Germanium Readout Module (GeRM)
Electronics Overview

4-5-2020

1.0 Introduction

The GeRM Electronics is comprised of 2 printed circuit boards as shown below in figure 1. The bottom board is the detector module and goes in-vacuum. This board has 12 custom ASIC chips, named MARS, that are wire-bonded to the Germanium strip detector. The upper board is the Readout and Processing Module, which hosts an FPGA with an ARM processor running Linux and provides interfaces such as Gigabit Ethernet and SFP modules for high throughput data. The two boards are interconnected via a pair of rigid flex printed circuit boards and a vacuum tight feedthrough board, which for simplicity is not shown in figure 1.

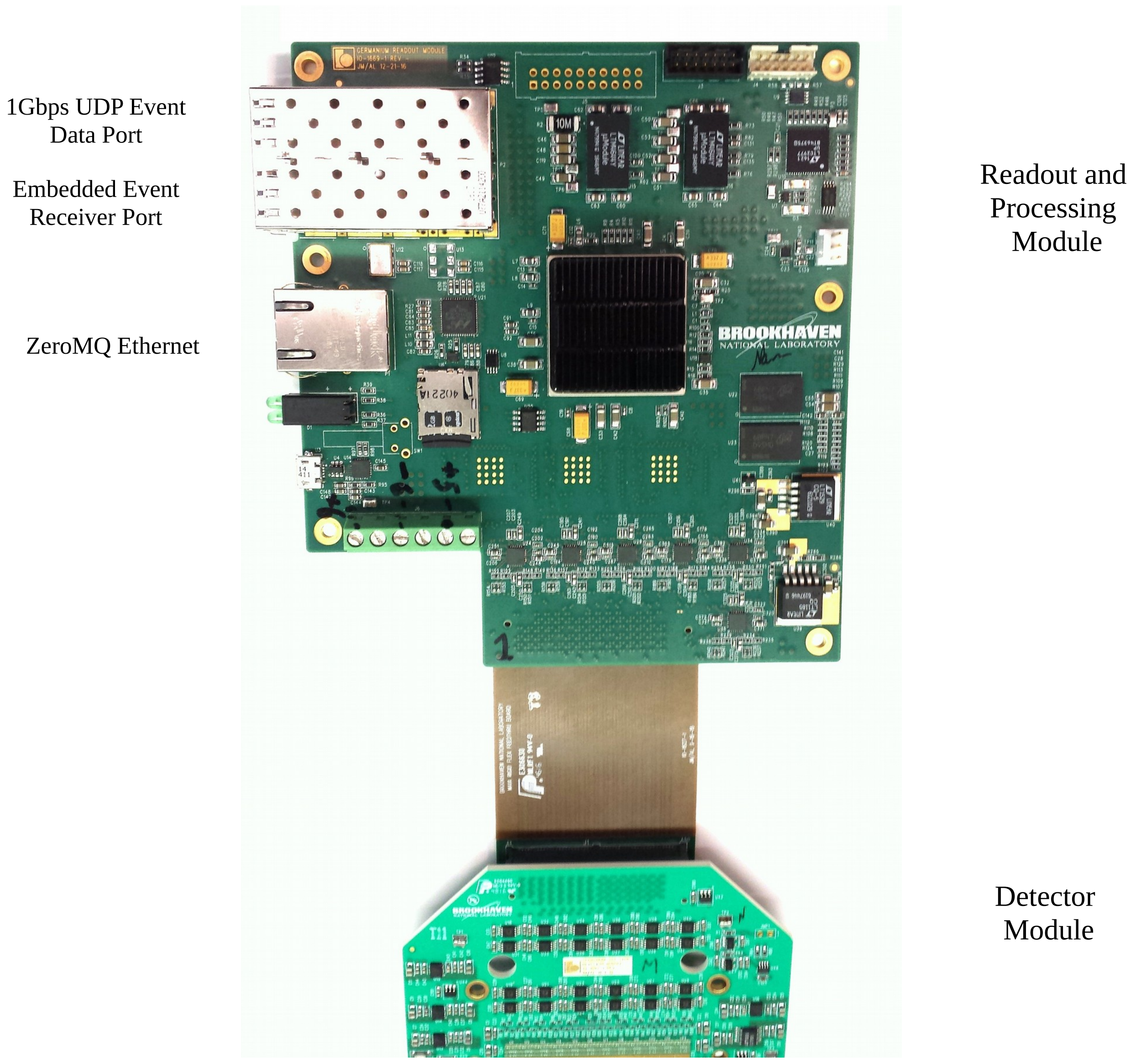

1Gbps UDP Event Data Port
Embedded Event Receiver Port
ZeroMQ Ethernet
Readout and Processing Module
Detector Module

Figure 1. GeRM Electronics

2.0 Detector Module

The detector module electronics board contains 12 custom ASICs developed by the Instrumentation Division, named the Multi-element Amplifier and Readout System (MARS). These chips are comprised of 32 front-end channels with low noise charge preamplifier, fifth order shaping amplifier, peak amplitude detector, threshold discriminator and time detector. The 12 ASICS are wire-bonded directly to the Germanium 384 Strip Detector and also the printed circuit board. When charge collected by the detector exceeds the programmable threshold the ASIC outputs a flag and a digital word containing the address. It also outputs an analog output for the Peak Detector output (PD), which contains the energy information and also a Time Detector output (TD), which contains the timing information. The PD and TD signals are then buffered and delivered to the Readout and Processing Module for digitization.

3.0 Readout and Processing Module

The main component on the Readout and Processing Module is a Xilinx Zynq FPGA. This FPGA contains a hard dual-core ARM A9 processor which runs Debian Linux. The non-volatile memory for the fpga bit file, linux kernel, and root-file system is contained on an SDCARD. The user can login to the sytem via ethernet connection using ssh. A micro-usb serial port connector is also provided for a console login. The other interfaces available are via the SFP (Small Form Pluggable) connectors. The top-most SFP slot is a high speed Gigabit Ethernet connection running UDP which provides the list mode event data from the detector. The bottom-most SFP slot is for an embedded event receiver, compatible with the NSLSII timing system.

