Adapting the Reconfigurable SpaceCube Processing System for Multiple Mission Applications

David Petrick, Daniel Espinosa, Robin Ripley, Gary Crum, Alessandro Geist, and Thomas Flatley

NASA Goddard Space Flight
Greenbelt, MD 20771
david.j.petrick@nasa.gov

Abstract—**This paper highlights the methodology and effectiveness of adapting the reconfigurable SpaceCube system to solve complex application requirements for a variety of space flight missions. SpaceCube is a reconfigurable, modular, compact, multi-processing platform for space flight applications demanding extreme processing power. The SpaceCube system is suitable for most mission applications, particularly those that are computationally and data intensive such as instrument science data processing. We will show how the SpaceCube hybrid processing architecture is used to meet data processing performance requirements that traditional flight processors cannot meet.**

This paper discusses the flexible computational architecture of the SpaceCube system and its inherent advantages over other avionics systems. The SpaceCube v1.0 processing system features two commercial Xilinx Virtex-4 FX60 Field Programmable Gate Arrays (FPGA), each with two embedded PowerPC405 processors. The FPGAs are mounted in an innovative back-to-back method, which reduces the size of the circuit board design while maintaining the added benefit of two FPGAs. All SpaceCube v1.0 cards are 4" x 4", yielding a small, yet powerful hybrid computing system. The architecture exploits the Xilinx FPGAs, PowerPCs, and necessary support peripherals to maximize system flexibility. Adding to the flexibility, the entire system is modular. Each card conforms to a custom mechanical standard that allows stacking multiple cards in the same box.

This paper will detail the use of SpaceCube in multiple space flight applications including the Hubble Space Telescope Servicing Mission 4 (HST-SM4), an International Space Station (ISS) radiation test bed experiment, and the main avionics subsystem for two separate ISS attached payloads. Each mission has had varying degrees of data processing complexities, performance requirements, and external interfaces. We will show the methodology used to minimize the changes required to the physical hardware, FPGA designs, embedded software interfaces, and testing.

This paper will summarize significant results as they apply to each mission application. In the HST-SM4 application we utilized the FPGA resources to accelerate portions of the image processing algorithms more than 25 times faster than a standard space processor in order to meet computational speed requirements. For the ISS radiation on-orbit demonstration, the main goal is to show that we can rely on the commercial FPGAs and processors in a space environment. We describe our FPGA and processor radiation mitigation strategies that have resulted in our eight PowerPCs being available and error free for more than 99.99% of the time over the period of four years. This positive data and proven reliability of the SpaceCube on ISS resulted in the Department of Defense (DoD) selecting SpaceCube, which is replacing an older and slower computer currently used on ISS, as the main avionics for two upcoming ISS experiment campaigns. This paper will show how we quickly reconfigured the SpaceCube system to meet the more stringent reliability requirements.

TABLE OF CONTENTS

1. INTRODUCTION

SpaceCube is a family of Field Programmable Gate Array (FPGA) based on-board science data processing systems developed at the NASA Goddard Space Flight Center (GSFC) [1]. The goal of the SpaceCube program is to provide 10x to 100x improvements in on-board computing power while lowering relative power consumption and cost. SpaceCube is based on the Xilinx Virtex family of FPGAs, which include processor, FPGA and digital signal processing (DSP) resources. These processing elements are leveraged to produce a hybrid science data processing platform that accelerates the execution of science data processing algorithms by distributing computational functions among the elements. This approach enables the implementation of complex on-board functions that were previously limited to ground based systems, such as on-board product generation, data reduction, calibration, classification, event/feature detection, data mining and real-time autonomous operations. The system is fully reconfigurable in flight, including data parameters, software and FPGA configuration, through either ground commanding file transfers or autonomously in response to detected events/features in the instrument data stream.

Background

The SpaceCube processing system was started at GSFC in 2006 with Internal Research and Development (IRAD) program funding [2]. A series of internal prototype demonstrations to NASA officials showcased the computational power and its inherent reconfigurable advantages over typical space processors. NASA recognized the clear potential of the technology, and provided the funding needed to increase the technology readiness level (TRL) for space flight applications. Specifically, the Hubble Space Telescope Servicing Mission 4 management team infused SpaceCube as the main avionics for an experimental payload called Relative Navigation Sensors (RNS) [3]. The use of SpaceCube within the RNS system will be described in detail later in this paper.

