Distributed Data Acquisition and Storage Architecture for the SuperNova Acceleration Probe

Alan Prosser, Guilherme Cardoso, John Chramowicz, John Marriner, Ryan Rivera, Marcos Turqueti

***Abstract*— The SuperNova Acceleration Probe (SNAP) instrument is being designed to collect image and spectroscopic data for the study of dark energy in the universe. In this paper, we describe a distributed architecture for the data acquisition system which interfaces to visible light and infrared imaging detectors. The architecture includes the use of NAND flash memory for the storage of exposures in a file system. Also described is an FPGA-based lossless data compression algorithm with a configurable pre-scaler based on a novel square root data compression method to improve compression performance. The required interactions of the distributed elements with an instrument control unit will be described as well.**

***Index Terms*—Data acquisition, data compression, distributed memories, space vehicle electronics.**

I. Introduction

THE SuperNova Acceleration Probe (SNAP) is a proposal for a satellite observatory being prepared for the Joint Dark Energy Mission (JDEM) [1]. The observatory design features a 2 meter telescope with a field of view of approximately 0.7 square degrees. A half-billion pixel imaging camera consisting of visible light CCD and near-infrared (NIR) detectors provides imaging capabilities from the visible to the near infrared portions of the spectrum. The CCD imager provides coverage for wavelengths from 0.35 μm to 1.0 μm. The NIR imager is a HgCdTe detector providing coverage for wavelengths from 0.9 μm to 1.7 μm. Also included are spectrometers to support the identification of Type IA supernovae. The mission includes a supernova survey and a weak-lensing survey.

The instrument is being designed as a distributed readout and memory system with multiple partitions called slices. Each slice consists of non-volatile memories to store multiple exposures in a distributed file system. A total of 36 slices will be dedicated to the CCD imaging array. Each CCD slice will support $(3510)^2$ pixels. An additional 36 slices will be dedicated to the NIR imaging array. Each NIR slice will support $(2048)^2$ pixels. Data are to be collected for exposures of 300 seconds each. During each exposure period, there will be no transfer of data to or from the front end electronics. After each exposure, a readout period of approximately 30 seconds will occur during which the data collected during the exposure are transferred to the slices for compression and storage. The compressed data for each exposure will be stored in a standard packet format in NAND flash memories on the slices. The activities of data readout, compression, and file storage are coordinated by a master flight computer called the Instrument Control Unit (ICU). Approximately once a day, the files representing the exposures are extracted from the flash memories and transmitted to a ground station.

II. Data Acquisition System Architecture

Figure 1 illustrates the key elements in the data acquisition system. The front end electronics modules include detector readout ICs that control the delivery of image data from the detector elements. The data delivered from these readout ICs are sent over 25 Mbps links to a slice unit. Independent slices are dedicated to independent detector elements. This independence is required to eliminate the possibility of a failure in one slice causing another slice to malfunction. CCD channels will deliver data from four pixels as a group framed by start, stop, and a single parity bit. The data word for each pixel is 16 bits long with the 2 most significant bits representing a scale factor. The pixels are interleaved meaning that, following the start bit, 16 bits representing the first pixel are transmitted, followed by 16 bits each for the second, third, and fourth pixels. This group of pixels is followed by a single parity bit computed using all of the 64 bits followed by a stop bit. When the channel is inactive, the channel is held high.

Manuscript received 11 May, 2007; revised 17 October, 2007. This work was supported by Fermi National Accelerator Laboratory operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United State Department of Energy.

Alan Prosser is with the Fermi National Accelerator Laboratory, PO Box 500 Batavia, IL 60510 (e-mail: aprosser@fnal.gov).

John Chramowicz is with the Fermi National Accelerator Laboratory, PO Box 500 Batavia, IL 60510 (e-mail: chramowicz@fnal.gov).

John Marriner is with the Fermi National Accelerator Laboratory, PO Box 500 Batavia, IL 60510 (e-mail: marriner@fnal.gov).

