

ARAFE Master Documentation

Brian Clark ^{*1}, Patrick Allison ^{†1}, and Thomas Mueres ^{‡2}

¹Department of Physics & Center for Cosmology and Astroparticle Physics (CCAPP), The Ohio State University

²Wisconsin IceCube Particle Astrophysics Center (WIPAC), University of Wisconsin-Madison

April 26, 2017

Abstract

Documentation of the firmware and software design for the ARAFE master.

Contents

1	Firmware Design	2
1.1	Firmware Repository	2
1.2	Modifications to the Energia defaults	2
1.3	Uploading Firmware	3
1.3.1	Uploading the Bootloader	3
1.3.2	Preparing the Firmware for the Bootloader	3
1.4	Information Memory	3
1.5	Monitoring Options	3
1.6	Fault Pin	4
1.6.1	Fault Curve	4
1.7	I ² C Device Registers	4
1.8	I ² C Interface	6
1.8.1	Sending a Command to a Slave	6
1.8.2	Reading a Monitoring Value	7
1.9	Custom Serial Interface	7
2	Software Design	7
2.1	Software Repository	8
2.2	Dependencies	8
2.3	Boot Strap Loader: “ArafeBSL”	8
2.4	Control Software: “ArafeD”	9
2.5	Python Serial Commander	10
3	Other Documentation	11

^{*}clark.2668@osu.edu

[†]allison.122@osu.edu

[‡]meures@icecube.wisc.edu

1 Firmware Design

The firmware for the ARAFE Master is designed in Energia, which makes the compile and upload simple. Developers should use Energia 17, **not** Energia 18. The firmware allows the ARAFE Master to serve as a serial master to the ARAFE PC slaves, and as an I²C slave to the ATRI board/ SBC.

1.1 Firmware Repository

The firmware is stored on the ARA DAQ Github [here](#). The `arafe_master.ino` file is the firmware source code and the `arafe_master.cpp.out` file is the compiled hex file that is loaded on the uC. A boot strap loader (BSL) for the uC is included in `mspboot.zip`. The `process_hex.py` is a python notebook for converting the output Energia hex file into a form that the BSL can work with.

The `identify-serial-ports.sh` script is a script that can be run on a Linux device to locate all plugged in serial devices, and is run by executing `sudo bash identify-serial-ports.sh`. This is useful if you are trying to connect to the ARAFE master by linux serial (i.e, through screen or pySerial) and need to know what device the ARAFE master is (i.e., `/dev/ttyUSB0`, etc). The `python_serial_commander.py` is a Python notebook with pySerial and a command line interface that makes interacting with the ARAFE master serial interface easy. More details are in section 1.9.

1.2 Modifications to the Energia defaults

For the firmware to work, a few modifications were necessary to the Energia defaults. They are the following:

1. Overwriting of the crystal enable, which is present on the MSP430FR5739 development board but not on the ARAFE Master. The enabling of the crystal causes an unnecessary two-second delay. Removing this is simple. We place an empty `void enableXtal() {}` function in `arafe_master.ino` which overrides the weakly defined default in Energia.
2. We utilize some pins on the micro-controller that are unused on the development board. This requires a modification of the `Energia.h` and `pins_energia.h` files in the `hardware/msp430/variants/fraunchpad/` directories of Energia. The changes add pins PJ.4, PJ.5, P2.3, and P2.4.
3. A mistake was found with the energy I²C `twi.c` library for slave eUSCI devices. The enable bits of the UCBZI2CCOA0 never got set, because the register was defined as a byte instead of a word. We recommended this change to Energia ([here](#)).

To fix issues 2 and 3 simultaneously, we forked the `hardware/msp430` directory and inserted the above fixes. Our custom version of the `hardware/msp430` library is on github [here](#). To compile the ARAFE Master firmware, one should delete the `hardware/msp430` directory in Energia 17 and replace it with the github contents. That is, run `git clone https://github.com/ara-daq-hw/energia-hardware-msp430.git msp430`.

1.3 Uploading Firmware

1.3.1 Uploading the Bootloader

First, the BSL `mspboot.zip` should be installed on the uC; it is located in `MSP-Boot/Simple/MSPBoot.txt`. This can be done with the TI FET Programmer, but this is expensive and usually unnecessary. Cheaper and just as easily, one can use a TI development board. All you have to do is jumper the 3.3V, TEST, RST, and GND pins from the development board over to the ARAFE master board. Then the development board can be connected via USB to your computer, and programmed using the FET-Pro430 Lite. Detailed instructions for doing this can be found [here](#) and [here](#).

