**Quabo Firmware Release 10.0**

**Panoseti Multi-Quabo Design and firmware version readback**

**Firmware file quabo\_0100\_2295CF16.bin**

Wei/Rick January 28, 2020

This is the first official multi-quabo firmware release. It also has the firmware readback feature, and maintains the remote programmability of version 9.x. We’ve gone back to the simple focus motor control, since the focal plane assembly doesn’t yet have limit switches

**Multi-quabo**

One to four quabos may be plugged into a Mobo, and are individually addressable. They must be installed in sequential sockets, starting at J1. The one plugged into J1 is called Quabo 0, etc. The two LSbits of the IP address are set by position:

J1 00

J2 01

J3 10

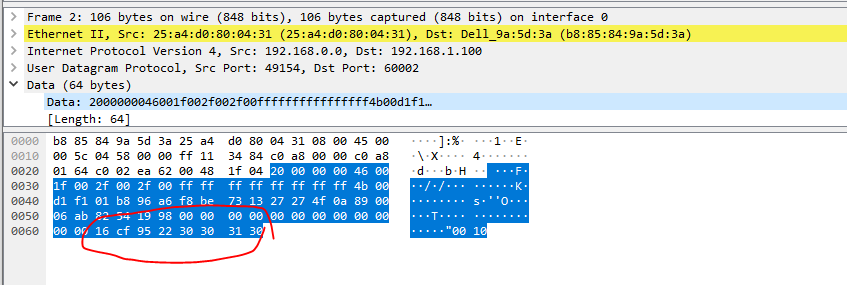
J4 11

(Note that this is different than what I wrote in a previous release. We did the obvious thing of swapping the bit assignments to make the IP addresses sequential.)

The rest of the IP address bits are set by the 8-bit jumper block on Mobo.

**Firmware Version Readback**

The name of the new .bin file (the FPGA configuration file that goes in the flash) is *quabo\_0100\_2295CF16.bin*. The ‘0100’ is just the version number. The appended 8 hex digits form a 32-bit number that is generated at the time of the bin file generation, based on the “epoch time” read from the computer operating system. The two parts combined form an 8-byte sequence (4 ASCII characters- in this case 0x30, 0x31, 0x30, 0x30- plus the 4-byte time code) which is read by the Microblaze processor on boot-up and are inserted in every housekeeping packet. The purpose of this is to provide an easy-to-read identifier, plus an automatically-generated part, which will prevent us from generating two compiles with identical names. Details of this process are in the Appendix. Here’s the Wireshark output, showing the 8 bytes at the end of the housekeeping packet. We can discuss what to do with this info.



**Focus Motor Control**

This is now just the “move up or down x steps” function. We introduced a more sophisticated control in v9.1 but we’ve undone that, to be restored when we get the focal plane assembly together with limit switch, etc.

**Loading up the firmware**

We need to replace the GOLD firmware version now, because the previous version, lacking the ability to separately address the four quabos on a Mobo, was unable to load new firmware to a quabo plugged into J2, J3, or J4 of Mobo. We shouldn’t need to do this again. The new GOLD version is called

*quabo\_0100\_GOLD\_2295DBE1.bin*

If the quabo has the old GOLD version (as I think all do at this point), you need to do the following command, as described in the release notes for v009.0:

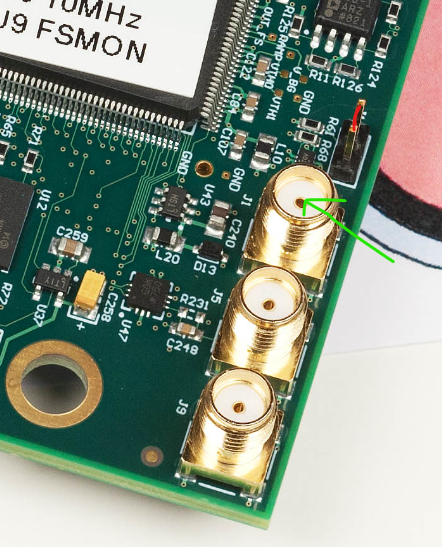
**client.put\_bin\_file(‘quabo\_0100\_GOLD\_2295DBE1.bin’, 0x0)**

(if you don’t have the old GOLD version, then you need to connect a JTAG cable and program the board as described in v009.0)

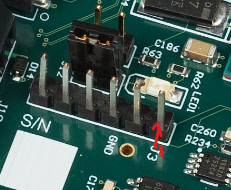
Then you need to load up the new operating (“silver”) version, like this:

**client.put\_bin\_file(‘quabo\_0100\_2295CF16.bin)**

**Testpoints, SMAs and J3**



The J1 SMA is now 1PPS out- these should be within 1ns of each other from Quabo to Quabo. J9, in the corner, is still the Maroc fast shaper output. J5 is unused.



Pin 1 pulse\_height\_trigger output: the OR of all of the triggers of the 256 channels, masked by the internal (to the FPGA) mask bits and synchronized to the 100MHz clock

Pin 2 unused

Pin3 WR\_UART\_RXD(3.3V, must connect TXD from your module)

Pin4 WR\_UART\_TXD(3.3V, must connect RXD from your modul)

We can use this uart port to access WR shell, and confirm whether WR part works well.