Ryan Rivera is with the Fermi National Accelerator Laboratory, PO Box 500 Batavia, IL 60510 (e-mail: rrivera@fnal.gov).

Marcos Turqueti is with the Fermi National Accelerator Laboratory, PO Box 500 Batavia, IL 60510 (e-mail: turqueti@fnal.gov).

Guilherme Cardoso was with the Fermi National Accelerator Laboratory, PO Box 500 Batavia, IL 60510. He is currently with Aguila Technologies, 310 Via Vera Cruz, Suite 107, San Marcos, CA 92069 (e-mail: bcardoso@aguilatech.com).

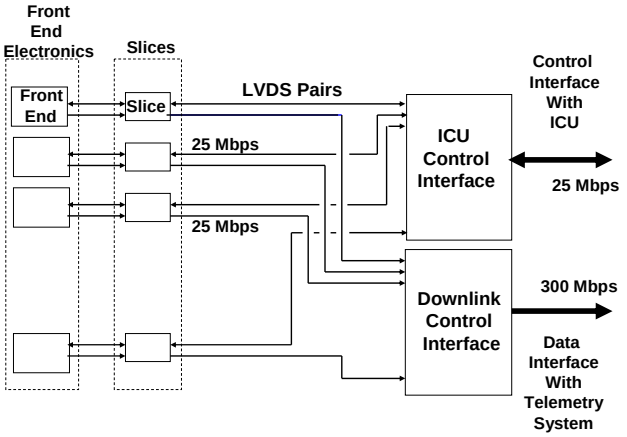

Fig. 1. DAQ System Architecture

NIR channels will deliver data on four separate connections operating synchronously. The processing of NIR detector pixels will be done in groups of four so that pixels are compressed in blocks in which neighboring pixels are compressed as a unit.

The slices and front end components are programmed and controlled by the ICU. Each ASIC on the front end modules and an FPGA on the slices are assigned a unique ASIC ID code used to address the device to be programmed. Broadcast write transmissions will be supported so that multiple targeted ASICs may be commanded with the same command cycle. Commands issued to a slice FPGA will be filtered from the command stream so that they do not arrive at the ASICs of the associated front end module.

The ICU issues commands and receives responses using a controller interface that implements an SPI-like serial protocol. The ICU interface controller is responsible for decoding the slice channel address and routing the command to the designated slice over a dedicated 25 Mbps link from the controller. Responses from the slice will have the slice channel address encoded in the bit stream returned to the ICU.

Figure 2 illustrates the major components of a slice.

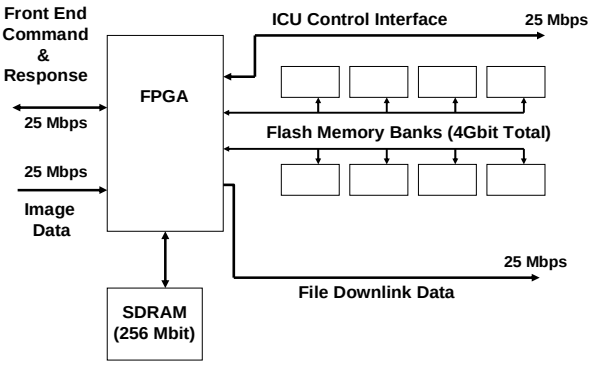

Fig. 2. Slice Hardware Architecture

At the heart of each slice is an FPGA which communicates with the ICU via the ICU control interface. All communications for the channel are processed by the slice FPGA. The FPGA forwards only those commands issued to front end module ASICs over a dedicated control interface to the front end module. There is also a dedicated science data interface over which the detector data are delivered to the slice during front end readout operations. During the downlink operations, the slice FPGA is commanded to extract stored image files and transmit them to a downlink controller which delivers the data for all slices to the spacecraft telemetry system.

The exposure files are stored in flash memory on the slice. The flash memories are organized in two independent banks (for a total of 4 Gbits) to prevent a failure on an I/O bus from eliminating an entire slice channel. The location of file data in the flash memory is coordinated with the ICU. The ICU can maintain a file system in which the flash memory blocks used to store the data for a given exposure is recorded.

