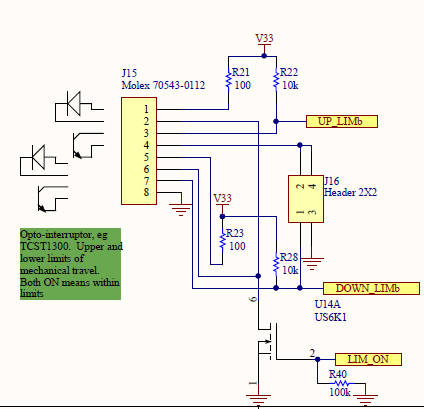
**Quabo Firmware Release 009.1**

**Panoseti Focus Stepper Control**

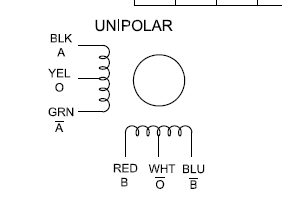
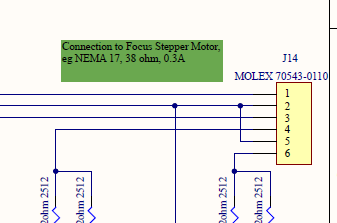
RR January 7, 2020

I’ve implemented a new focus control logic, as recommended by Franklin, where we can command the system to calibrate to a zero position (based on the limit opto-interruptor) or to move to a position relative to that zero. Firmware file is **quabo\_v0091.bin**, installed per the instructions in release 009.0 (only the 0091 version needs to be loaded; the GOLD version stays 0090).

The wiring needed for this is as shown here. The single opto-interruptor is wired to Mobo J15 pins 1 to 4; pins 5 to 8 are unused. A jumper needs to be installed on J16 between pins 3 and 4.



The stepper motor is wired like this to J14



Pin Winding Color

1 A Black

2 VCC Yellow

3 /A Green

4 B Red

5 VCC White

6 /B Blue

The control logic is implemented in MicroBlaze software. After each commanded move the current position is stored in non-volatile flash RAM so that after a power cycle the processor will know where it’s starting. A timeout of about 10ms per stepper pulse is implemented in gateware so that a software bug can’t cause a long-term high current draw (the stepper motor draws about 600mA from +24v; the motor and dropping resistor on Mobo would get hot if left on for a long time, but no harm would be done). I don’t think it’ll be an issue but the flash has a minimum lifetime of 100,000 write/erase cycles (we erase/write once after the operation is compete, not after every step).

The system is commanded by sending a UDP packet with the first byte = 0x05. A 16-bit value sets the focus target, in number of steps relative to the zero calibration (one step is 1/200 of a revolution, 4 microns of movement with the 0.8mm leadscrew). If the value is set to 0 the system recalibrates by finding the point at which the lower limit is hit, then backing off a fixed amount from that.

There are four constants STEPONTIME, STEPOFFTIME, ENDZONESIZE and BACKUP which tell the software the stepper pulse ON and OFF times, how many steps it takes to get out of the zone where the limit switch is dark but the mechanical stop has not been hit, and how far to back off after finding the point at which the limit switch is illuminated again. We’ll probably want to hard-code these in MicroBlaze software, but I’ve made them software-settable at this point for mechanical debug. The interface is described here, and is implemented in **control\_quabo\_v08.py**, with *endzone*, *backup, step\_ontime, step\_offtime* hard-coded in python (so easily changeable). The value of 10,000 for step\_ontime and step\_offtime give an approximate 2ms stepper pulse, and a 2ms off time between pulses, similar to the Arduino setting.

Set Stepper command Packet Length 64 bytes

Payload offset Byte Contents

0 0x05

4, 5 focus target, #steps, unsigned 16b number, 0 for recalibrate

8 fan speed, 0 to 15

10,11 ENDZONESIZE

12,13 BACKOFF

14,15 STEPONTIME

16,17 STEPOFFTIME

Following completion of the stepper operation a response packet is sent back, as follows. My python code does not receive or interpret this, but it can be viewed in Wireshark, and contains four 32b signed integers, in Little-Endian format, which may be useful for debug.

Set Stepper Response Packet Length 64 bytes

Payload offset Byte Contents

0 0x05

4-7 stored position, read from flash at start

8-11 commanded steps- the net total steps commanded during the operation

12-15 final position- should equal target

16-19 status vector:

Bit 0: recal was attempted

Bit 1: recal was started in upper endzone, so we hit the mechanical stop

The stepper operation can take many seconds, and the processor is single-threaded, so it can’t respond to other commands, take science data, or output housekeeping during this time.

The Stepper Response packet is sent to the housekeeping UDP port, so the program which logs housekeeping data needs to check the first byte to reject this packet (my hk\_log\_quabo\_v04.py had this problem; I fixed it in v05).

