790-00002
Revision 1

SW/FW Specification

M2 Focus Mirror Controller

October 5, 2010

Rich Lobdill

Software

Computer Interface

Similar to previous LCO-designed BlackFin-based controllers, there are two computer interfaces provided: Ethernet (10/100 BaseT) and USB. The user interface is Ethernet, the USB connection is for development to give access to the U-Boot and uCLinux console on the BlackFin.

Motion Control Axis

The M2 controller board controls three stepper motors that are deployed as linear actuators. Each axis has a working travel range of +/- 3mm (+/- 4mm max) . The motors are placed in a circular pattern spaced at 120 deg. They provide the tip, tilt and z-axis displacement of the M2 mirror as it focuses the light coming from the primary mirror going to the main imaging instrument.

For each axis, there are both a positive and negative limit switch.

Also, for each axis there is a linear encoder with a travel of about 12mm that independently measures linear displacement. Each linear encoder has an index mark at a point roughly midway through its travel. A complicating factor in the design is that all three encoders will register movement for a movement of any one of the motor axis. Another of the properties of this system is that the index mark on the encoder will not directly map to the positive and negative travel limit switches of its associated stepper motor.

Ben H. will be the resource for providing the detail on how to reconcile the movement of motors with the movement of the linear encoders.

PMD Motor Controller (Indexer) I/O

Since this is an application using a SuperFin module, the motor controller is a 4-axis PMD chipset (MC55420). The communication with the PMD chipset is via CAN bus from the BlackFin. The following I/O on the PMD chipset will be used for each axis:

PMD I/O
Step (Pulse)
Direction
Position Encoder (quadrature)
Index

Positive Limit
Negative Limit
At Rest

Note: The Home input is not used in this application

Heidenhain Linear Quadrature Encoder (ST 1278)

The linear encoders provide measurement of linear displacement of three points around the mirror assembly. From a software point of view, this encoder should function very similar to a rotary encoder except that it is directly measuring linear displacement.

Maximum Travel	12 mm
Index mark	Approx 5 mm below upper stop*
Measurement resolution	1 um (TTLx5 interpolation)

* Upper stop is fully compressed position of encoder

IMS Motor Driver

The task of taking the step and direction signals and driving the stepper motor will done by an IMS IM481H Hybrid Microstepping Motor driver. Some of its important features:

- 2.1 A peak (1.5 Arms) current per phase @ 24 V
- Microstepping to 256 steps/step (binary and decade microstepping choices)
- 2.5 MHz max clocking rate (max rate for PMD controller is 5MHz)
- Drive current programmed by DAC voltage (each axis independent)

Run Current and Hold Current

Three user defined values will be required

1. Run Current (Peak) Range: 0.2 to 2.10A (resolution 0.001 A)
2. Hold Current (as above) (Hold always less than or equal to Run)
3. Hold Current delay time: Range 0 to 2.047 sec (1 ms resolution)

These inputs will be passed to the FPGA with the following integer translation:

User input range	Integer value to FPGA (decimal)
0.2 to 2.1 A	200 to 2100
0 to 2.047 sec	0 to 2047

The DAC address decoding is such

Motor Axis	DAC Channel	DAC address
1	0	020h
2	1	021h
3	2	022h

DAC Scaling (16 bit)

Data	DAC voltage out	IMS Driver current setting
0	0	0
00FFh	2.048 V (1/2 scale)	2.048 A
8340h	2.100 V	2.100 A (max setting)
FFFFh	4.096 V (full scale)	Not used in this application

The firmware will be responsible for deciding when the DACs will be set to the run current value and when they will be set to the hold current value. See the Firmware Specification portion of this document for details.

