**Fall**

16

**University of Michigan – Astronomy Department**

M2FS Control Systems

Jeb Bailey

The unabridged version of the authoritative system design and implementation document for the Michigan Magellan Fiber Spectrograph

08

**Fall**

Table of Contents

System Overview 3

Primary Enclosure Systems 3

Cylindrical Mount Systems 4

Secondary Cage Systems 4

Control concepts 4

Software 4

Tests 7

Architecture 5

System Requirements **Error! Bookmark not defined.**

Installation 7

BitBaking (heaven forbid) 7

Outstanding issues 7

FLS Imager 7

FPGA 8

IO 8

Arch 9

master\_control 9

shutter\_control 10

ham\_start\_control 11

my\_iser\_top 12

cam\_serializer 12

dtp\_checker 12

pixel\_counter 12

pll 12

# 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 or near each of the 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.

## Primary Enclosure Systems

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

SpectrographMotion 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.

SlitMotion 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 free-fall 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.

ShutterM2FS 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.

## Cylindrical Mount Systems

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

GuiderA Pololu hobby RC servo controller connected to two hobby servos, one functioning as a filter wheel and the other adjusting the camera focus.

Shack-HartmanAn Arduino & LED for the calibration light source and a Pololu simple motor controller for lenslet motion.

Plugging FeedbackM2FS uses a pair of computer speakers and a character display to provide auditory and visual feedback during the fiber plugging process. When used together the fiber illuminator & fiber imager provide real-time determination of any miss-plugged 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 miss-plugged in the FITS header.

## Secondary Cage Systems

These last systems are in the secondary cage area.

Fiber IlluminatorConsists 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 UnitThe 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.

# Software

This section describes all of the various software aspects of the control systems. It covers the basic operating environment, software architecture, software implementation, installation, troubleshooting commands, and finally, instructions for rebuilding the entire system from scratch.

## Operating Environment

The BBxM runs version of the Angstrom Linux distribution, based on the 3.2.18 mainline kernel. The image is built to include a variety componenents and a few kernel patches using the Angstrom/OpenEmbedded toolchain with a M2FS overlay. The final sub-section will describe some of the steps involved in rebuilding the installation image, though there should never be any need for this.

Software Packages:

Python, ipython, network ups tools, systemd, connman, git, M2FS-Control

/usr/local/ups/bin/upsrw

/usr/local/ups/bin/upscmd myups beeper.disable

## Architecture



Figure Overview of control architecture

The control software is structured as in Figure 1. The general idea is that each system is controlled by its own program. These programs, or Agents, are all instances of a control program class called Agent (agent.py). An Agent listens for a socket connection and accepts ACSII commands over that connection, which is an instance of a SelectedSocket. The default connection limit is one. The agent may also connect to other agents via SelectedSocket connections. Each hardware device is controlled by a single agent, typically over a SelectedSerial connection. There are a few that make use of command specific C programs to execute certain tasks. Some agents control more than one hardware device. Both SelectedSocket and SelectedSerial are implementations of the SelectedConnection interface.

If agent A receives a command and as a result sends a command to agent B, agent A will only expire the command if the original source disconnects. If agent B disconnects or doesn’t respond there is no mechanism to handle this asynchronously. While a per command watchdog timer seems to be the appropriate way to handle this, a proper implementation has not been done.

The general framework should support Agent A receiving commands from two different sources and the commands should be handled properly: both executing if they do not conflict or, if they do, the latter, resulting in a busy response. The implementation is very spotty. Conflicting commands must be handled piecemeal and if the first command makes use of blocking communication the second command may appear unresponsive for some time. This is a major limitation of the current Agent/Command/SelectedConnection mechanism and merits improvement.

## Implementation

### Agents

These programs are all subclasses of agent.

Director The director is the top level agent. It handles communication with the GUI, performs basic command verification (additional verification is left up to the individual agents) and passes commands along to the appropriate agents. For the occasional command that involves multiple agents it sends agent specific commands on to the individual agents.

Galil Agents (R &B) Theses agents, one for each side, handle motion and status commands for the filter exchanger, focus, hires elevation and azimuth, lores elevation, disperser slide, and fiber imager pickoff mirror. The agent maintains a record of which commands are currently executing on the Galil and disallows new commands that are blocked by an executing command. The agent also ensures that hardware limitations on the number of maximum simultaneous motions are met by rejecting commands when no threads are available.

Shack-Hartman Agent The agent sets the brightness of the Shack-Hartman calibration LED unit via /dev/shLED and controls the lenslet position via /dev/shLenslet..

This agent supports two incoming connections so that the Datalogger Agent can connect and request the temperature value from the lenslet controller.

Slit Controller This agent connects to both the shoe agents and runs the non-agent program slit\_position\_monitor TODO for mo

Shoe Agents (R & B) These agents, one for each side, handle motion and status commands for the tetrii in the fiber shoe.

These agents support two incoming connections so that the Datalogger Agent can connect and request the temperature value from the lenslet controller.

Plug Controller The

Datalogger Agent The

### Others

Galil Programmer This is a command line utility to flash the Galil motion controllers with the low level control code. The typical command line when testing the R side is:

galilProgrammer.py –f ../lib/m2fs.dmc –d /dev/galilR –auto

The command to burn the code onto the galils for production is:

galilProgrammer.py –f ../lib/m2fs.dmc –d /dev/galilR --burn

program\_galils.sh This is a convenience shell script to program both Galils. It stops and starts the galil agents before and after programming.

### Galil Software

#AUTO

#SHTDWN

#CALLRT

#INIT

#M2FSCFG

#MOMONI

#MCTIME

#ININT

## Installation

TODO

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

### Troubleshooting

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

## 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

## 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.

# 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.

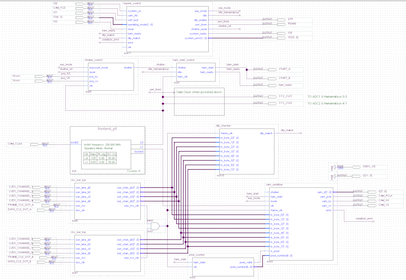
cam\_d[11:0] – 8 bit pixel data input to BBxM. Latched by cam\_pclk.

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 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

# Hardware Listing

* 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 Tripp-Lite UPS
* 1 Fiber Illuminator
  + 1 Microvision PicoP PDK (BBxM + projector module)
  + 2 100BASE-FX Jetcon Ethernet tranceiver
  + 1 HSVS Transmitter custom TTL-to-fiber converter
* 1 Fiber Imager
  + 8 Hamamatsu 18kHz line rate linear CMOS arrays
  + 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