1.3.2 Preparing the Firmware for the Bootloader

The compiled firmware for the uC should be installed using the BSL software described in section 2.3. To compile and prepare the firmware for the BSL, do the following. First, update your Energia installation with the changes described in section 1.2. Then, load the `arafe_master.ino` into Energia. Compile the firmware, and find the compiled hex file: Sketch → Show Compilation Folder. Second, this compiled hex file requires further processing. There is a line the ‘hexfile’ module for python may not recognize (‘record 3’). This line should likely start with “:04000003”. Delete this line. Next, process the hex file using the `process_hex.py` script. This relocates the vector table from 0xFF80 to 0xFB80 and also fills out all unspecified memory spaces with 0xFF. It also throws an error if the size of the main sketch exceeds the space available. Use it like “`./process_hex.py hexfile.hex outfile`”. Now, the BSL software can be used to upload the firmware.

1.4 Information Memory

We utilize the information memory of the micro-controller to hold important information about the board. This can be seen in the firmware file at the declaration of `info_t`. We store the board revision, power on defaults for all of the slaves, and a signature, which marks if the power on settings for all the slaves has been set before or not. The `info_t -> power_default` stores which slaves will come on at power up. This can be changed after deployment by altering the defaults through their I²C registers, as described in section 1.7.

1.5 Monitoring Options

There are several built in monitoring features on the board, and they are accessed as analog ports in Energia. This means they can be read with the Energia `analogRead` function. Monitoring is possible for the following:

- Item 0: 15V_MON – the value of the 15V rail that feeds the board (port 14)
- Item 1: CUR0 – the current to slave 0 (port 17)
- Item 2: CUR1 – the current to slave 1 (port 15)

- Item 3: CUR2 – the current to slave 2 (port 13)
- Item 4: CUR3 – the current to slave 3 (port 33)
- Item 5: !FAULT – the value on the fault pin, explained in section 1.6 (port 34)
- Item 6: 3.3VCC – the value of the 3.3V rail that feeds the microcontroller (port 139)
- Item 7: device temperature – the temperature of the microcontroller (port 138)

1.6 Fault Pin

From a design perspective, it is useful to know when an individual ARAFE slave is turned off – either because we are holding it in fault, or because the eFuse has blown for some reason. However, because the microcontroller has a limited number of pins, it was impossible to put a fault pin on every channel’s eFuse. Instead, the fault pin multiplexes all of the individual fault outputs from the eFuse switches onto a resistive ladder, forming a very simple ADC (read about the theory [here](#)). When each fault pin is enabled, and therefore pulling down on the resistive ladder, the voltage on the fault pin changes. The value of the fault pin can be mapped uniquely to a configuration of which slaves are on or off.

1.6.1 Fault Curve

As an example, the fault curves for M2000 (the ARAFE Master for A4) is given at room temperature in table 1 and plotted in figure 1. The “power setting” is the decimal representation of the which slaves are on/off (not in/ in fault) expressed in binary. What is meant by this is the following. If a binary number were [slave3status, slave2status, slave1status, slave0status], then 0000 would mean all slaves off, and would correspond to decimal 0. 1010 would mean slave 3 and 1 on, slave 2 and 0 off, and would be decimal 10, etc.

1.7 I²C Device Registers

The I²C device registers are as follows. They can be written to or read from to control the master and slave boards. Procedures for doing some of this is described in section 2.

- Register 0: Power Control Register (PWRCTL)
 - Bit [3:0]: Bits indicates which slaves are currently powered on. Bit [0] is slave 0, etc.
 - Bit [7]: This bit set high to actually update the power based on bit [3:0]. Cleared when the update is complete.
- Register 1: Power Defaults Register (PWRDFLT)
 - Bit [3:0]: Bits indicates which slaves are powered on at start-up. Bit [0] is slave 0, etc.

Decimal Value	ADC Counts at $\sim 20^\circ\text{C}$	Slave 3	Slave 2	Slave 1	Slave 0
0	359	0	0	0	0
1	375	0	0	0	1
2	393	0	0	1	0
3	412	0	0	1	1
4	430	0	1	0	0
5	454	0	1	0	1
6	479	0	1	1	0
7	507	0	1	1	1
8	543	1	0	0	0
9	580	1	0	0	1
10	622	1	0	1	0
11	671	1	0	1	1
12	722	1	1	0	0
13	788	1	1	0	1
14	867	1	1	1	0
15	965	1	1	1	1

Table 1: The fault table for M2000. Plotted graphically in figure 1.

