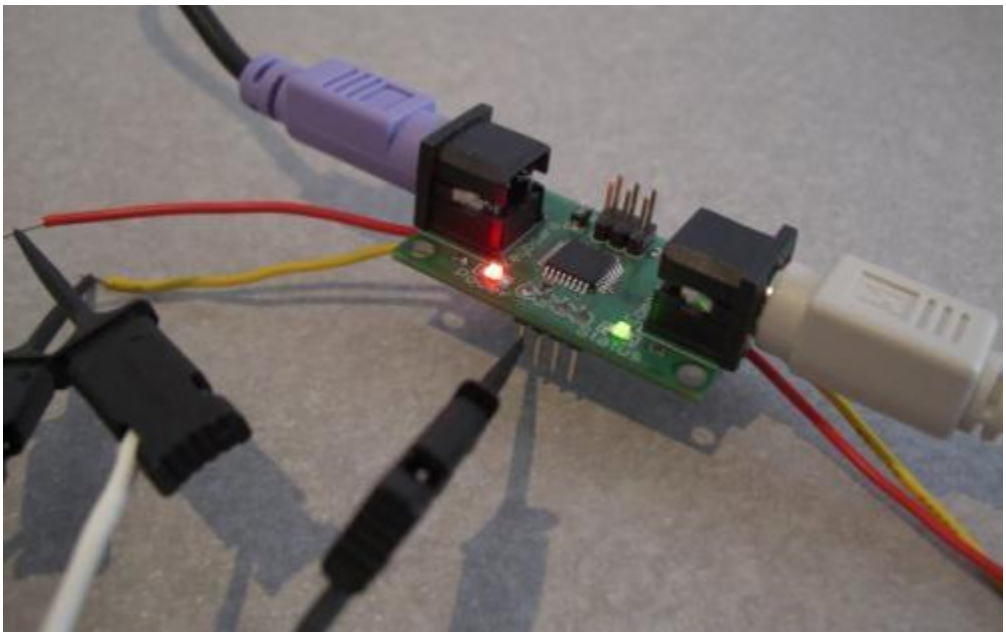
The Key-Counter -

I wonder how many letters I type in a day. Is it 1000 key presses? 10,000? Or is it something horrendous like a quarter million? Just how much of my life is given up to key pounding? There was no good way to tell. I could try to write a VB app of sorts to monitor key presses but then I have to deal with the operating system and whether or not the OS is really reporting all the key presses (shift is a key after all). It's probably a trivial exercise under Linux, but I'm no Linux guru - I'm a hardware person.

So instead I settled on creating a hardware device that could sit in between the keyboard and the computer and count the key signals going back and forth. It was quite the learning experience!