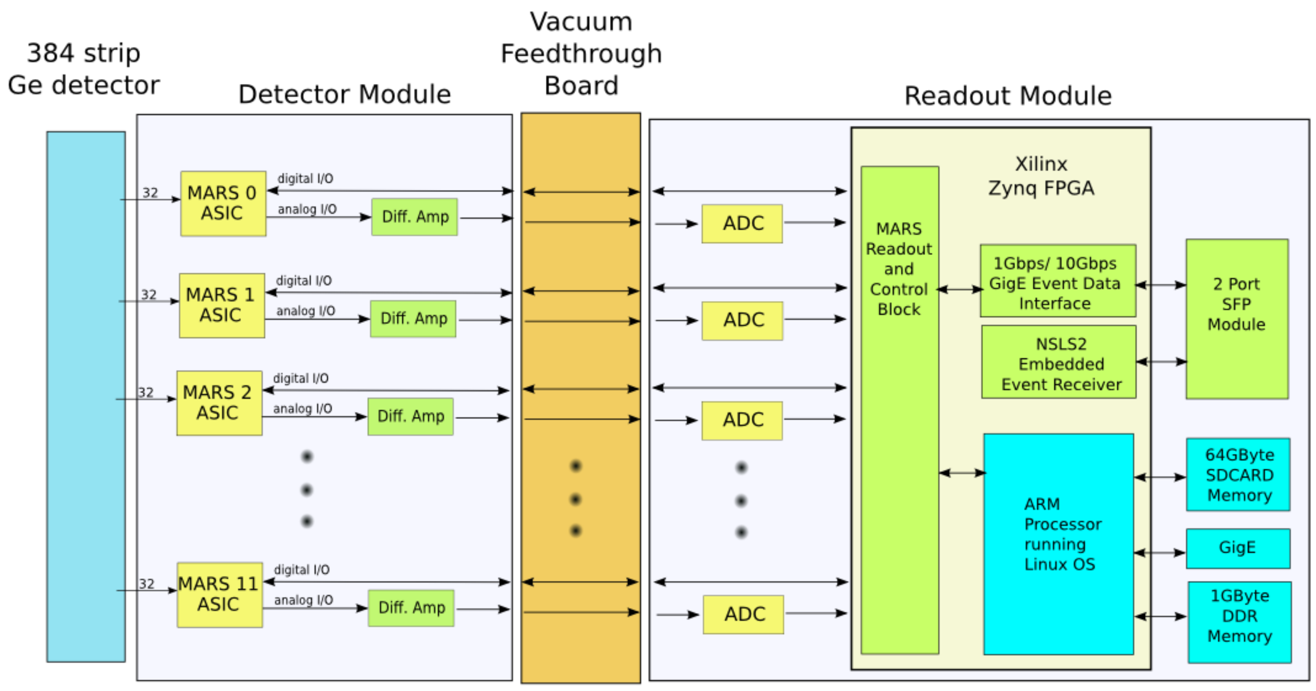


figure 2. GeRM Electronics Block Diagram

4.0 Zero-MQ Server

The primary method for communicating from the outside world with the detector is via ethernet using the Zero-mq protocol. A zserver program is provided which provides two primary functions. First it handles read/write requests to/from the FPGA registers to control the various aspects of the electronics and readout (Request/Reply). Second, it transmits out all event data from the detector (by reading registers on the FPGA) and sends them out using the Publish/Subscribe method.
The server can be started on the command line with :

```
 > ./germ_zserver1 [0,1]   where 0=no debug messages and 1=full debug messages
```

4.1 Register Reads/Writes

To read and write registers on the system the simple request/reply zero-mq method. The request packet should contain 3-32bit words (Word1=R/W(0,1), Word2=Address, Word3=Data). For information on Address and Data parameters, see section 3.2, which provides the complete address mapping. The reply message has the same format and for a read operation Word3 contains the data returned from the FPGA.

4.2 Event Data

Event Data is transmitted by the system using the publish/subscribe zero-mq method. For each event 2-32 bit words are transmitted which has the format shown in table 1, the topic for the message is set to “DATA”. The First 32 bit word contains the strip address of the photon event, along with the energy and fine resolution timestamp. The second 32 bit word contains the coarse 28 bit timestamp, which has a resolution of 40ns. When a frame starts, this timestamp is reset. It will roll-over after about 42 seconds. At the completion of each frame, a special packet is transmitted which contains a single 32 bit word which has the current frame number, the topic for this message is set to “META”. This allows the client to separate the incoming data stream based on frame number.

Word 1 (Photon Time/Energy)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Address									Time (TD)										Energy (PD)											

Word 2 (Time Stamp)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Time Stamp																											

Table 1. Event Data Format

5.0 FPGA Interface Registers

The FPGA registers are accessible to Linux via a provided kernel driver or using a simple mmap command from userspace. The physical address of the registers and their definitions are shown below.

