**System Overview**

The observer controls the instrument via a native OS X cocoa application running on the Magellan Clay observer's workstation. The front-end application performs simple command translation and communicates with the instrument control computer via a TCP socket connection. Plate files are copied to (and may be retrieved from) the instrument via a standard network share with drag-and-drop file operations in the OS X Finder.

The instrument control computer is a Beagleboard-xM (a 1GHz ARM 512MB RAM 16GB flash hard disk single-board-computer) running Angstrom Linux 3.2. This computer handles all communication with the outside world, runs the majority of the instrument control software, and controls spectrograph components (mostly via USB). The fiber imager & fiber projector; CCDs & shutter; and calibration unit lamps & arm are special cases. See their sections for details.

The instrument subsystems are described in the following sections and are categorized as follows: spectrograph table motion, slit motion, fiber imager, fiber projector, temperature & shock monitoring, battery backup, shutter (controlled directly by the CCD sidecar), Shack-Hartman, guider, plugging feedback, and calibration lamp unit.

Electronics enclosures are located on all three instrument components, with the primary electronics box mounted on one end of the spectrograph cart. It is connected to the conical mount enclosure via USB and the secondary cage electronics via Ethernet & TTL over multimode fiber.

The following systems are located within the primary electronics box or within the spectrograph enclosure.

**Spectrograph Motion** Each spectrograph uses eight axes: focus, disperser slide (used to switch between HiRes and LoRes modes), HiRes azimuth and elevation, LoRes elevation, filter exchanger (2 axes), and fiber imager pickoff. A Galil Motion Control DMC-4183 handles low-level details such as limit switches, encoders, backlash, interlocks, and calibration of these axes. We use the DMC-4183, as observatory staff is familiar with its operation.

**Slit Motion** We use a custom eight-axis stepper controller embedded in each fiber shoe to handle low level control of the tetris slits. It consists of a custom drive board mated to an Arduino Mega 2560 (an ATMEGA based microcontroller).

**Datalogging** The instrument is equipped with three customized Arduino Pros each equipped with an Adafruit Industries Datalogger Shield, digital temperature sensors, a three-axis accelerometer and an SD card. Four “C” cell batteries should power each datalogger for the life of the instrument. Each unit records temperature readings every minute and any impact or freefall event above a certain threshold. The instrument computer downloads and collates logged automatically whenever powered.

**Battery** The battery backup system consists on an off the shelf UPS connected to the instrument control computer. In the even of a power outage, should less than a preset amount of battery life remain the instrument will safely power down.

**Shutter** M2FS uses two Uniblitz CS-90 iris shutters, both driven by a multi-channel driver from the same manufacturer. The shutter is actuated by a level transition from the CCD sidecar electronics directly to the Uniblitz controller and is not typically controlled by the instrument computer. An arduino hooked to the controller’s override provides state monitoring and the ability manually open the shutter (MARIO, THIS IS ONLY IF YOU WANT IT, I KNOW WE TALKED ABOUT IT).

**Fiber Imager** The fiber imager is a custom high-speed (18kHz) imaging system used to image the fiber ends during plugging and slit positioning. When used in conjunction with the fiber illuminator the imager allows the instrument to compare fiber plug positions against a user-supplied list. The imager is also used to monitor fiber throughput during slit positioning.

The following systems are located in the Conical Mount electronics box.

**Guider** Pololu simple RC controller. Two hobby servos.

**S-H** Arduino for led dimmer. Pololu simple motor controller for lenslet motion.

**Plugging Feedback** M2FS uses a pair of computer speakers and a character display to provide auditory and visual feedback doffing the fiber plugging process. When used together the fiber illuminator & fiber imager provide real-time determination of any misplugged fibers. The instrument is thus able to notify the plugger as they make the mistake and, should they decide not to correct it, mark the fiber as misplugged in the FITS header.

These last two systems are in the secondary cage.

**Fiber Illuminator** Consists of a second Beagleboard-xM driving a Microvision PicoP projector. The projector’s horizontal and vertical synchronization pulses connected to the fiber imager via TTL over fiber. The BBxM is controlled by a direct Ethernet link (again over fiber) to the BBxM in the primary enclosure.

**Calibration Lamp Unit** The M2FS calibration lamps consist of XXX and are controlled by the observatory software. The calibration unit, although designed for M2FS, will be a facility component and thus not controlled in any way by the M2FS control computer.

Control concepts

The general concepts are as follows: 1) use standardized communications and interfaces whenever practical, 2) prefer premade cables to custom wiring, 3) use USB (see 1), use open-source and hobbyist electronics solutions whenever suitable.

System status

The system architecture is completely mapped out. Wiring for the spectrograph cart is 90% finished. Barring the Guider, parts are in hand for the conical mount enclosure (S-H, plugging feedback). Shoe electronics have been fabricated and verified.