Mircostepping

The microstepping value is written to an FPGA register and is common to all axis

FPGA address	Data
000h	D[3:0]

Data	uSteps/Step	Steps/rev (1.8deg motor)
0h	2	400
1	4	800
2	8	1600
3	16	3200
4	32	6400
5	64	12800
6	128	25600
7	256	51200
8	5	1000
9	10	2000
A	25	5000
B	50	10000
C	125	25000
D	250	50000
E	1	200

F	180	36000

Motor Driver Status

There are two bits of driver status. The 'Full Step" bit is a pulse that occurs every time the motor transitions through a full step position, if this is used it will probably be in conjunction with FPGA firmware. The 'Fault Output' line should be regularly monitored. Fault will be enabled for either an Over Current or Over Temperature condition exists. This condition can only be cleared by either cycling the power or toggling the reset bit (presently available only via push-button on PCB).

FPGA address	Data
01Eh	D[5:0]

D5	D4	D3	D2	D1	D0
Fault Axis 3	Full Step Axis 3	Fault Axis 2	Full Step Axis 2	Fault Axis 1	Full Step Axis 1

Infrared Thermocouple Readout

There are three channels of thermocouple 'non-contact' temperature readout

Temperature Measurement Identifier	ADC Channel	Programmed Attenuation
1	0	0 dB
2	1	0 dB
3	2	0 dB

Referring to the "Superfin Software Interface Document" (M. Inbar, July 2010), the Gain through the attenuator for each channel will be equal to 1 (i.e. no attenuation). The ADC channel scan mode will be 'Default".

ADC outputs are 16 bit values presented in two's complement format. That is, the integer range of the ADC output should be:

Range of ADC Readout
+32,767 to -32,768

To transform ADC counts into measured temperature use the following equation:

ADC counts to displayed temperature
Temp (degC) = -0.00126936 x (ADC value) – 0.8896 degC

Display Resolution of measured temperature
0.001 degC

Conventional Temperature Sensors

There are three channels of temperature readout that generated via LM335 sensors used in many previous instruments. The range of the measurement is from 0 to 409.6 degK

Temperature Measurement Location	ADC Channel	Programmed Attenuation
M2 Electronics	4	0.63 (4.01 dB)
Truss 1	5	0.63 (4.01 dB)
Truss 2	6	0.63 (4.01 dB)

Range of ADC Readout
+32,767 to -32,768

ADC counts to displayed temperature
Temp (degC) = (-0.0099206) x (ADC value)-273

Display Resolution of measured temperature
0.01 degC

Fan Speed Setting and Feedback

There is a single fan mounted on the top of the M2 assembly. The speed of the fan will be user programmable in the sense that the user can program a speed value between 0 and 100%. The user will have an On/Off control button in addition to the speed value setting. The actual speed of the fan (in RPM) will be measured by the firmware and will be reported to the user. An error condition will be generated if the RPM reading is zero under the following conditions:

- On/Off control is set to On
- Speed setting greater than zero.

The hardware speed control will be realized by applying a pulse width modulated (PWM) signal to one of the GP Output pins (GP_Out_D4) of the SuperFin.

I am not clear as to what FPGA read/write registers will be used to implement this feature.

Parameter	User Setting	Resolution

Fan Speed Setting	0-100%	1%
On/Off	On/Off	
Fan Speed Measurement	0 to xxx RPM	10 RPM

Firmware

This specification covers firmware requirement specific to the M2 Focus Mirror Controller. It does not cover the general purpose firmware such as communications between FPGA and BlackFin, interface to the ADC input channels, or how to remotely program the FPGA via BlackFin.

Fan PWM

A single fan is hooked to SuperFin signal GP_Out_D4. Since this signal is driven by an open drain MOSFET it can directly drive the fan. The fan requires speed control which will be done via PWM. The software will have a user speed variable of 0 – 100% which will directly relate to the duty cycle of the drive signal.

Waveform Period	100 ms
Duty Cycle Range	0 to 100%
Resolution	1%

Fan Tach / Alarm

The Fan Tach signal from the fan is connected to SuperFin input signal Opto_Input0. The firmware is to measure the period of that signal and store it in a register for the software to read. The value will ultimately be displayed to the user in units of RPM.