Register Map
Base Address 0x43C00000

Address (words)	Address (hex)	Register Name	Function
0	00	MARS_CONFIG_LOAD	Used for configuring the MARS internal registers bit 1 : Config Load bit 2 : Config FIFO reset bits 27-16: Config ASIC (sends MARS serial data stream)
1	04	FP_LEDS	Controls the LED's on the front panel of the Readout Module bit 1 : Top Right LED bit 2 : Bottom left LED bit 3 : Bottom Right LED
2	08	MARS_CONFIG	Data Word Register to load MARS internal registers, written 32 bits at a time.
3	0C	FPGAVER	Readback of FPGA firmware version
4	10	MARS_CALPULSE	Each bit connects to a MARS asic tpck input
5	14	MARS_PIPE_DLY	Aligns MARS digital data with digitized PD and TD signals (Expert Use Only)
6	18	RSVD	
7	1C	RSVD	
8	20	MARS_RDOUT_ENB	Enables the Readout of MARS ASIC. Bit 11:0. One bit for each of the 12 MARS chips
9	24	EVENT_TIME_CNTR	Resets the event time counter. The event time counter is a 29 bit counter which is free running at 25MHz. The count value is latched and recorded with each event.
10	28	SIM_EVT_SEL	For testing the readout when real events are not available, this will generate simulated events at a rate set by the SIM_EVENT_THROTTLE register 0 = real events 1 = simulated events
11	2C	SIM_EVT_RATE	Sets the event rate when the SIM_EVENT_SEL register is set to 1
12	30	ADC_SPI	Connects to the PD & TD ADC SPI port

			bit 0 : sda bit1 : sck bt2 : csb
16	40	CALPULSE_CNT	Number of calpulses to generate. Changing CALPULSE_MODE register from a 0 to 1 will initiate the generating of calpulses.
17	44	CALPULSE_RATE	Number of 40ns clock ticks between calpulses. Note: Calpulse rate is the sum of RATE and WIDTH.
18	48	CALPULSE_WIDTH	Number of 40ns clock ticks of calpulse signal being high.
19	4C	CALPULSE_MODE	0 = bit bang mode, register 0x10 is used to generate calpulses. 1 = fpga state machine controls calpulses, via registers 0x40, 0x44, 0x48. Register 0x10 is used to select which asic's are selected.
20	50	TD_CAL	Used for calibrating the TD ramp time. Adds n clock ticks to asserting MARS enable pin.
24	60	EVENT_FIFO_DATA	Reads the Event Data
25	64	EVENT_FIFO_CNT	Number of events currently queued up to be read from the EVENT_FIFO_DATA register
26	68	EVENT_FIFO_CNTRL	Control Register for Event Data bit 0 : fifo enable bit 1 : testmode. Sends test count pattern for data bit 2 : fifo reset
32	80	DMA_CONTROL	Control Register for DMA operation bit 0 : enable bit 1 : testdata enable bit 2 : fifo reset bit 3 : test data counter clear bit 4 : DMA mode. 0=continuous, 1=single BURSTLEN
33	84	DMA_STATUS	Status Register for DMA operation bit 0 : dma active
34	88	DMA_BASEADDR	Base physical address for DMA transfers
35	8C	DMA_BURSTLEN	Number of 32 bit words to DMA
36	90	DMA_BUFLEN	DMA Buffer length (in lwords), DMA address will wrap back to the BASEADDR when it reaches the BUFLEN
37	94	DMA_CURADDR	Current Address to DMA write pointer
38	98	DMA_THROTTLE	When DMA is in testdata mode, sets the rate for DMA writes. 0=100Mwords/sec, 1=50Mwords/sec, ...

40	A0	UDP_IP_ADDRESS	Sets the IP address for the SFP UDP Interface.
48	C0	DMA_IRQ_THROTTLE	Controls the IRQ rate for the DMA operations. Interrupt after every (n+1) 4kbyte transfers
49	C4	DMA_IRQ_ENABLE	Enable Interrupts after DMA operation
50	C8	DMA_IRQ_COUNT	Counter of number of interrupts sent
52	D0	FRAME_SOFTTRIG	1 = Start a new event frame. Readback reports if current frame is acitve
53	D4	FRAME_TIME	Sets the frame length. Resolution is 1us. Max frame time is 2^31*1us ~ 35min. Interrupt is generated when the frame time expires. (IRQ # 61)
54	D8	FRAME_NUM	Sets the frame number on write. Read returns the current active frame number. Increments when frame completes
55	DC	FRAME_MODE	Sets the frame mode 0 = normal mode. Frame starts on FRAME_SOFTTRIG and completes after FRAME_TIME 1 = debug mode. Frame length is set to infinite.