- Bit [7]: This bit set high to actually update the power defaults based on bit [3:0]. It updates the non-volatile copy of the register stored in information memory. Cleared when the update is complete.
- Register 2: Monitoring Control Register (MONCTL)
 - Bit [2:0]: The monitoring value to convert. The monitoring value can be chosen from the table in section 1.5.
 - Bit [5:4]: The low bits of the conversion (two lowest bits of the 10 bit ADC)
 - Bit [7]: The bit set high to actually convert the monitoring value. This is cleared when register 2 and 3 are actually updated.
- Register 3: Monitoring Register (MONITOR)
 - Bit [7:0]: The high bits of the conversion (eight highest bits of the 10 bit ADC)
- Register 4: Slave Control Register (SLAVECTL)
 - Bit [1:0]: Destination for the slave command, i.e., which slave to write to. Bit [0] is slave 0, etc.
 - Bit [6]: Bit set high if the command times out.
 - Bit [7]: The bit set high to actually transmit the command and argument registers. This bit is cleared when the response is received or the timeout is received.

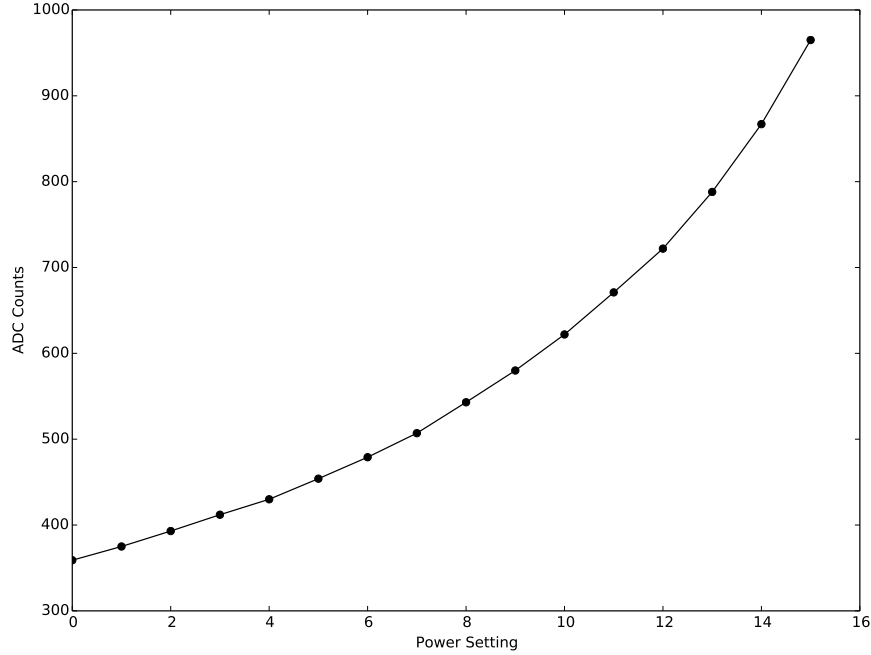


Figure 1: Graphical representation of the fault curve in table 1.

- Register 5: Command Register (COMMAND)
 - Bit [7:0]: The 8-bit command to be transmitted to the slave.
- Register 6: Argument Register (ARG)
 - Bit [7:0]: The 8-bit argument to be passed to the slave.
- Register 7: Acknowledgement Register (ACK)
 - Bit [7:0]: The 8-bit acknowledgement from the slave.

1.8 I²C Interface

The ARAFE Master uses the Energia `Wire.H` libraries for I²C communication. The communications protocol is relatively standard. Any I²C master should transmit to the ARAFE Master slave in a series of two bytes. The first byte should always be the register of interest as described in section 1.7 and the second byte should be the content for that register.

The I²C address for all of the ARAFE Masters is decimal 30, which is 0x1E in 8-bit hex, and 0xF in 7-bit hex. It is important to use the 7-bit address in all communications, because Energia stores the I²C address as a 7-bit number.

1.8.1 Sending a Command to a Slave

Sending a command to a slave is a multi step procedure. You must load both the COMMAND and ARG registers first, and then finally transmit the SLAVECTL register

content with the high bit set. The COMMAND and ARG options that can be sent to the slave are listed in the ARAFE slave protocol here.

For example, to set the attenuator on slave 1 signal channel 3 to 127 would require you first to write to the COMMAND register 0x03 for signal channel 3 (according to the slave protocol). Next you would write 0x7F (127 in decimal) to the ARG register. Finally you would write 0x81 to the SLAVECTL register. 0x81 = 10000001, where bit 7 being set high triggers the transmission to the slave, and bits [1:0] = 001 selects slave 0. You could interchange the order in which you sent the COMMAND and ARG values, but the SLAVECTL must come last.