Slice channels which are serving NIR detector readout also include a 256 Mbit Synchronous DRAM (SDRAM) to be used as an intermediate storage device. Unlike the CCD detectors, the NIR detectors can be read more than once per exposure so that signal averaging can be applied to reduce the effects of readout noise in the electronics. The SDRAM will be used to hold sums of pixel data (one sum for each pixel) as multiple readout cycles are processed for a single exposure. Then, after a sufficient number of readout cycles have been accumulated, the pixel data can be shifted to complete the averaging process. The resulting pixel averages reside in the SDRAM. Once the data have been processed in this manner, they can be delivered to the data compression block in the FPGA. CCD channels can be compressed on the fly as the data streams in from the detector front end modules. Exposure data for the NIR channels will be compressed during the following exposure period.

III. FPGA Firmware Architecture

The functional blocks of the FPGA firmware are illustrated in Figure 3. Commands from the ICU are decoded and routed within the FPGA to the various programmable blocks by the command processor. Responses to read and status requests issued by the ICU are formatted in the output formatter. Blocks are enabled for operation by setting a bit corresponding to the respective blocks in the block enable register.

The science data front end processor is responsible for serial to parallel conversion of incoming image data during an exposure readout period. In CCD slice channels, the parallel data words are passed directly to the data compression block. In NIR slice channels, the parallel data words are first processed by the accumulation and SDRAM controller blocks before they are passed to the data compression block.

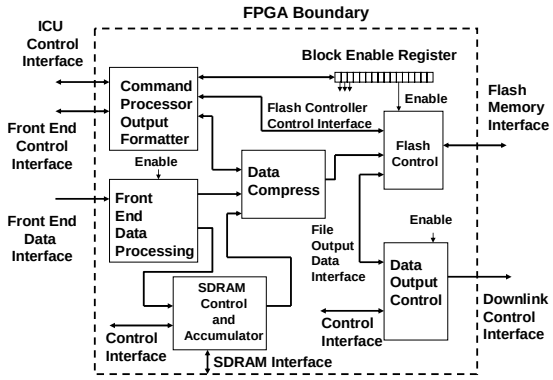

Fig. 3. Slice FPGA Firmware Architecture

The science data compression block consists of a strictly lossless data compression algorithm that is based on a subset of the CCSDS Lossless Data Compression recommendation [1]. This lossless block is preceded by an optional pre-scaling block that is not strictly lossless but truncates low order bits that contain little useful information because they are dominated by detector noise [2]. The lossless compression component is implemented as a parallel algorithm in the FPGA firmware. The algorithm concurrently evaluates the resulting sequence lengths for fourteen different compression options. The option selected is the one which results in the shortest compressed sequence length. An option code is included in the compressed file bit stream for each block of processed data. This option code identifies the selected option used to encode the block of data. If no option results in a sequence shorter than the length of the original, uncompressed data, the data words are not compressed and are labeled with an option code indicating “no compression”.

The optional pre-scaler, which is used to map data words into shorter code words, is implemented in firmware as a lookup table and binary search which iteratively refines the estimation of the code word to represent the data. Pixel data are compared against the contents of a lookup table which represent ranges over which the data may be represented by a compressed word of shorter length. The final compressed value is determined through a binary search of the contents in the lookup table. The number of steps is bounded by $\log_2(N)$ where N is the number of code words in the approach. Like the CCSDS algorithm, this pre-scaler can be applied in real time to the data arriving from CCD channels.

The compressed data will be passed out from the compression block as CCSDS source packets [3]. Each packet will include data from a fixed number of pixels to make packet decoding straightforward. These source packets are delivered to page buffers and written to the flash memory. A count of the number of bytes is recorded for each exposure file on each slice. After a readout cycle is completed, the number of compressed bytes is read from each slice by the ICU and recorded. The byte counts will be used during downlink operations so that only the required number of bytes will be transmitted to the ground.