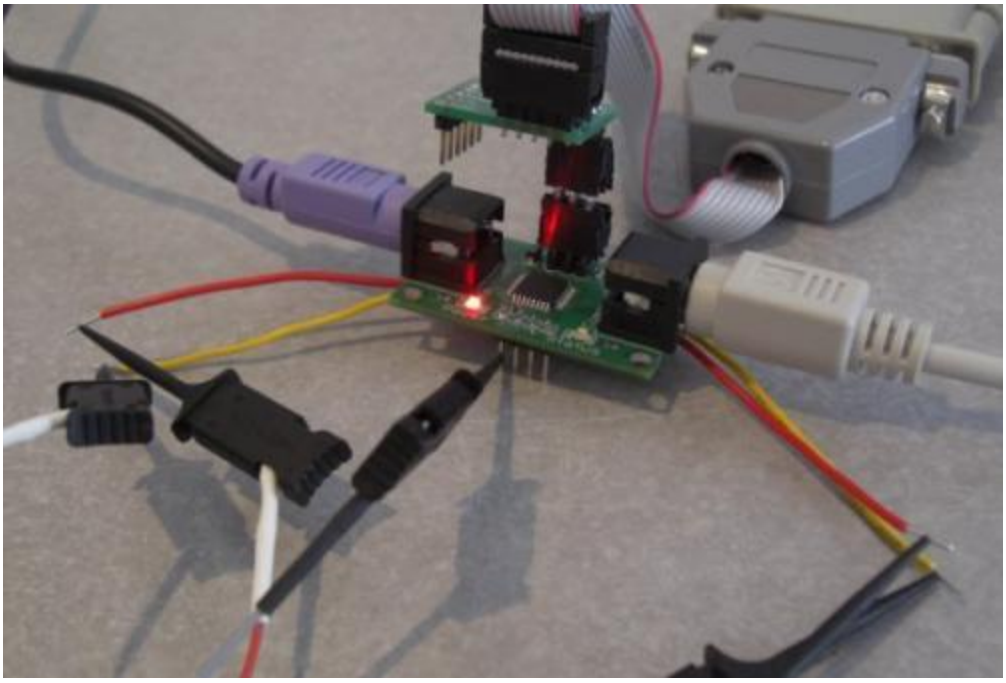


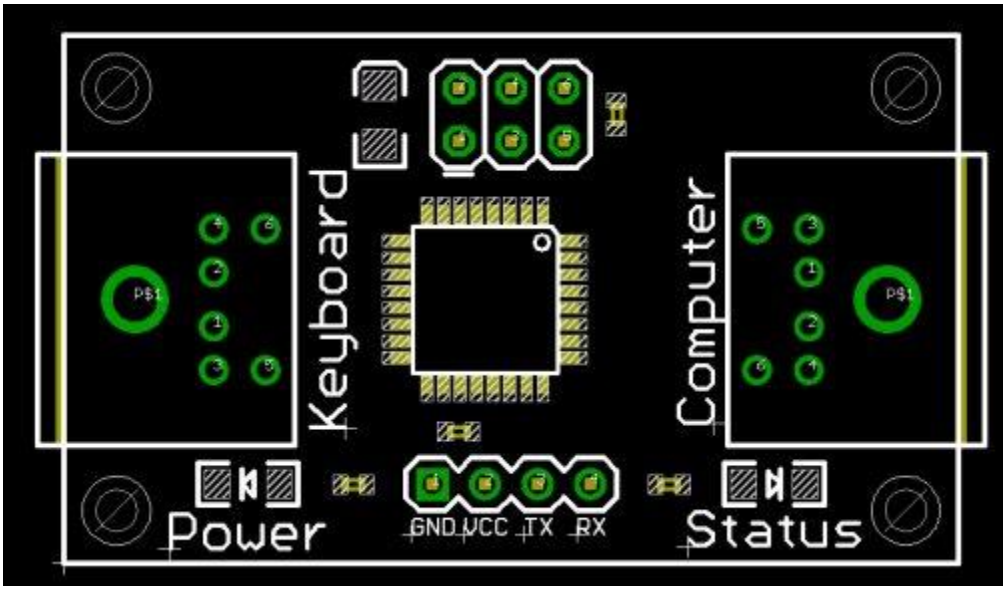
Key-Counter - Just an ATmega168 in between two mini DIN 6 connectors. You can get a Key Counter here.

``` Key presses today: 000001298

Assuming I can get the hardware to work, I will have a device that can listen to and manipulate PS/2 commands. What can one do with such a device? The applications are actually pretty wide. Of course I could try to turn it into a keyboard key logger and use it maliciously, but that's not really that fun.

What if we could change every 't' to 'T'? That would be really annoying (and fun to do to someone else). To manipulate a 't' like that, we cannot just listen, we have to listen and then change the data. A logger can't do that. But the Key-Counter can!

What if you needed to re-route all the QWERTY commands to a Dvorak setup? This hardware could do it without a single driver or OS requirement!

What about an algorithm that predicts what you're trying to spell? Just like predictive text on your cell phone, the Key-Counter could try to suggest words as you're typing them.

But wait - if I can receive keys from the keyboard and send them to the computer (and have text display on the screen), can I not use the sending of various keys to the computer as debug statements? I could have an entire menu pop up without having to open a terminal window or other software. Anywhere I can type, I can get the Key-Counter to display its information. ``` Key presses today: 000003413 Rather than using a UART and a troublesome RS232 circuit and terminal window,
```