6.0 High Speed Event Data Interface

In addition to the Zero-MQ interface, Detector Event Data is also transmitted via the bottom SFP slot via a UDP protocol. This interface provides a higher bandwidth interface which can accommodate event rates up to 12 Mevents/sec. The IP address for this interface is set through the zeroMQ interface, via register 0xA0. The MAC address of this interface is hardwired to 00:01:02:03:04:05. The interface does not currently support ARP requests so a manual entry to the host computer ARP table must be made. From the command line type “arp -s ipaddr 00:01:02:03:04:05”, where ipaddr is the address programmed into register 0xA0. This command will associate the hardwired MAC address of the interface with the programmed IP address. Before event data will be transmitted, the interface needs to be enabled by sending a simple 3 word UDP enable packet to the device with the format shown in Table 2. The device will send an acknowledge packet with the same information.

Lword Number	Function	Value
0	Key	0xdeadbeef
1	Address	0x1
2	Data	Bit 0: 0 = transmit events via UDP disabled. 1 = transmit events via UDP enabled.

		Bit 1: 0 = transmit real event data 1 = transmit test data (count pattern)

Table 2. UDP Enable Packet Format

After the host successfully sends this packet, event data will be transmitted to the host, using as its destination IP address and destination MAC address the values it received from the enable packet. The format of the event packet(s) are shown in Tables 3-5. Event data will begin streaming at the beginning of the next frame and will be packaged into UDP packets having a payload size of 1024 bytes. Each packet will have as its first 32 bit word a packet counter, which can be used to determine if any packets were lost in transmission. At the start of a new frame, a unique 32 bit word will be sent in words 2 and 3 to mark the beginning of a frame (Table 3). Packets will continue to be sent with event data, always having a packet counter attached at the beginning (Table 4). Finally when the end of a frame is reached, the last 2 words in the packet will have a unique format as shown in Table 5. This packet will have a variable length depending on the number of events in a frame.

32 bit Word Number	Function	Value
0	Packet Counter	Running counter
1	Start of Frame Marker	0xfeedface
2	Frame Number	Running counter
3...1023	Event Data	See Table 1.

Table 3. 1st UDP packet at Start of Frame

32 bit Word Number	Function	Value
0	Packet Counter	Running counter
1..1023	Event Data	See Table 1.

Table 4. 2nd – (n-1) UDP packet

32 bit Word Number	Function	Value
0	Packet Counter	Running counter
1..(n-2)	Event Data	See Table 1.
n-1	Events lost to overflow	count
n	End of Frame Marker Low	0xdecafbad

Table 5. 1st UDP packet at End of Frame

A C program is provided (germ_getudpevents.c) which sends the enable packet and then receives the event data. When an end of frame marker is received, the received data for that frame is saved to disk. To help develop and debug this interface a special test mode is provided which provides a simple counter pattern with an adjustable burst length and adjustable data rate. Test data transmission is initiated by writing a UDP packet to the Readout Packet with details provided soon. The file format of the data is shown below in Table 6.

32 bit Word Number	Function	Value
0	Start of frame marker	0xfeedface
1	Frame Number	# of frame in run
2..(n-2)	Event Data	See Table 1.
n-1	Events lost to overflow	# of overflows
n	End of Frame Marker Low	0xdecafbad

Table 6. Binary file format of raw event data

7.0 System Initialization and Command Line Interface

To get the system up and running a few initialization tasks need to be performed. The user can ssh into the system using the command

```
> ssh root@10.0.143.160.
```

The default password is root, which should be changed using standard linux administration commands. The default ip address is set to 10.0.143.160, but this can also be changed by modifying the file /etc/network/interfaces (DHCP is also supported) To easily change the ip address the user can also connect to the console of the system using a standard micro-usb cable. The console will enumerate as /ttyUSB0 or /ttyUSB1, depending on other devices connected to the host computer. Once connected to the system, change directory to the germ subdirectory. A simple bash script can be executed

```
> ./germ_init_all
```

which provides all the low level initialization registers that need to be set. Next the framing configuration needs to be setup using the bash script

```
>./frame_setup
```