IV. Data Compression And Storage

Figure 4 illustrates the data compression architecture implemented in the FPGA.

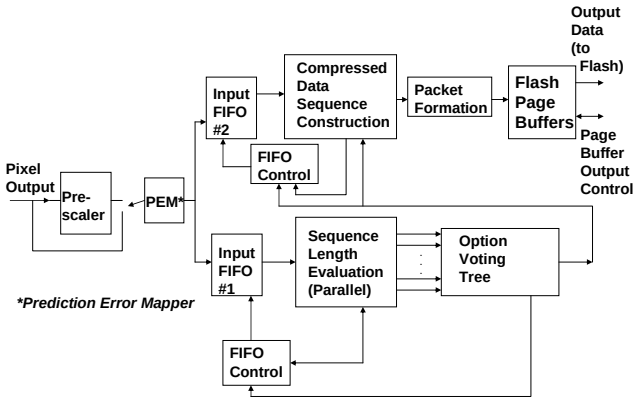

Fig. 4. Data Compression Architecture

Data will be delivered to this block either directly from the CCD readout chip or from the SDRAM of a NIR channel. The data may be passed through the optional pre-scaler, followed by the lossless data compression block. An option to force no compression is included in the lossless compression block. The data values are assembled into blocks corresponding to a fixed number of pixels and the best choice among the options is evaluated by comparing resulting sequence lengths in parallel as the block data accumulates. Once the sequence lengths have been determined, a binary voting tree is used to declare the best option. This option code is then fed forward so that the pixel data corresponding to the block just evaluated can be compressed into the output bit stream according to the rules of the lossless data compression algorithm. Because data for four channels from the CCD detector are interleaved, input FIFO #1 is present to accumulate four full compression blocks prior to the evaluation of data. This accumulation represents a fixed FIFO start up latency. Once the best option is declared by the voting block, the pixels in the input FIFO #2 are extracted and, using the chosen option, assembled one bit at a time into the compressed data sequences. These bits are passed out of the sequence construction block as bytes where they are input to the CCSDS Source Packet formation block. This block assembles the compressed data corresponding to a fixed number of detector pixels into source packets with a primary header and an optional secondary header. The secondary header will contain a packet ID, an exposure ID, and a state description of the conditions under which the data were taken. Included in the state description would be application specific parameters such as the selected compression technique (no compression, lossless compression with pre-scaler, lossless compression without pre-scaler). Packets will be assembled in two packet buffers which will be alternated by the firmware. When one buffer is being written with data, the other buffer can be passing the data for a

previously constructed packet forward to one of two receiving page buffers. A page buffer is designed to contain the same number of bytes as a page in the flash memory architecture. The transfer of data in page-sized units is the most efficient way of utilizing the flash memories.

Although the packets are constructed from data with a fixed number of pixels, the size of the packets will vary due to the application of compression. However, the packet specification was defined such that even the longest possible packet must have a length less than or equal to the size of a page buffer. Restricting packets to this size prevents a bottleneck in the transfer from packet buffers to page buffers.

V. CCD Channel Operation Sequence

The sequence of events for processing exposure data includes the following activities:

1. Exposure readout set up – this is the configuration of each slice to prepare it to accept image data and to compress and store the image data in allocated flash memory blocks. The process of setting up an exposure read out will take place during the time in which an exposure is actually taking place, while the shutter on the focal plane of the imager is open.
2. Exposure readout – this is the actual process of transferring the data from the detector to the slice FPGA. During this time, data words are delivered to the slice FPGA and either compressed in real time (for CCD slice channels) or averaged and stored in the SDRAM for compression after the readout period (for NIR slice channels).
3. File downlink set up – this is the process of configuring each slice to prepare for extracting file data from the flash memories.
4. File downlink – this is the process in which previously stored file data are extracted from the flash memory and forwarded to a downlink controller for transmission to a ground station.