The version of the SpaceCube that was initially developed in the 2006-2009 timeframe is known as SpaceCube v1.0. Follow-on versions have been developed [1]; however the design and use of SpaceCube v1.0 will be the focus of this paper.

2. HYBRID FLIGHT COMPUTING

There is a growing need for higher performance processing systems for space. Instrument precision and speed capabilities are rapidly evolving which levies tougher electrical interfacing and data bandwidth requirements on the computing node of the system. In addition, on-board processing of the data products, in some cases in real-time, is now a common requirement.

On-board processing improves system efficiency and functionality in two areas. First, by allowing the spacecraft to preprocess data products on board, a smaller or compressed data volume per data set can be sent to ground, which increases the amount of time an instrument can be turned on and collecting data. It is typical for high data rate science instruments to constrain their data collection to 10-20% of the mission time to fit within the limited downlink bandwidth. This problem continues to grow as instrument capabilities increase. Second, it enables for applications on board the spacecraft to make autonomous decisions on the processed data products. This ability opens up a much more challenging range of mission objectives that can be targeted for space applications.

Typical space processing systems generally consist of a single radiation hardened processor such as the BAE RAD750, Aeroflex LEON3FT, BroadReach BRE440, or General Dynamics ColdFire which deliver less than 300 DMIPS. These standard processing systems are very good at providing general services such as Command and Data Handling (C&DH), Guidance and Navigation Control (G&NC), and simple instrument control. These processing systems are not good candidates for applications that require implementing fast computations of complex algorithms on a high bandwidth or large volume data source.

Another common component found in typical space processing systems is the anti-fuse FPGA, which generally have very good radiation immunity. The corresponding circuit board and FPGA architectures are designed for a set of very specific mission requirements. However, these architectures are very hard to design and intrinsically expensive to change such that they are portable to multiple missions, dynamic functional requirements, or new post-launch mission objectives or corrections.

A new approach is needed to meet the increasing challenges required by space processing systems. A hybrid computing system that combines multiple processors, reconfigurable FPGAs, flexible interface options, with a modular architecture is the solution that will bridge the gap between todayôs avionics requirements and yesterdayôs typical stand-alone sequential processing architecture. A hybrid computing architecture is able to retain the function of a multi-purpose computer that runs typical C&DH and G&NC. However, in addition to these types of tasks, it has the advantage of supporting computationally complex tasks that require FPGA co-processors to handle math such as FFT, matrix manipulation, parallel floating point operations, or implementing an advanced interface such as CameraLink, Spacewire, gigabit Ethernet, or support the implementation of a custom interface.

The modularity of such a system allows for the quick adaptation to changing avionics requirements. A modular system, for example, can support adding a bulk memory card, a custom electrical interface, or expand the I/O bandwidth required. A modular and reconfigurable system yields a high probability of using the same basic avionics package for different mission applications, or follow-on missions, even if interface and computing requirements are drastically different.

SpaceCube fits the need of a hybrid, reconfigurable, modular space processing system. This paper will show how cost and schedule can be reduced by reusing the same basic system for new missions. Reuse of hardware architecture greatly reduces the amount of up front Non Recurring Engineering (NRE) costs and time associated with building a new system with new requirements from the ground up.

3. SPACECUBE V1.0 DESCRIPTION

The SpaceCube v1.0 system is a compact, modular, low power, reconfigurable multiprocessing platform for space flight applications demanding extreme computational capabilities and/or high data rate/volume interfaces. The SpaceCube v1.0 processing system is based on the Xilinx Virtex-4 FX60 FPGA that includes two embedded hard IP PowerPC405 processors.

This specific FPGA was the subject of radiation testing and characterization by many groups including, but not limited to the Xilinx Radiation Test Consortium and the GSFC Radiation Effects and Analysis Group [10-16]. The SpaceCube design leverages this work to properly mitigate radiation effects within the system, as will be discussed later in this paper.