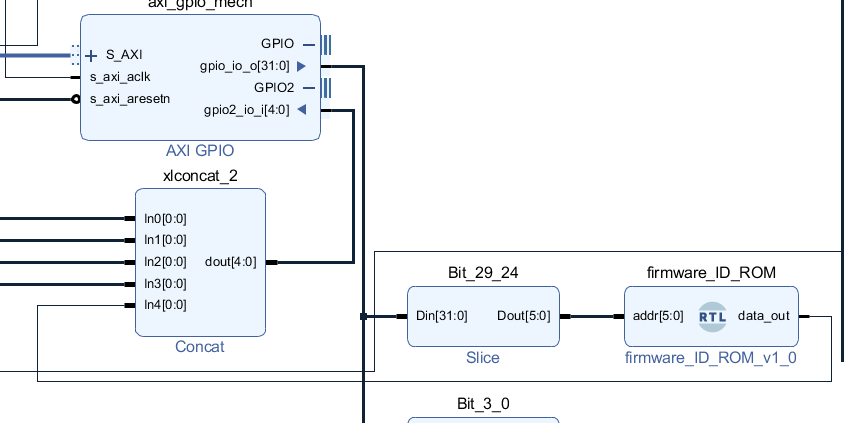
Pin5 MB\_UART\_RXD(3.3V, must connect TXD from your module)

Pin6 MB\_UART\_TXD(3.3V, must connect RXD from your module)

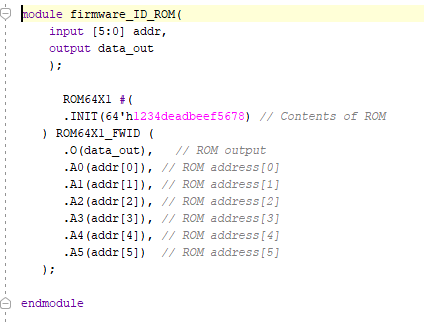
**Appendix: Firmware version readback scheme for Panoseti**

We’ve wanted a way to assign a unique identifier to the compiled firmware which can be read back from the FPGA, so that we can be sure what version is installed in each FPGA. Additionally, this should provide some protection against the user just forgetting to increment the version number. Here’s what I came up with.

In the “hardware” design (the gateware that sets all the guts of the FPGA) I put a block called firmware\_ID\_ROM, which interfaces to a GPIO via 6 outputs and 1 input:



This is just a single 6-bit LUT configured as a 64by1 ROM:



After compiling the hardware design, we can open the implemented design and change the 64-bit INIT value to something meaningful that we can read back. I’ve written a tcl script (see below) to do this.

The value is read in MicroBlaze software with a simple loop, and placed in two 32-bit global variables called FWID\_0 and FWID\_1:

**void** **get\_FWID**(**void**)

{

s8 addr;

**for** (addr = 31; addr >= 0; addr--)

{

XGpio\_DiscreteWrite(&Gpio\_mech, GPIO\_OUT\_CHAN, (addr<<24) | (focus\_limits\_on <<23) | (shutter\_power<<22) | (shutter\_open<<21) | (fan\_speed<<17));

FWID\_0 = (FWID\_0<<1) | ((XGpio\_DiscreteRead(&Gpio\_mech, GPIO\_IN\_CHAN) & 0x10) == 0x10);

}

**for** (addr = 63; addr >= 32; addr--)

{

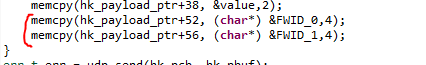
XGpio\_DiscreteWrite(&Gpio\_mech, GPIO\_OUT\_CHAN, (addr<<24) | (focus\_limits\_on <<23) | (shutter\_power<<22) | (shutter\_open<<21) | (fan\_speed<<17));

FWID\_1 = (FWID\_1<<1) | ((XGpio\_DiscreteRead(&Gpio\_mech, GPIO\_IN\_CHAN) & 0x10) == 0x10);

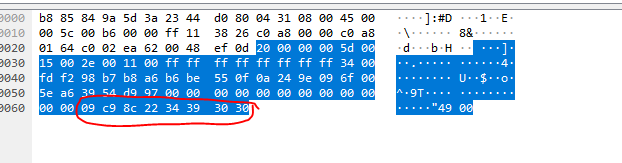
}

}

We can then put these 8 bytes into the housekeeping packet (we’ve reserved 64B of payload but are only using 44 for housekeeping) by adding these two lines to the SendHousekeeping() routine:



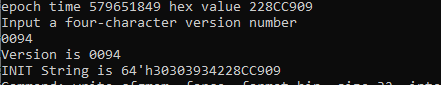
Result shows up at the end of the HK packet, in Little-Endian format:



Here I’ve set the version number to “0094”. See below for the meaning of the other 4 bytes

**Generating the .bin file**

I’ve written a script called create\_binfile.tcl. After the user runs through implementation, the script opens the implemented design and asks the user for a 4-character version code. Then it gets “epoch time” (the number of seconds since 1970 Jan 1), subtracts 1e9 (to get a <= 32b result) and converts it to hex, concatenating with the 4-byte version code:



Then it generates the .bin file (for writing to flash) with the name “quabo\_” plus the version code and timecode, eg, quabo\_0094\_228CC909.bin. The idea being that even if we use the same number twice, we can tell from the unique timecode that the file is different. And we can match the value read back to the name of the file programmed.

Note that if we always use this method for generating flash files we’ll change the version number and timestamp even if we’re only changing MicroBlaze software, and not otherwise changing the gateware, which is good.