This file configures the length of each frame, the starting frame number and the frame mode. For the normal frame mode (=0) th frame starts on FRAME_SOFTTRIG and completes after FRAME_TIME. For the debug mode (=1), the frame length is set to infinite.
Next, the ZeroMQ server needs to be started, so that communication with clients can be established.

```
> ./germ_zserver1 [0,1], where 0=no debug messages, 1=full debug messages.
```

The next step, would be to configure the MARS ASIC's which is described in section 7.1. After the ASIC configuration is complete, calibration pulses can be sent to the ASIC's to verify everything is working. For this run the script

```
> ./mars_calpulse_loop, which will send calibration pulses to the ASICs.
```

Finally, start a frame (the parameters of which are described in the ./frame_setup script previously run.

```
> ./frame_trig
```

The zeromq server will then start displaying messages, which describe the channel(s) that were pulsed as well as energy and timing information as shown below in figure 3.

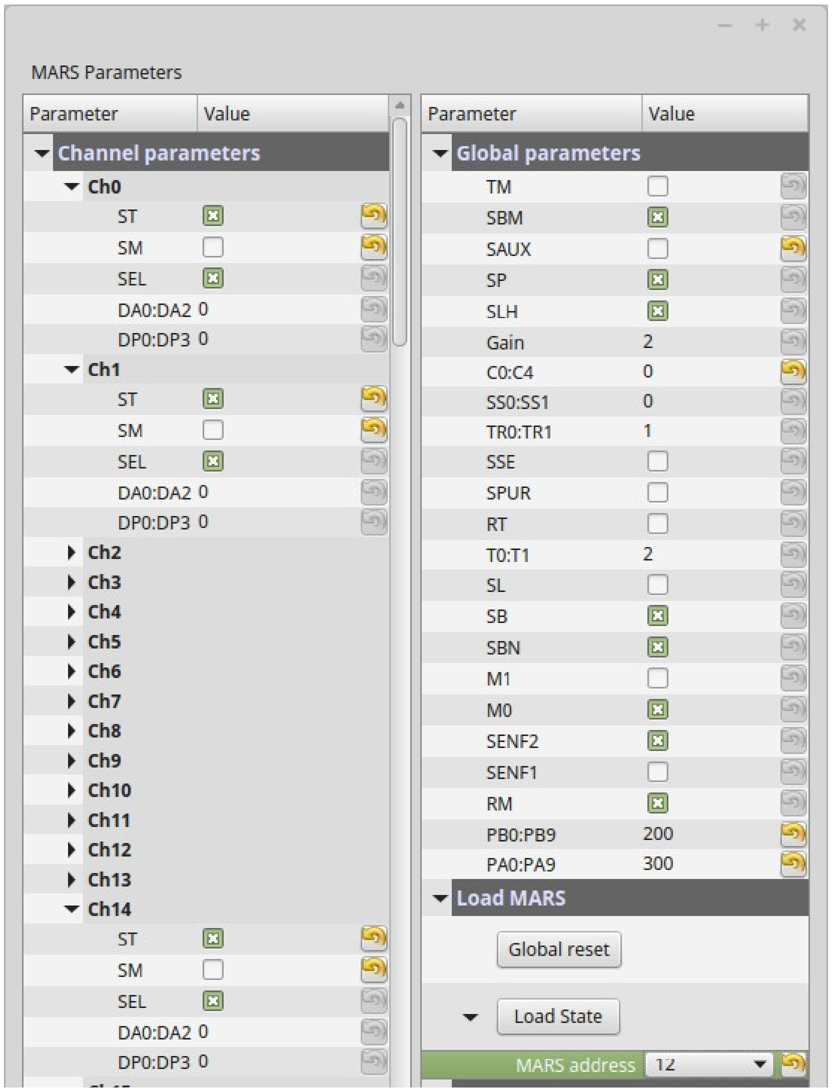
```
Frame Started...
ASIC:    2   Chan:    0   PD: 2013   TD:   37   Timestamp: 362843279
ASIC:    2   Chan:    1   PD: 2066   TD:  185   Timestamp: 362843292
ASIC:    2   Chan:    2   PD: 2109   TD:  382   Timestamp: 362843305
ASIC:    2   Chan:    0   PD: 2026   TD:   36   Timestamp: 115401561
ASIC:    2   Chan:    1   PD: 2068   TD:  183   Timestamp: 115401574
```

```
ASIC:    2   Chan:    2   PD: 2121   TD:  382   Timestamp: 115401587
ASIC:    2   Chan:    0   PD: 2016   TD:   37   Timestamp: 115679474
ASIC:    2   Chan:    1   PD: 2074   TD:  185   Timestamp: 115679487
ASIC:    2   Chan:    2   PD: 2104   TD:  382   Timestamp: 115679500
Frame Complete...
Total Events in Frame=9  Rate=450.0
```