A. Modular Stacking Architecture

The SpaceCube v1.0 mechanical design uses a custom stacking architecture. The system is comprised of various slices that are stacked together using a 122-pin connector from IEH Corporation [7]. The system uses a dual-redundant I2C bus for low data rate transfers between all cards in the stack. Each card is given a unique address on the bus. The base system requires a power slice and a processor slice. This architecture allows for adding mission unique cards, if necessary. Four rods are used to hold the box together once all slices have been mated together. Figure 1 depicts the SpaceCube v1.0 modular architecture. This version of the system required five slices (2 power, 2 processor, 1 I/O). Figure 2 shows the picture of the flight box that corresponds to the model in Figure 1. This configuration of the system is 7.5-lbs. and is 5-in x 5-in x 7-in in size [9].

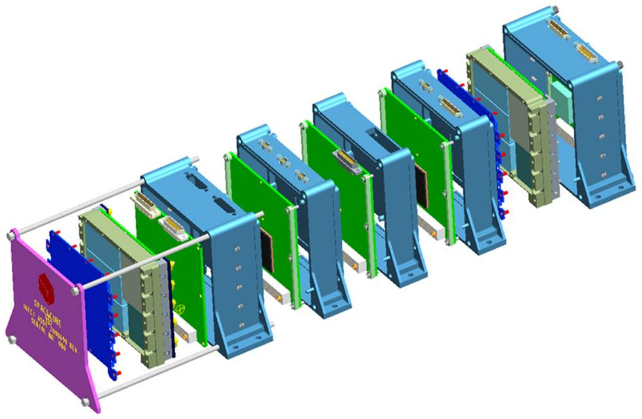


Figure 1 - SpaceCube v1.0 Modular Slice Architecture

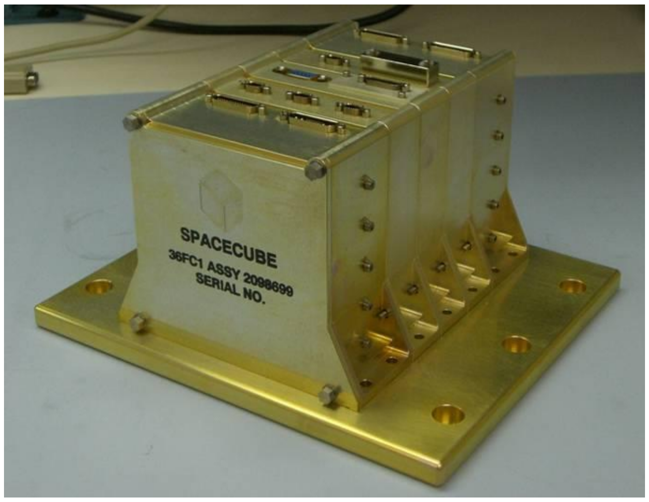


Figure 2 - SpaceCube v1.0 Flight System

Each circuit board within the system is 4-in. x 4-in in size. A mechanical tray holds the card in place and allows the stacking connector to protrude through the bottom of the slice. The card edges are bolted down to its respective slice enclosure. In addition to the structural mount, the card edge is also the thermal interface for each card. Figure 3 shows the flight processor card slice enclosure.