Electronic components

* Beagleboard xM
  + 1GHz 512MB DDR ARM Single-Board-Computer
* 2 Galil DMC-4183 8 axis stepper motor controller
* 2 Fiber Shoe controllers
  + custom 8 axis stepper driver mated to an Arduino Mega
* 2 Uniblitz CS-90 Shutters
* 1 Uniblitz multi-channel shutter driver
* 1 300VA UPS
* 1 Fiber Illuminator
  + 1 Microvision PicoP PDK
  + beagleboard xM + projector module
  + Controlled by main computer over 100BASE-FX ethernet
  + Projector HSync and VSync monitored by Fiber imager via TTL-to-fiber
* 1 Fiber Imager
  + 8 Hamamatsu 18kHz line rate linear CMOS arrays to image fiber ends
  + Each set of 4 connected to a custom multiplexing analog front end (AFE)
  + AFE data is captured and formatted into standard webcam format via a custom FPGA-based expansion board for the BB xM.
* 1 Pololu Simple Motor controller for the S-H
* 1 Arduino for the S-H LED
* 1 pair USB speakers for plugging feedback
* 1 20x4 character display for plugging feedback
* 3 Dataloggers
  + Analog Deviced 3axis accelerometer
  + ~5 13 bit digital temperature sensors
  + Battery powered for life of instrument
* 1 Arduino For primary electronics box monitoring
  + lights in the box when door opens
  + soft-power switch
* 1 Guider Camera thingy
  + 1 composite video usb capture adapter (ASSUMING NO OCIW GUIDER)
  + Arduino controller hobby servo for guider filter

# Software

## Tests

zcat /proc/config.gz | grep DRV\_TWL4030  should show =y if =m kernel was built wrong

systemctl should list all the agents mentioned in the architecture section.

## Architecture

The control software is structured as in figure x. While the structure was

Each python program is a subclass of agent, which provides implements basic functionality.

The galil code consists of routines designed to safely mo

## System Requirements

The BBxM runs a system image that is built by the Angstrom Distribution’s OpenEmbedded toolchain. An M2FS overlay has been created such that when the M2FS image is built all required kernel patches and software are pulled in automatically (at least in theory). See the how-to document TODO should you ever need to do this.

At present, NetworkUPSTools is not done automatically and must be built by hand. Also connman-tests

## Installation

TODO

Clone the git repo, run install.sh or bitbake m2fsimage

## BitBaking (heaven forbid)

TODO

 compiling in changes to the angstrom kernel

## Outstanding issues

Ntp fails in systemctl (perhaps because conman isn’t pulling an IP

May not request an IP (does at UM, not at Carnegie)

CABLE=`/usr/lib/connman/test/get-services | grep “/net” | awk ‘{ split( $2 , a , “/” )} { print a[5] }’`

/usr/lib/connman/test/set-ipv4-method $CABLE dhcp

but this doesn't request another ip if the cable is unplugged

# FLS Imager

The imager consists of eight linear high-speed (~19.5kHz line rate) 512 pixel CMOS imager chips (Hamamatsu S10453), each on their own PCB. These are connected in sets of four to an analog front end based on the a four channel 8-bit ADC (AD9289). The two AFEs are connected to a Cyclone III (EP3C40F324C8) based camera expansion board, or “data formatter,” we have built for the Beagleboard xM. The general idea is that the FPGA is controlled via an I2C 16 channel GPIO expander from the BBxM and synchronizes the image capture with the FLS Projector via the horizontal and vertical synchronization signals from the projector. The FPGA provides clock and control signals to the AFEs & Hamamatsus and receives and formats the raw pixel data. Finally it transmits a composite image of the captured data to the BBxM using the the CCDC ISP interface configured for 8-bit synchronous operation.

The analog front end is relatively simple: the FPGA provides a clock (10MHz) and a start signal to the imagers, when start is high they collect the light and when low they clock out the analog pixel data. The ADCs convert all 4 channels simultaneously on the clock, serialize each channel, and send the data out over 4 LVDS DDR channels along with a data clock and a frame clock (80, 40, and 10 MHz, respectively). There are a few additional control lines that are mentioned later.

The imager has a two operating modes: frame capture and line capture. In frame capture mode the imager exposes for an entire projected frame (1 projected frame = 1 8x512 pixel 8bit 4kB frame sent to BBxM). In line-capture mode the imager captures each projected horizontal line individually and compsites the resulting 480 (projected lines) 8x512 images into a single 600x4096 pixel 8bit ~2.4MB frame. This is accomplished by selectively capturing horizontal lines over the course of 5 sucessive projected frames. At sequence start the imager exposes for a single horizontal line, captures the data (which takes roughly 2 lines time), then repeats, capturing lines 1,4,7,… of the first projected frame. The imager then dithers in time by one horizontal line and capture lines 2,5,8… on the second frame, etc. Ultimately this is done for five successive frames, dithering only on the first four for calibration reasons. For the logic behind these operating modes see the section on the Fiber Locator algorithm.TODO

# FPGA

This is the low-level heart of the fiber imager.