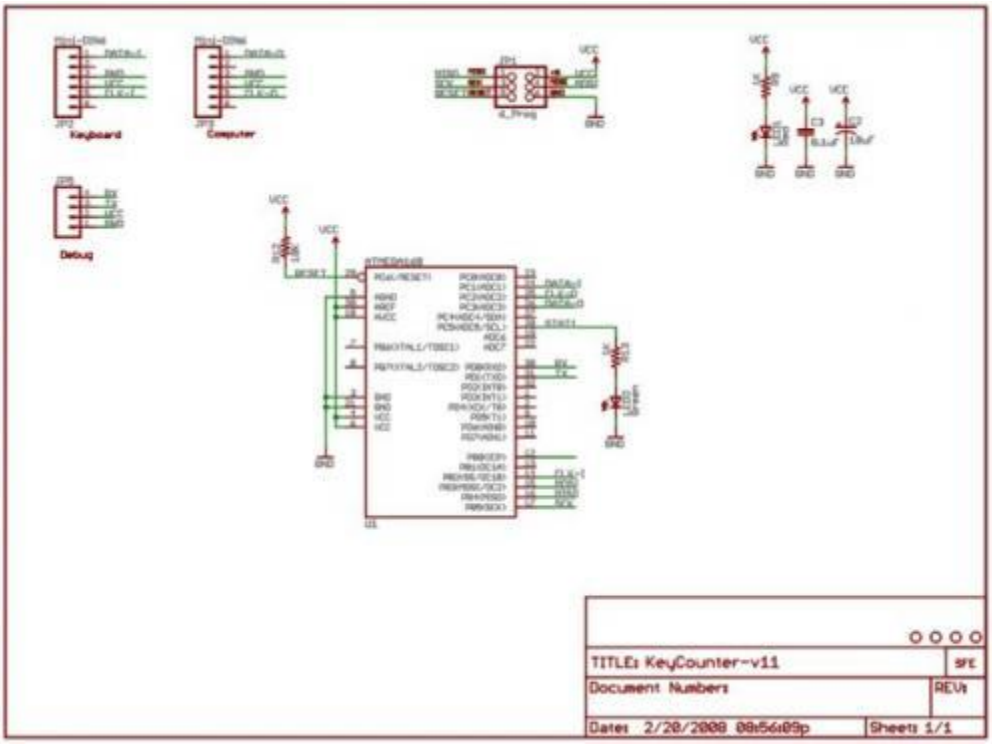
I could create a board that did analog to digital conversions and then report those values over the PS2 port. This board could effectively type the data straight into a spreadsheet. Nifty.

Once you have control of the PS2 commands zipping back and forth, the ideas start to flow. For now, let's just try to see how many keys are pounded in a day.

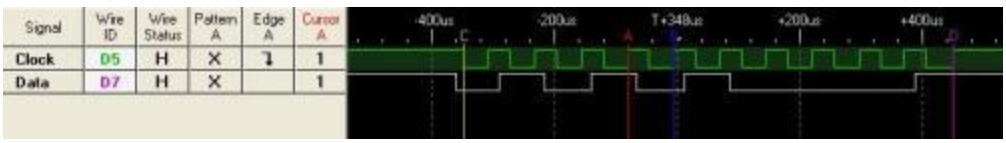
``` Key presses today: 000005866

Key-Counter:

- Firmware
- Schematic

---

**PS/2 Protocol**

Where does one find out information about how the keyboard communicates with the computer? Google of course. A quick search of 'ps2 protocol' brings up the holy grail of PS2 information. Computer-Engineering.org has done a great job of describing the electrical and physical specification of the PS2 protocol.



Key-Counter under interrogation. You can see the purple keyboard on one end, the white PS2 cable going to the computer, the gray and red AVR-PG2 programming cable (with 10-pin to 6-pin converter PCB). I also have 5 wires connected to my LogicPort logic analyzer.
```



Key-Counter Layout

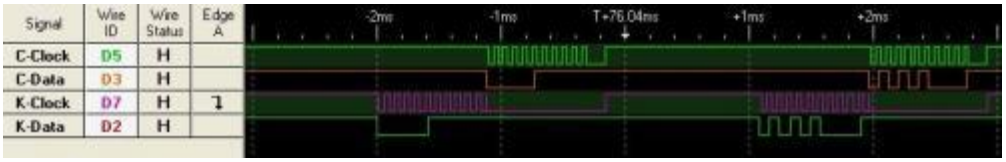


The electrical connections between the keyboard and computer were relatively straight forward. I'm not going to go over the PS2 protocol in depth - it's all spec'd out in the site listed above.



Here is the 'q' key being pressed.