A Fan Alarm bit will be toggled in a status register under the following conditions:

- Fan PWM drive is set to a non-zero value
- Fan speed is being measured as zero

Stepper Motor Run/Hold Current

The IMS stepper motor drivers have programmable drive current that is set via voltage on one of the input pins. In this design, three DACs are connected to each of the three motor drivers. It will be the job of firmware to switch the DAC outputs between two user programmed settings that represent the drive current and the hold current for each axis. Additionally, there is a user programmed time delay (one for each axis) that represents the delay between the end of a motor move and the application of the hold current DAC value.

A discussion thread of this subject is covered under an email with the subject line:

Stepper motor drive/hold current... driven by SW or FW? (last updated 9/2/2010)

The first page of this thread is attached as an appendix to this specification

The outcome of this discussion is that it seems that the proper way to implement the feature is via firmware. In short, the firmware should take the three parameters from the software:

- Run Current (0.2 to 2.1 Amps)
- Hold Current (same range as above)
- Hold Current delay (0 to 2.047 seconds)

With these values, the firmware should look at the status of the ‘at rest’ signal for each axis. If ‘at rest’ is false, then the DAC should be set to the Run Current value. When a transition from ‘not at rest’ to ‘at rest’ takes place, the firmware should wait an amount of time equal to the Hold Current delay time and then set the DAC to the Hold Current value.

Motor Controller Interface

Conversion of balanced inputs to single ended.

Typical signals from encoders are balanced and need to be converted to single ended prior to being routed to the PMD motor controller chips.

Balanced input signals to FPGA (n has a value of 1-4 representing the 4 axis of control)
QuadAn(+)
QuadAn(-)
QuadBn(+)
QuadBn(-)
Indexn(+)
Indexn(-)

These signals need to be converted to single ended to generate the following FPGA output signals
QuadAn
QuadBn
Indexn

Perform the conversion following this table:

(+) Input	(-) Input	Output
H	H	H
H	L	H
L	H	L

L	L	L

Appendix 1:

Stepper motor drive/hold current... driven by SW or FW?

Rich Lobdill show details Sep 1

to Michael, Eric, Ben, Joe

Okay, I could use some help here.

We will be using a motor driver that I have wired up such that we can program the motor drive current via a DAC voltage. I want the user to be able to define the following parameters for each axis independently:

1. Motor Drive Current
2. Motor Hold Current
3. Time delay between the motor stop and the onset of the hold current (in the range of zero to maybe 1 or 2 seconds)

The switch from one current to the other is done by changing the DAC voltage.
The time delay feature uses a signal from the the PMD motor controller chip called 'At rest' which is enabled when the motor is not moving.
The 'at rest' signals are routed from the PMD chip to FPGA I/O pins.

So, I can see two ways to go about this.

Software-centric:

Ben keeps all the user input data and watches an FPGA register that holds the 'at rest' status. Just prior to a motor move command he sets the DAC to the drive current value. He then looks for a change in the status of the 'at rest' signal. When 'at rest' is enabled, he waits the programmed amount of time and then sets the DAC to the hold current values.

Firmware-centric:

Ben takes the user inputs, coverts them to integer and sends them to the FPGA for storage. The FPGA then takes on the task of setting the DACs by watching the 'at rest' signal and providing the correct delay between the enabling of 'at rest' and the setting of

the DAC to the hold current value.

It seems during the writing of this that the Firmware approach would be the most efficient but I see it might take one extra signal. I think we would want the drive current DAC value to be set and settled before the first step command made it to the motor driver. I think that means that Ben would need to send a signal to the FPGA to go ahead and set the DAC to the drive current level. I'm not sure how good the timing will be between sending a signal to the FPGA and sending a more command via CAN bus to the PMD motor controller chip.

Comments please.

Ricih