## IO

Inputs (from BBxM):

250MHz single ended clock - used to create the system clock

8 lines from a i2c GPIO expander – used for imager configuration. Toggle rate typically <~1Hz

cam\_fld – used to start image capture

Inputs from each AFE:

4 LVDS 80MHz DDR lanes, one lane per imager

LVDS 40MHz data clock, latch data on each edge

LVDS 10MHz frame clock, latch byte on positive edge

PLL lock signal – we can mostly ignore this

End of Line signal - ANDed EOL from each the 4 imagers. Toggles on readout of last pixel. We ignore this.

Outputs to each AFE:

LVDS Hamamatsu start signal – The imagers expose when high and readout when low.

LVDS 10MHz system clock. Serves as readout clock for imagers and sample clock for ADC

Data test pattern enable – Tells ADC to send the byte 0xC0.

Output to BBxM

8 lines via an i2c GPIO expander – used for status and error reporting

cam\_pclk: ~80MHz pixel clock. BBxM latches pixel data on configurable edge.

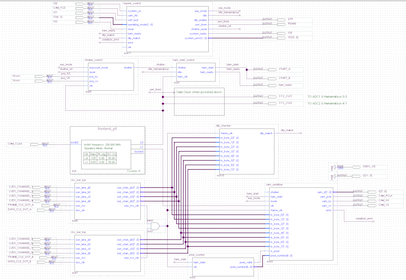
cam\_d[11:0] – 8 bit pixel data input to BBxM. Latched by cam\_pclk.

cam\_hs – horizontal sync pulse for image frame. May be able to omit if BBxM is setup to assume a frame size.

cam\_vs – vertical sync pulse for image frame. May be able to omit if BBxM is setup to assume a frame size.

## Arch

This section describes each of the low-level modules on the FPGA.



### master\_control

#### Inputs

##### system\_on

When asserted high the system shall power on.

##### cam\_fld

“Go” signal from the BBxM. Polarity can be whatever is convenient. When enters the active state the imager latches the configuration inputs and begins operating in the configured mode. Pulse the take a single image.

##### self\_test

execute a self test

##### operating\_mode[1:0]

mode 0 = frame mode. mode 1 = line mode. Modes 2 &3 reserved.

##### reset

reset the system to power-on configuration

##### ham\_ready

input from the ham\_start\_control module. Used to let the BBxM know when it can start an set cam\_fld and have the imager respond.

##### dtp\_match

output from the dtp\_checker module. Used to compute self-test results

##### error

error flag from module X. used to compute error state

##### clk

logic clock, just use the 250MHz clock from the BBxM?

#### Output

##### exposure\_mode

indicates operation in frame-mode or line-mode

##### idle

control line to the ham\_start\_control module. Tells the module that no exposure is in progress.

##### exposure\_mode

indicates operation in frame-mode or line-mode

### shutter\_control

#### Inputs

##### exposure\_mode

Indicates operation in frame-mode or line-mode

##### proj\_hs

HS signal from projector

##### proj\_vs

VS signal from projector

##### Reset

reset, it isn’t implemented TODO

##### clk

input clock > 2x proj\_hs, which is about 18kHz

#### Output

##### shutter

low when the electronic shutter should be closed and high when it should be open.

### ham\_start\_control

This module is used to insure that the constraints on the high and low period of the ham\_start signal are met. Under normal operating modes the module will essentially pass shutter straight though to ham\_clk.

When not idle it will disallows transitions that occur too rapidly, but does nothing to ensure they don’t occur too slowly. This shouldn’t be an issue as the shutter should be within spec for both operating modes. When idle, the module ignores shutter and autotoggles ham\_start, keeping it low for the maximum possible number of ham\_clks and high for the minimum possible.

It outputs ham\_ready as true whenever an exposure can be safely started/stopped (that is whenever a change of shutter will be passed through to ham\_start) as an additional bit of guidance.

The module functions by keeping track of the number of ham\_clk positive edges since ham\_start last changed.

#### Inputs

##### shutter

requested electronic shutter state

##### idle\_hamamatsus

When high the module will ensure the Hamamatsu start line stays within the datasheet spec.

##### ham\_clk

The clock sent to the Hamamatsus

#### Outputs

##### ham\_ready

high if an exposure can be started or stopped

##### ham\_start

the start signal to the Hamamatsu detectors

### my\_iser\_top

### cam\_serializer

### dtp\_checker

### pixel\_counter

### pll