One frustrating aspect of firmware or kernel development on commodity hardware, such as cheap evaluation board or production devices, is the necessity of power-cycling the target device to reboot it every time the developer needs to load and run a new software build.

It sometimes happens that a development board is designed with proper management electronics to ease software development or automated testing, but in most cases the developer has reset the board manually, and sadly quite often reset buttons are unaccessible or just non-existent, requiring the developer to unplug and replug the power cable. If this ends up in your workflow and at the end of the day your fingers hurt, something is wrong.

This project is a small AVR/V-USB based board to control the power supply of development boards and other low voltage and USB powered devices. It allows to program a sequence of events for the output ports, has LED indicators for port status, and additionally provides power measurement on both USB and main power channels, and uses a bootloader for easy firmware upgrade… All in a solid and funny looking Hammond blue box!
Main Microcontroller

The brain of the board is the usual ATmega168 microcontroller, powered from the “control” USB port at 3.3V with a linear regulator, and clocked at 12MHz.
The firmware is based on the V-USB library to provide easy host connectivity. The protocol is designed as a vendor specific USB device, with just some custom control requests to enqueue events, clear the queue, query the status and reset the device. Use the source!

**Bootloader**

The microcontroller implements a customized version of bootloaderHID from obdev.at. The bootloader uses the control USB port, and is modified to run for some seconds every time the microcontroller exits from reset. This means that to upgrade the board firmware, the only operation to do is to issue a reset command, or power cycle the board itself if the current firmware is broken… So, the board itself is unbrickable, and allows automated firmware upgrade as long as you don’t upload a completely broken firmware!

**USB Ports**

The two USB channels (usb1 and usb2) have two couples of mini-USB to USB host ports, with straight-through data and common ground.

The USB supply line of each port goes through a TPS2552 precision adjustable power distribution switch. The TPS provides configurable (via resistor) overcurrent protection and power control. Additionally, each power line has a shunt resistor connected to an INA219, an I2C bidirectional current and power monitor, that provides power consumption information to the microcontroller.
The power monitor is useful to debug battery charging algorithms or to profile the power consumption of an USB device.

**Main Power Port**
An additional power line using terminal block connectors allows to control a generic power supply line for the target board. The limitation is the absolute maximum voltage of the INA219, and that is 26V, and the fact that the negative terminals are connected together with USB ground and shielding.

![Power Monitor Diagram](image)

The generic power line uses a small TE SMD relay, rated for a current of up to 2A per contact. The two contacts are wired in parallel, and eventually a high current version of the same relay is available.

Again, a shunt resistor and an INA219 are used to provide power consumption measurement.
An interesting alternative usage for this port would be to connect a battery instead of a normal power supply. As the INA219 is bidirectional, that would allow to measure the whole charge/discharge curve of the system.

**Additional I/O**

If you are really REALLY unlucky, your development board requires you to close a jumper or press a button to enter some kind of “programming mode”, or simply to turn it on.

In this case, you should really complain to the hardware engineer who designed it, but this is not always possible, so the two additional I/O channels (io1 and io2) have an isolated output that just shorts when active, and can be used to simulate the pressure of a switch, or just as an additional power line.
Box

This kind of circuits are usually used in a mess of tangled cables and other bare boards, and is not going to survive for long without a proper box.
So, the PCB fits nicely in an Hammond 1551K Box, and if you buy the translucent version (marked as 1551KTBU), you will be able to see the status LEDs on top and under each port… Nice!
**Firmware Operation**

The firmware is quite simple, and the most important thing is an USB `CUSTOM_RQ_EV_PUSH` control request to enqueue events on a queue periodically processed in the main microcontroller idle-loop. A dedicated timer is configured to give precise delay, and each event can be specified to be delayed from the precedent, allowing to program complex power on/off sequences.

The other important thing is the `CUSTOM_RQ_STATUS` control request to read the output statues, and the instant power consumption from the INA219 ICs.

**Control Application**

The userspace application is a `libusb` based application (named `pm`), that is easy to script and allows to control all the features of the device, including printing the actual status.

That's the syntax help of the application:

```
$ pm -h

Usage: pm -h

    pm -R

    pm [options]

options:

 -h       this help
 -R       reset device
 -s       show status
```
-w     wait delay in tenth of seconds
-C     clear event queue
-e list enable listed i/o (comma separated)
-d list disable listed i/o (comma separated)

valid i/o: usb1,usb2,power,io1,io2,all

And this is an example of the status output:

$ pm -s

    USB1  USB2  POWER*  IO1  IO2
voltage_in:     5161   0  4130
voltage_out:    5148  920  4172
    current:      1   1  203
    power:       0   0  860
    fail: 00

Complex events can be programmed by calling pm for each event, specifying a delay for each one using the -w flag and a number in tenths of seconds, as in:

# turn everything off
pm -d all

# enable usb and power after 1/2 second
pm -w 50 -e usb2,power

# turn usb2 off after 6 seconds
pm -w 600 -d usb2

# turn usb2 back on after 1/2 second
pm -w 50 -e usb2

That's it! No complex operation and easy port mnemonic!