To set up an exposure, the ICU will provide a list of available flash memory blocks (including a chip ID targeting the specific flash devices on the slice) and write these to a flash block ID RAM on the slice FPGA. The ICU also provides the count of the number of such blocks provisioned. After the flash memory blocks have been provisioned, the ICU will configure the programmable blocks in the FPGA including the data compression blocks (pre-scaler and lossless), and the flash controller. The flash controller will be programmed with a “data readout” command code. Once the controller and compression blocks have been configured, the ICU will issue a command to the front end to begin the delivery of data. The slice is required to accept the incoming data fast enough to prevent any bottlenecks. The compressed packet data will fill the flash memory page buffers. Each byte written to the page buffers is counted. Once a page buffer is filled, the flash controller will retrieve the ID of the first allocated flash memory block from the flash block ID RAM and use this to construct a command sequence to be issued to the flash memory for executing a page program operation (which commits a page consisting of 2048 bytes of data to the designated flash block). At the end of a page program operation, a program status operation is executed. If the operation is errored, the location of the bad block is recorded by the slice FPGA. This information will be read by the ICU after the readout is completed so that the ICU can update a list of bad blocks, removing them from the list of blocks available for file storage.

Each time a filled page buffer is written to the designated block, a page counter in the flash controller is incremented. When the number of pages in a block is reached, the block is full. This causes the flash controller to retrieve the next allocated block from the flash block ID RAM. The number of blocks in use is counted. This process continues as long as the flash controller is enabled (by the enabling bit assigned to the controller in the block enable register). If the number of blocks consumed is greater than the number allocated by the ICU, the flash controller writes a “block overflow” code to its internal error register. It also suspends the acceptance of any further data from the front end. Once the block is disabled, the ICU can read the error code register to determine if the readout was successful. The ICU can also read the count of bytes that were written to the flash and the identities of any errored blocks observed during the process.

The file downlink process is set up by having the ICU write the identities of flash blocks that have been used to store a specific file. The file to be downlinked may be requested from the ground individually or as part of a group of files. The ICU retains the list of flash blocks in which the data resides. The ICU writes the block locations to the flash block ID RAM and the number of blocks for the file. The ICU also programs a file byte count for the file into registers in the flash controller and the FPGA’s downlink control block. Once the set up has been completed, the ICU enables the flash controller and the downlink controller. File data values are extracted by the flash controller and held in the flash page buffers. As soon as a buffer is filled, the downlink controller is notified and begins extracting data from the filled buffer. The downlink control block of the FPGA controls the flow of data because it must convert the bytes into a serial bit stream bracketed by start, parity, and stop bits. Each transmitted byte is counted and when the transmitted byte count equals the file byte count programmed by the ICU, the downlink control block of the FPGA suspends operation. If the process is terminated before this byte count is reached, an error code (early downlink termination) is set in the error register of the downlink control block.

Files are retained in the flash memories even after the data has undergone a downlink operation. This makes the data available if it is necessary to repeat the transmission process. For example, if poor weather conditions prevent the data from being received free of errors, the ground operations may

request the transmission of the file again. Since the files are stored in a random access fashion, individual files are available for extraction. Once a file has been received on the ground, the ICU can be commanded to erase the flash blocks in which the data for the file were stored. The ICU will achieve this by providing the flash block ID RAM with the blocks to be erased and by writing the "erase flash blocks" command code (along with a count of the number of blocks to be erased) to the flash controller. If flash operation status queries result in the identification of errored blocks, the ICU can obtain this information and remove such blocks from the list of available blocks for use in future image storage operations.

VI. NIR Channel Operation Sequence

NIR slice channels are operated differently due to the fact that the data will not be compressed concurrently with the readout process. As mentioned earlier, multiple readout operations are executed for a given exposure of the NIR detector. The final value stored for a given pixel is obtained by summing several "negative" readouts (resulting from readout of the detector before exposure), with a number of "positive" readouts (resulting from readout of the detector after exposure). The resultant sum is equivalent to the difference of the "after exposure" readout images with the "before exposure" readout images. The result is scaled by simple shifting by a programmable number of bits prior to storage in the SDRAM.