1.8.2 Reading a Monitoring Value

Reading a monitoring value is a multi-step procedure. You must write to the MONCTL register with the desired parameter you want to monitor from section 1.5, and also set the high bit to signal you want the conversion. After checking for the MONCTL register to update –by looking for the high bit to go low–you can read out bits 4 and 5 of the MONCTL register, which form the lowest two bits of the 10 bit ADC value. To get the eight high bits, you must read the MONITOR register. Note that the result is raw ADC counts, and that must be converted to get physical voltages and currents.

For example, to read the 15V rail value, you would first transmit 0x80 to MONTCL, where 0x80 = 10000000, where the high bit is set, and the low three bits are all set low to mark that we want monitoring value zero, which is the 15V rail. After waiting for bit seven to clear, you can retrieve the MONITOR register, upshift it's values by two bits, and add to it bits 4 and 5 of the MONCTL register.

1.9 Custom Serial Interface

The serial interface for the board is relatively simple, and custom built by T. Mueres. Preservation of a serial interface is important for debugging, and replaces the `arafe_slave_tester` when it is not available. The serial interface is very similar to the I² interface, except flanked by different start and stop characters. All serial transmissions must start with a “c” and end with a “!”. The first byte transmitted after the “c” is the I²C register to be written to (in hex) according to section 1.7, and the second byte transmitted should be the value to be written to that register (in hex).

For example, to turn all of the slaves on, you would transmit “c008F!”. The “c” starts the transmission. “00” is the register for power control. 8F is binary 10001111, where setting the seventh bit high signals the change, and setting the four low bits high turns the slaves on. “!” ends the transmission.

The serial interface is typical: 9600 baud rate, 8 bits, no parity, 1 stop bit. All of these settings can be chosen on a terminal interface of your choice. We recommend Tera Term.

2 Software Design

The software written for the ARAFE Master is designed to interface between the ARAFE Master and the ATRI expansion port.

2.1 Software Repository

The software is stored on the ARA DAQ Github here. The `arafe.h/c` files contain the core of the program, and handles reading and writing to the ARAFE master via the `i2cComLib` and `fx2ComLib`. The `arafed.h/c` contains the “main” function, and is a wrapper class for the functions in `arafe.h/c`. The `atriComponents.h/c` handles the interface between the ARAFE master and the ATRI expansion ports. The `arafei2c.h/c` is currently unused. The boot strap loader is stored in `arafebsl.h/c`, and is used to load firmware onto the uC.

The two compiled executables are the boot strap loader `arafebsl` (described in section 2.3) and the ARAFE Master control software `arafed` (described in section 2.4).

2.2 Dependencies

To issue commands with the ARAFE Master software, the software must have access to a programmed FX2 device and a programmed FPGA on the ATRI board. Data taking does not have to be initiated on the ARA station, but the `ARAAcq` daemon must be running to negotiate the I²C expansion port and socket/packet control.

The software explicitly relies on the `ARAUtil`, `AraRunControl`, `AtriControl`, and `AraFx2Com` libraries (this is viewable in the Makefile), so those must be installed on the station before the ARAFE Master software can be used.

2.3 Boot Strap Loader: “ArafeBSL”

Firmware should be uploaded to the micro-controller via the boot strap loader (BSL). The procedure has two steps. First, the byte 0x55 is written to the I²C address 0x80. The arrival of this byte signals an erase of the main program memory. Then, each byte of the compiled binary to be loaded (in this case, `arafe_master.cpp.out`) file is written to the same address (0x80) one byte at a time. A secondary description of this can be found on the github readme.

The erase of the old firmware with the BSL must be initiated within 10 seconds of a power cycle of the ARAFE master, when the device is primed and waiting for an interrupt/erase command. While the uC is in BSL mode, the on-board LED blinks at 1Hz. If no erase command arrives, the uC resumes the current firmware (if programmed). If it is not programmed, it will wait in BSL mode. After programming has begun, the LED flashes one per byte written, which effectively means that it is solidly on during the duration of programming. After programming is complete, the uC will reboot— this means the LED will blink for 10 seconds, and then shut off, signalling that normal operation has begun.

The programming procedure is implemented in the `arafebsl` executable. The executable can be run in two modes. Program mode and erase mode. The erase mode just sends the erase/interrupt byte and nothing else. The program mode will send the interrupt byte and then immediately program the firmware. The two functions are called as follows:

```
erase: ./arafebsl erase
program: ./arafebsl program /path/to/arafe_master.cpp.out
```


The `arafebsl` executable is compiled at the same time as the `arafed` executable (described in section 2.4) by running `make` in the `ArafeMasterSoftware` directory. The program also includes verbosity control. If the global variable `v` at the beginning of the `arafebsl.c` program is set to “0” the verbosity is turned off, and setting it to “1” turns verbose output on.