Decoding the basic key presses from a keyboard is simple enough: wait for the clock to start toggling and record the data bits.

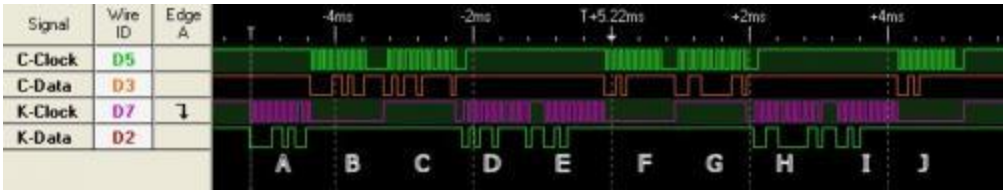


The 'q' key being released

What ever is heard coming from the keyboard (0xF0 followed by 0x15), the Key-Counter should pass it on to the computer.

Notice how the K-Clock line is pulled low for nearly 1ms after we receive 0xF0 from the keyboard. This is the Key-Counter pulling the K-Clock low. While the Key-Counter holds K-Clock low, it sends out the command to the computer. By holding K-Clock low, the Key-Counter is telling the keyboard that it (the computer) is busy and should not send any commands until the clock is released. Otherwise, we've found that newer keyboards send out commands with as little as 10uS in between commands. Since the Key-Counter doesn't buffer or handle interrupts, we have to tell the keyboard to sit tight for a second while we pass the last key press out to the computer.

Now if this is all we wanted to do (read key presses and pass them on to the computer), then we are done. What I really wanted was a small device that could hang out on my PS2 port. The problem arises when my computer turns on. The computer sends a series of commands to the keyboard to configure it. Host-to-device communication is very different and poses a few timing problems. My goal was to create a device that could handle any type of low-level communication. So the next step was to figure out the host-to-device (computer to keyboard) communication.



Pressing the 'Caps Lock' button

As you can see above, it can get pretty knurly.

A) The scan code of 0x58 (CAPS) is sent to Key-Counter when I press the 'Caps Lock' key

B) Key-Counter pulls the K-Clock line low and sends the 0x58 command out to the computer

C) The computer holds the C-Clock line low (initiating host-to-device communication) and sends the Key-Counter the command 0xED (Set/Reset LEDs)

D) Key-Counter passes along the 0xED command to the keyboard (host-to-device)

E) The keyboard sends 0xFA ('Acknowledge) to the Key-Counter

F) Key-Counter passes along 0xFA (Ack) to the computer

G) Computer responds with 0x06 (0b.0000.0110 = Caps Lock LED On, Num Lock LED On, Scroll Lock LED Off)

H) Key-Counter passes along 0x06 to the keyboard

I) Keyboard responds with 0xFA (ack) to the Key-Counter

J) Key-Counter passes 0xFA (ack) to the Computer

And the caps lock LED is lit! When I lift my finger from the Caps lock key a few hundred milliseconds later, there are another two commands (0xF0 and 0x58) indicating that I have released the caps lock key, but the computer doesn't send anything back to the keyboard.

So the device-to-host and host-to-device communication both work! The final test is to see if the Key-Counter will hinder the BIOS boot up process. This was very tricky because I needed a second computer to run the logic analyzer. I found that the computer will ask the keyboard to reset (0xFF) itself until the computer gets the response it expects from the keyboard. Because the Key-Counter sits in between, it can cause some glitches with timing and command collisions. To get around this, I force the Key-Counter to respond directly to reset commands rather than passing them on to the keyboard. This allows the keyboard to power on and respond to anything the computer throws at it, except for the reset command. This fixes the problems with boot up and allows the BIOS to think that everything is fine. The keyboard is configured by the BIOS and the computer starts loading the OS.

I used a LogicPort to analyze the various data streams. $400 is pretty steep for a hobby project, but I believe I am using about 4% of its capabilities. I'm not endorsing it - it's the best logic analyzer I've used simply because it is the only analyzer I've used. It did the job very well and without some sort of an external analyzer, it would have taken me 5-10 times longer to get the protocol figured out. The LogicPort is very handy to have when your PS2, I2C, serial, or SPI protocol starts giving you fits.

``` Key presses today: 000011661

I should go type some emails.

March 2nd, 2008
```