figure 3. ZeroMQ Server Output

7.1 MARS Configuration

After the zero-mq server is running, the MARS ASICS need to be configured. Each ASIC has a 432 bit serial string, which initialize all its parameters. This can easily be done using the included python program, “mars_config.py”, which is run on the host computer and communicates with the system via the zero-mq read/write interface. The GUI is shown below in figure. Each of the 12 MARS ASIC's has channel parameters to enable/disable a channel, and global parameters such as Gain, Shaping Time, etc. Please refer to the MARS documentation for details of each of the bits.



MARS Parameters

Parameter	Value
Channel parameters	
Ch0	
ST	✓
SM	☐
SEL	✓
DA0:DA2	0
DP0:DP3	0
Ch1	
ST	✓
SM	☐
SEL	✓
DA0:DA2	0
DP0:DP3	0
Ch2	
Ch3	
Ch4	
Ch5	
Ch6	
Ch7	
Ch8	
Ch9	
Ch10	
Ch11	
Ch12	
Ch13	
Ch14	
ST	✓
SM	☐
SEL	✓
DA0:DA2	0
DP0:DP3	0

Parameter	Value
Global parameters	
TM	☐
SBM	✓
SAUX	☐
SP	✓
SLH	✓
Gain	2
C0:C4	0
SS0:SS1	0
TR0:TR1	1
SSE	☐
SPUR	☐
RT	☐
T0:T1	2
SL	☐
SB	✓
SBN	✓
M1	☐
M0	✓
SENF2	✓
SENF1	☐
RM	✓
PB0:PB9	200
PA0:PA9	300
Load MARS	
Global reset	
Load State	
MARS address	12

figure 3. mars_config.py GUI for configuring MARS ASICs

8.0 Powering the system

The system requires 3 voltages (+5v digital and +/-6v analog) which are supplied using the included power supply. Both the detector module and the readout module are powered using these supplies. Since the detector module is in vacuum it is important to not to power the system while it is warm as the electronics may overheat. It is also important to power the electronics before the cryostat becomes too cold, as it will prevent the electronics from operating below their minimum operating temperature for commercial grade components (-40 degrees C) The procedure for system power-up and power-down are outlined below. The detector module includes three temperature sensors (TMP100) for monitoring the on-board temperature of the detector module. From the system command line these temperatures may be monitoring with the following command.

```
> ./tmp100
```

It is also recommended to independently monitor the cooling plate temperature with an RTD or other device that has a larger temperature operating range.

Cool-down Procedure:

1. With electronics off, begin cooling down cryostat to about -25 degrees C
2. Power electronics.
3. Monitor the electronics temperature using the tmp100 program.
4. Continue to cool down cryostat to desired temperature

Warm-up Procedure:

1. Begin warm up with electronics on.
2. When cryostat temperature reaches about -40 degrees C power off the electronics
3. Continue warm-up.