During the readout of NIR channels, the flash controller is not active. Instead, the accumulator controller and the SDRAM controller are enabled. Data accumulation is performed in one of two accumulation buffers by either adding to or subtracting from earlier accumulated values which are held in locations in SDRAM (one location for each pixel). The accumulator controller acts concurrently with the SDRAM controller using the shared pair of accumulation buffers. When a buffer is filled with accumulated data from an ongoing readout cycle, that buffer is turned over to the SDRAM controller which commits the partial sum to designated locations in the SDRAM. The SDRAM controller must write this data out to storage, pre-load the next buffer of data from another block of SDRAM locations, and meet its refresh cycle timing commitments before the second buffer has filled with accumulated data. A separate refresh counter will autonomously increment and the SDRAM controller will, between data transfer operations, read the counter. The count that is recorded will be used to execute an equal number of refresh cycles before the SDRAM controller returns to the activities of reading and writing from the accumulation buffers.

While the accumulator controller only needs to be enabled for readout operations (there is only one commanded state), the SDRAM controller needs to be enabled for data readout (accumulation) operations as well as for transfer to data compression operations. The SDRAM controller will extract the final averaged data for the n^{th} exposure during the m^{th} exposure period where m = n + 1. During the data compression phase, the SDRAM controller, the data compression blocks, and the flash controller will all be enabled. The flash block ID RAM will have been set up with the allocated flash blocks as well. The SDRAM controller is responsible for providing the necessary word strobes to the data compression block as it makes data from the SDRAM available to the compression block.

VII. Design Prototyping

Many of the features described here have been tested in a prototyping design called the TSDC Firmware Development Card. This card includes an Actel A3P1000 flash-based FPGA, 8 Gbits of flash memory, and 256 Mbits of SDRAM. This card emulates the slice hardware. The emulation of the front end (for delivery of image data) and the ICU control unit (for commanding the slice FPGA) is provided by a Programmable Test Adapter (PTA), a general purpose data acquisition board developed by the Electronic Systems Engineering Department at the Fermi National Accelerator Laboratory. The PTA occupies a PCI slot in a test bench PC running the Windows XP operating system. The application software for commanding the slice operations is written using Microsoft Visual C++. The major functions corresponding to slice operations (set up, readout, data compression, downlink, and flash block erase) are provided as button and text box controls to aid in the operation and debugging of the system.

While prototyping has made use of a reprogrammable FPGA, components of the system have been evaluated to obtain estimates of the resource utilization required in a typical space-qualified FPGA (an Actel RTAX2000S antifuse-based FPGA). A brief summary of these estimates is provided in Table 1:

TABLE 1
FPGA Logic Resource Estimates (RTAX2000S)

Functional Block	*R Cells*	*C Cells*
Lossless Compression: Option Sequence Length Evaluation	8 %	8 %
Lossless Compression: Compressed Data Formatting	6 %	5 %
Lossless Compression: Prediction Error Mapper	1 %	1 %
Lossless Compression: All Blocks	*17 %*	*17 %*
Flash Memory Controller	2%	6 %
SPI-like Communications Interfaces	2%	2 %
Total	*23 %*	*26 %*

The figures listed represent the percentage of the total number of respective cell type available in the indicated device. Not all functional blocks for the design have been estimated at this time.

In addition to logic resources, SRAM resources in the FPGA are also required to implement features such as the

flash block ID RAM, flash memory page buffers, CCSDS packet format buffers, accumulation buffers (for NIR channel processing), and pre-scaler lookup table storage. Estimates of the SRAM resource utilization to support these features are provided in Table 2 for the same space-qualified FPGA:

TABLE 2
FPGA SRAM RESOURCE ESTIMATES (RTAX2000S)

FPGA SRAM Functional Block	*Bytes*	*Device Utilization*
Flash Block ID RAM	512	1.4 %
Flash Memory Page Buffers	4096	11 %
CCSDS Source Packet Buffers	4096	11 %
NIR Channel Accumulator Buffers	2048	5.5 %
Square Root Pre-scaler Lookup Table	8192	22 %
Total	*18944*	*51 %*

The figures listed represent the percentage of the total number of SRAM bits available in the indicated device.

