and H ports to scan with as they were all together. I added port G for a total of 20 scanlines, (for the 19 cable traces). I started out using port D to detect on, but for some reason it wasn't recognizing several of the rows, so I switched to using port C. At first I implemented scanning by setting my scan lines as outputs, and setting all but one to logic high, with one at logic low, with the detector lines set as input with internal pullup. This worked all well and good for singular input per row, but as soon as two keys in the same row were pressed simultaneously, it would register that the first key was raised, and then the second key was pressed,

After a while I realized that in order to be able to detect multiple presses in the same row, I'd need to exploit the Tri-state nature. It occurred to me that a logic high shorted to a logic low would not necessarily be either, so I would need to instead of driving the non-scanning lines high, would need to let them naturally be high, and be pulled low as needed. So now scanning became turning on only one line as output, the rest as input with internal pullups, and detecting via a whole input byte with pullups. This however was causing issues as keypresses seemed to bleed over into the next scanline, no matter how long I paused in between keystrokes. The only way I could find to alleviate this, was to turn off all output lines and detect what I knew would be nothing, which seemed to completely erase my bleeding issue.

So at last I now had the matrix mapped out, and was able to reliably detect when a key was pressed. Adding the un-press event was simple.

The next step was to enable some sort of PS/2 scancode output. Taking my trusty solution for Lab 3, and the PS/2 cable from yet another old keyboard (this one was destroyed when the L and N keys stopped working). At first it looked like I was transmitting the scancodes too quickly, however some additional pauses fixed that. Soon my Lab 3 was able to detect all the scancodes I was sending across the PS/2 cable.

The next step was to add some additional functionality. The first thing I tried was creating a secondary matrix lookup that used the Dvorak layout instead of the standard querty layout. Mapping the keys was fairly straightforward, however implementing the switching between the two layouts was problematic.

The next feature that I added was some macros to several of the keys that are not normally found on a Windows keyboard, such as the HELP button, which now explicitly spells out HELP, and the Eject key which now sends my name, Greg Sandstrom.

The next thing I tried was actually plugging my new ghetto keyboard into a real computer PS/2 port, not just my solution to Lab3. This would cause the mouse on that computer to stop working until the keyboard was removed. I tried a couple things to detect the computer host bringing the clock line low and then attempt to receive the host to device communication, however I don't think I quite got the timing right, as the same hanging would occur on the host device.

All in all, I thought that it was an interesting project. I had never realized before how much work goes into a keyboard. I also learned a few lessons along the way. The first of which is that there are many uses of IDE cables, not just for hooking up old hard drives. The Second lesson was that the Tri-state principle can be very useful when used correctly. The third lesson is that when keeping things that have fancy connectors, save both sides of the connector, just in case. And lastly that desoldering stuff is pretty cool.