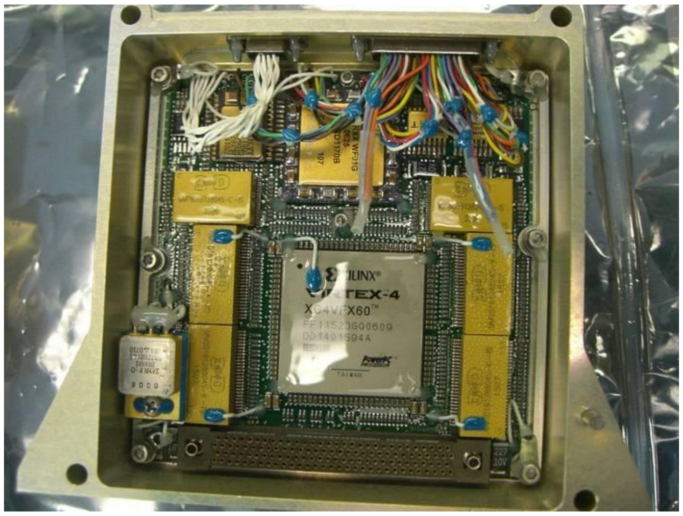


Figure 3 - Slice Enclosure of Processor Card

B. Power Slice Design

The power slice consists of two circuit boards. The Low Voltage Power Card (LVPC) has the typical EMI filter and DC/DC components found in space flight power supplies. The LVPC will accept 28V +/- 8V and provide 5.0V, 3.3V, 2.5V, 1.5V, and +/- 12V to the stacking connector. On power-up, 2.5V, 3.3V, and 5.0V are automatically turned on to support the main controller circuitry on the processor card. The LVPC supports switched services for 1.5V, 2.5V, 3.3V and +/-12V. The main controller on the processor card switches on these services by commands to support the Xilinx FPGAs. The processor card has a custom point-of-load circuit that regulates the 1.2V required for the core voltage of the Xilinx devices.

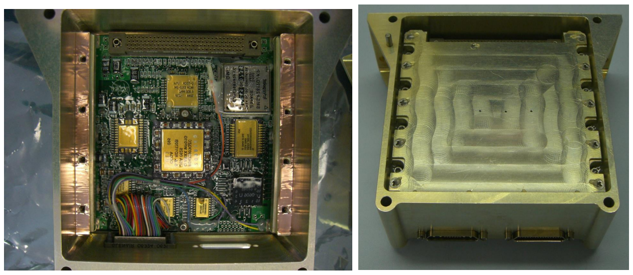


Figure 4 – Power Slice Assembly of DCC and LVPC

The second card in the power slice is the Digital Control Card (DCC). The DCC supports various functions including collecting local voltage and temperature data, SpaceCube 10BASE-T Ethernet and 1553 interfaces, controlling processor reset and power loss warning signals, and switching load services on the LVPC. An Aeroflex FPGA is used to control these functions and to communicate with the processor card via the I2C bus.

The LVPC and DCC cards are stacked together inside of one enclosure slice. The LVPC assembly requires a heat sink to handle the power loss within the DC/DC bricks. Figure 4 shows the assembly of the power slice. On the left, the DCC sits at the bottom of the enclosure. Two side rails are installed above the DCC which are seen along the edges of the chassis. Next, the LVPC is mated to the DCC board and bolted to the side rails. The LVPC is shown on the right with the heat sink installed to its circuit card assembly.

C. Processor Card Design

The processor card features two Xilinx Virtex-4 FX60 devices in a back-to-back fashion. Figure 5 shows that the processor board fully utilizes both sides of the circuit board. Each Xilinx FPGA has two embedded PowerPC405, each capable of 750+ DMIPS. This results in four processors per card yielding a total processing capacity of 3000 DMIPS/card in addition to the 113,760 logic cells, 256 DSP slices, and 8,352 Kb of Block RAM resources [24].

The card features a good balance of peripherals to support the Xilinx FPGAs and processors given the limited board space. Peripherals include four 256MB Synchronous Dynamic Random Access Memory (SDRAM) modules, two redundant 512MB NAND flash modules, 20 configurable full duplex LVDS/RS422 interface modules, 42 stacking connector I/O, and required clock and power circuitry. The front panel I/O connectors are configurable as required by each application. Figure 6 shows a high level component diagram of the processor card design.