**Appendix: MicroBlaze C code**

//Move the focus stage to 'target' above the zero point, 0<= steps <= MAXSTEPS

//Steps = 0 tells the routine to recalibrate

//statptr points to int array giving several status values:

// initial (stored) position

// #steps actually moved

// final position

// status (whether a cal was done, whether a hard limit was hit)

//Return value is 0 if all is well, -1 if problem

**#define** MAXSTEPS 50000

//#define ENDZONESIZE 300

//#define BACKOFF 200

//#define STEPONTIME 10000

//#define STEPOFFTIME 10000

**int** **set\_focus**(u16 target, **int**\* statptr)

{

**if** ((target > MAXSTEPS) || (target < 0)) **return** -1;

//Turn on the limit switch LED

focus\_limits\_on = 1;

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

//Read from flash where the stage was left. If no stored value, return is -1

**int** position = get\_stored\_focus();

u8 last\_phase = 0;

**if** (position != -1) last\_phase = position & 0x3;

//We'll return that value in the statptr array, and report it back to the host

\*statptr = position;

//moved will hold the number of steps that the move\_stepper fcn has commanded

// (not the number passed to it, if it hits an electrical limit)

**int** moved;

//move\_sum will hold the total number of steps commanded

**int** move\_sum = 0;

//move\_stat will hold other status flags

**int** move\_stat = 0;

**if** ((position < 0) || (target == 0) || (position > MAXSTEPS))

//either we're commanded to recal (target = 0) or we don't have a stored position

// (position < 0), or we don't have a stored position that makes sense,

//so we need to recal

{

//Set the LSB of move\_stat to indicate that a recal was attempted

move\_stat |= 1;

**if** (!focus\_limit\_dark())

{

//We're in the middle range. Step down until we hit limit

move\_sum += move\_stepper(0 - MAXSTEPS, last\_phase, 0);

//And a bit more

move\_sum += move\_stepper(0 - ENDZONESIZE, last\_phase, 1);

}

//We're at one of the limits

//Maybe we're at the bottom limit, try to move up

moved = move\_stepper(ENDZONESIZE, last\_phase, 1);

move\_sum += moved;

**if** (focus\_limit\_dark())

{

//Must have been in the upper endzone and hit the upper mech stop,

//Set bit 1 of move\_stat to indicate this

move\_stat |= 2;

//Move down out of the endzone, with limits overridden

move\_sum += move\_stepper(0 - ENDZONESIZE, last\_phase, 1);

//Then move all the way to the bottom limit, with limits enabled

move\_sum += move\_stepper(0 - MAXSTEPS, last\_phase, 0);

}

**else**

{

//We moved out of the bottom endzone, back off a bit

move\_sum += move\_stepper(BACKOFF, last\_phase, 1);

}

//We've completed calibration to 0, so call this position 0

position = 0;

}

//We've recalibrated to 0 or didn't need to. If target = 0, we'll fall through and return

//Else we had to recal bc we had no stored focus, so move to the desired point

**if** (target > 0)

{

//We have the stored position, or we've recal'ed to 0

//move\_stepper returns the number of steps actually moved

// which will be less than that commanded, if a limit is hit

moved = move\_stepper((target - position), last\_phase, 0);

position = position + moved;

move\_sum += moved;

}

store\_focus(position);

\*(statptr + 1) = move\_sum;

\*(statptr + 2) = position;

\*(statptr + 3) = move\_stat;

focus\_limits\_on = 0;

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

**return** 0;

}

//Move the stepper a number of steps,

// stop when limit is hit if limit\_override = 0

//We pass last\_phase to the fcn so it can start with the right pulse phase

**int** **move\_stepper**(**int** steps, u8 last\_phase, u8 limit\_override)

{

**volatile** **int** delay;

**int** moved = 0;

u8 direc = (steps>=0);

**int** stepnum = last\_phase;

**if** (direc == 1)

{

**while**(stepnum < (steps + last\_phase))

{

**if** (!limit\_override && focus\_limit\_dark()) **break**;

stepnum++;

moved++;

ChangeStepper(stepnum & 0x3);

**for** (delay = 0; delay< STEPONTIME; delay++);

ChangeStepper(4);

**for** (delay = 0; delay< STEPOFFTIME; delay++);

}

}

**else**

{

**while**(stepnum > (steps + last\_phase))

{

**if** (!limit\_override && focus\_limit\_dark()) **break**;

stepnum--;

moved--;

ChangeStepper(stepnum & 0x3);

**for** (delay = 0; delay< STEPONTIME; delay++);

ChangeStepper(4);

**for** (delay = 0; delay< STEPOFFTIME; delay++);

}

}

**return** moved;

}

**void** **ChangeStepper**(u8 phase)

{

//Two windings are energized for each phase

//If phase is not 0, 1, 2, or 3, turn off all windings

u8 mask = 0;

**if** (phase == 0) mask = 0x3;

**else** **if** (phase == 1) mask = 0x6;

**else** **if** (phase == 2) mask = 0xc;

**else** **if** (phase == 3) mask = 0x9;

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

}