2.4 Control Software: “ArafeD”

ArafeD serves as a ARAFE Master control interface, which communicates to the ARAFE Master over I²C through the ATRI expansion ports. `arafed` has five internal functions, given below in alphabetical order.

- Default Power (“defaultpwr”)
 - This sets the power default for all of the slaves. This updates the non-volatile information memory, and decides which slaves will receive power at boot.
 - Execute by running “`./arafed defaultpwr x x x x`” where `x = 0` for slave off, or 1 for slave on. Slaves are listed in the order slave0, slave1, slave2, slave3.
 - Example: to turn all slaves on by default except slave 1, execute `./arafed defaultpwr 1 1 0 1`.
- Help (“help”)
 - This lists all of the available functions.
 - Execute by running “`./arafed defaultpwr x x x x`” where `x = 0` for slave off, or 1 for slave on. Slaves are listed in the order slave0, slave1, slave2, slave3.
 - Example: to turn all slaves on except slave 1, execute `./arafed defaultpwr 1 1 0 1`.
- Monitoring (“monitor”)
 - This returns the ADC counts of a given monitoring pin. The monitoring options are listed in section 1.5. It follows the procedure described in section 1.8.2.
 - Execute by running “`./arafed monitor x`” where `x` is the number of the monitoring value you want.
 - Example: to retrieve the ADC counts for the 15V input, you would execute `./arafed monitor 0`.
- Power Control (“power”)
 - This changes what slaves are currently turned on.
 - Execute by running “`./arafed power x x x x`” where `x = 0` for slave off, or 1 for slave on. Slaves are listed in the order slave0, slave1, slave2, slave3.

- Example: to turn all slaves on except slave 1, execute `./arafed defaultpwr 1 1 0 1`.
- Slave Control (“slave”)
 - This transmits a command and argument to a specific slave. It follows the procedure described in section 1.8.1.
 - Execute by running `./arafed slave slave# command arg` where `slave#` is the slave you want to send to ($0 \rightarrow 3$), `command` is a command value in hex, and `arg` is a argument in hex. Valid commands and arguments are described in the slave communication protocol document.
 - Example: to set signal attenuator 3 on slave 0 to 127, execute `./arafed slave 0 0x03 0x7F`.

The `arafed` executable is compiled by running `make` in the `ArafeMasterSoftware` directory. The program also includes verbosity control. If the global variable `v` at the beginning of the `arafed.c` program is set to “0” the verbosity is turned off, and setting it to “1” turns verbose output on.

2.5 Python Serial Commander

Suren Gourapoura wrote a serial interface using the `pySerial` and `Cmd` libraries. It is available as the “`python_serial_commander.py`” notebook in the `arafe_master` GitHub repository. It is initiated by running `sudo bash python_serial_commander.py`.

It supports a command line through python, and you can get a listing of all available functions by typing “`help`”. The serial commander has three main functions. Described below.

- Slave Power (“allslave_power”)
 - This sets the power on all the slaves.
 - Execute by running `defaultpwr x x x x` where `x = 0` for slave off, or 1 for slave on. Slaves are listed in the order `slave0`, `slave1`, `slave2`, `slave3`.
 - Example: to turn all slaves on except slave 1, execute `allslave_power 1 1 0 1`.
- Serial Initialization (“serial_init”)
 - This initializes a serial port.
 - Execute by running `serial_init port` where `port` is the serial device on the computer. For example, this could be `“/dev/ttyUSB0”` etc.
 - Example: to initialize a serial port on `/dev/ttyUSB2` execute `“serial_init /dev/ttyUSB2”`
- Set Attenuator (“set_atten”)

- This sets the attenuator on a given slave and channel.
- Execute by running “`set_atten slave# chan sig/trig setting`” where `slave#` is the slave number from 0 to 3, `chan` is the channel on that slave from 0 to 3, `sig/trig` is whether you want to set the signal or trigger attenuator (0 for sig, 1 for trig), and the setting is the attenuation setting from 0 to 127.
- Example: to set signal attenuator 3 on slave 0 to 127, execute “`set_atten 0,3,1,127`”

3 Other Documentation

Most other documentation for the ARAFE master, including useful information on the I²C interface design and the registers, can be found here on docDB 1485.

Revision History

- April 26 2017, Brian Clark: Typo fixes, addition of fault curves, and description of hex prep software.