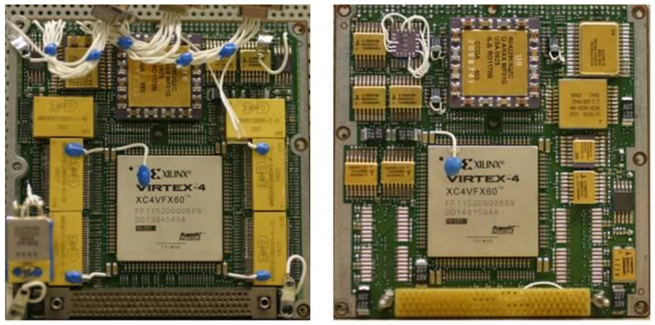


Figure 5 - Processor Card, Top and Bottom Sides

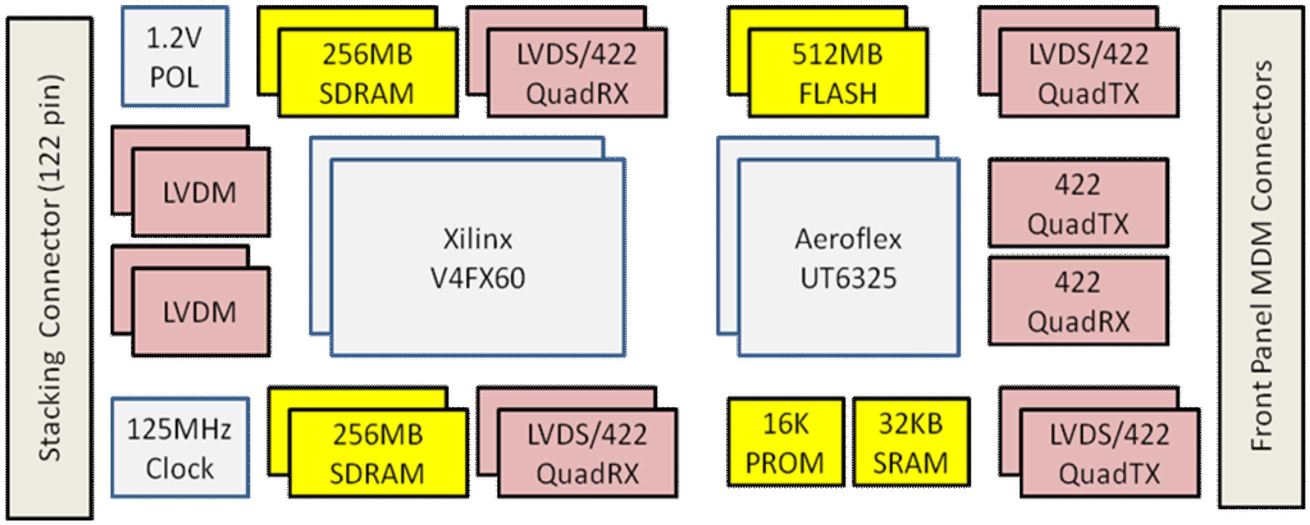


Figure 6 - High Level Block Diagram of Processor Card

Aeroflex Service Design—Two back-to-back Aeroflex FPGAs control the power sequencing of the switched power rails via I2C commands, reset control, watchdog timers, mission elapsed timer, scratch-pad ram, Xilinx configuration, non-volatile storage access, and monitor health and status. To provide all of these services as well as facilitate reconfiguration and reuse of the one-time-programmable (OTP) Aeroflex FPGA an embedded 8-bit soft-core microcontroller (SpaceRISC) was designed and used as an alternative to using a complex state machine. This design decision has proven useful in not only enhancing the services provided but also allowed for debug and test code to be loaded into the OTP FPGA to better facilitate initial board testing as well as system integration and test activities.

The SpaceRISC is based on a standard commercial device that can only address 16KB of the 32KB SRAM. We turned this limitation into a benefit by developing a memory controller that could conditionally operate out of the top or bottom half of the memory while simultaneously providing a side channel for read/write access to the ǿinactiveô portion of ram. This facilitates a complete reconfiguration of the microcontroller while the current flight application is still running. The first stage boot loader (FSB) for the SpaceRISC is stored in an onboard radiation hardened 16KB PROM. The SpaceRISC cannot execute code directly out of the PROM so a hardware boot-loader IP core was created to copy data from the PROM to the external SRAM before bringing the SpaceRISC out of reset. The SpaceRISC FSB will then search for the latest Run-Time Application (RT-App) to load and execute. The RT-App is stored in a Quad-Triple Module Redundant manner; should the FSB not successfully load the RT-App it will fall back to the previous version. The RT-App is fully capable of performing this boot loader sequence from ground commands or alternative configuration files which allows multiple variants of the RT-App to be stored in flash while still preserving the ôGoldö boot configuration.

As the RT-App starts to execute it will first check to see if the startup was due to a watchdog timer reset or a clean power up. In the event of a watchdog timeout (WDT) the RT-App will check the configuration table for a set of flags to determine the next course of action. The current flight configuration allows for a programmable threshold of WDT and reverts to the ôGoldö application code should it exceed the threshold. After a nominal proceed condition is met the RT-App needs to enable the Xilinx FPGAÆs by turning on the switched power rails and configuring the FPGA. The bitstreams used for configuration are determined by a configuration file that is stored in flash.