In order to protect against single event upsets (SEUs) in the design, error detection and correction blocks will be implemented to envelope the sensitive memory portions of the design. For example, the lookup table for the optional pre-scaler needs to be protected so such measures will be applied to this table. Other portions of the design that may benefit from this approach include the accumulator buffers in the NIR channel processing and the flash memory page buffers. The data being written in packet format will have error detection bits added before the data are written to the flash.

ACKNOWLEDGMENT

The authors wish to thank Chris Bebek, Robert Abiad, and Henry Heetderks of the Lawrence Berkeley National Laboratory and Gunther Haller of the Stanford Linear Accelerator Center for helpful suggestions.

REFERENCES

[1] G. Aldering et.al., “Overview of the Supenova/Acceleration Probe (SNAP)”, Proc. SPIE Vol 4835, pp/ 146-157, 2002.
[2] “CCSDS Recommendation for Lossless Data Compression”, CCSDS 121.0-B-1, Blue Book, May 1997.
[3] H. Lin, J. Marriner, “Preliminary Report on Lossy Image Compression by Square Root Prescaling”, unpublished.
[4] “CCSDS Report Concerning Telemetry: Summary of Concept and Rationale”, CCSDS 100.0-G-1, Green Book, December, 1987.

Alan Prosser received a B.S. degree in physics from Case Western Reserve University in 1979. He received an M.S. degree in electrical engineering from Case Western Reserve University in 1982. He has been employed by the Applied Physics Laboratory at Johns Hopkins University, AT&T Bell Laboratories, and Agere Systems. Since August, 2004, he has been with the Electronic Systems Engineering department at the Fermi National Accelerator Laboratory in Batavia, IL.

Guilherme Cardoso received the B.S. degree in electrical and computer engineering in 1998 from the Universidade Federal do Rio Grande do Sul, Brazil. He received M.S. and Ph. D. degrees in electrical and computer engineering from the Illinois Institute of Technology in 2000 and 2004, respectively. He received an MBA degree from the University of Chicago in March, 2007. He was employed at the Fermi National Accelerator Laboratory from 1998 until February, 2007. Currently he is Vice President of Technology with Aguila Technologies, Inc. in San Marcos, CA.

John Chramowicz received an A.A.S degree in electronic engineering technology from DeVry University in 1985. He received a B.S. degree in electronic engineering technology from DeVry University in 1986. He has been employed at the Fermi National Accelerator Laboratory since August, 1986. He is currently with the Electronic Systems Engineering department at Fermilab.

John Marriner received a B.A. degree in physics from the University of California, Berkeley, in 1970. He received a Ph. D. degree in physics from the University of California, Berkeley in 1977. He was employed at the Lawrence Berkeley National Laboratory from 1974 through 1978. He has been with the Fermi National Accelerator Laboratory since 1978. He is currently a member of the Experimental Astrophysics Group at Fermilab. He is a fellow of the American Physical Society.

Ryan Rivera received a B.S. degree in computer engineering from the University of Illinois, Urbana-Champaign in 2004. He received an M.S. degree in electrical engineering from the University of Illinois, Urbana-Champaign in 2006. Since July, 2006, he has been a member of the Electronic Systems Engineering department at the Fermi National Accelerator Laboratory in Batavia, IL.

Marcos Turqueti received a B.Sc. degree in electronics from UFRGS, RS, Porto Alegre, Brazil in 1999. He received an M.S. degree in Electronics Systems and Devices from ITA, SP, Sao Jose dos Campos, Brazil in 2001. Since September, 2001, he has been a member of the Electronic Systems Engineering department at the Fermi National Accelerator Laboratory in Batavia, IL.