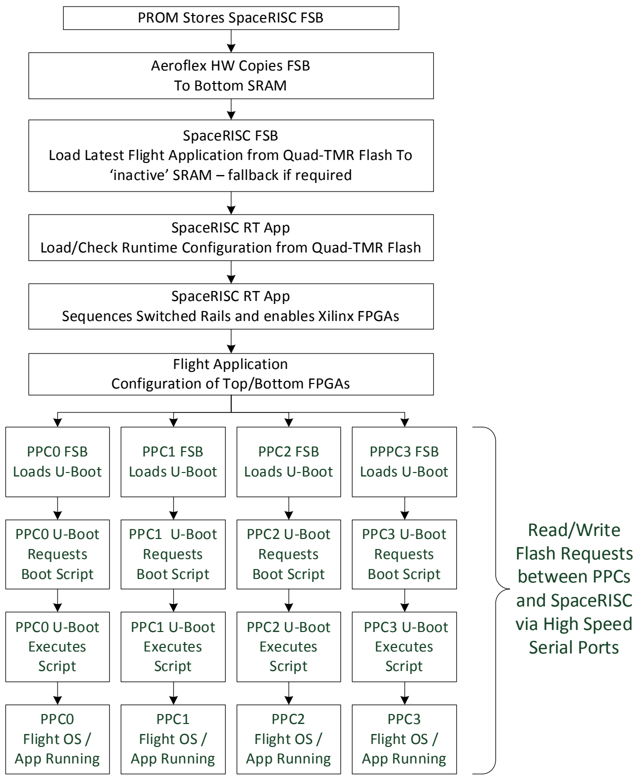

PROM Stores SpaceRISC FSB
Aeroflex HW Copies FSB To Bottom SRAM
SpaceRISC FSB Load Latest Flight Application from Quad-TMR Flash To 'inactive' SRAM – fallback if required
SpaceRISC RT App Load/Check Runtime Configuration from Quad-TMR Flash
SpaceRISC RT App Sequences Switched Rails and enables Xilinx FPGAs
Flight Application Configuration of Top/Bottom FPGAs
PPC0 FSB Loads U-Boot
PPC1 FSB Loads U-Boot
PPC2 FSB Loads U-Boot
PPPC3 FSB Loads U-Boot
PPC0 U-Boot Requests Boot Script
PPC1 U-Boot Requests Boot Script
PPC2 U-Boot Requests Boot Script
PPC3 U-Boot Requests Boot Script
PPC0 U-Boot Executes Script
PPC1 U-Boot Executes Script
PPC2 U-Boot Executes Script
PPC3 U-Boot Executes Script
PPC0 Flight OS / App Running
PPC1 Flight OS / App Running
PPC2 Flight OS / App Running
PPC3 Flight OS / App Running
Read/Write Flash Requests between PPCs and SpaceRISC via High Speed Serial Ports

Figure 7 - Simplified Processor Boot Sequence

The RT-App then brings the PowerPC (PPC) processors out of reset and the PPCs start to execute their First Stage Bootloaders (PPC FSB). The PPC FSB will then request the second stage bootloader from the SpaceRISC; we have chosen to use UBoot as our second stage bootloader. UBoot will then request a boot script from the SpaceRISC that contains commands and the file addresses required to load the flight operating system and applications. The files are read from the SpaceRISC with a series of ôget file infoö commands and ôflash read requestsö. The ôget file infoö commands take in a file ID that is translated to a flash address by the SpaceRISC. The SpaceRISC then reads the file headers and send it back to the PPC. This header contains information about the file address in flash, data CRC length and if the image is mirrored across multiple devices. The PPC and SpaceRISC then perform a series of flash read request and response packets until all of the data is transmitted and UBoot can load the OS/Application. The simplified boot sequence is shown in Figure 7.

Flash File Mitigation—NAND Flash technology is known to be susceptible to radiation Single Event Effects (SEE) including Single Event Upsets (SEU) and Single Event Function Interrupts (SEFI). Each processor card flash module is composed of four independent dies inside. The SpaceRISC NAND Flash Controller has the capability of performing mirrored read and write operations, which store the same file in one or more die. In addition, software in the SpaceRISC has the capability of adding Triple Modular Redundant (TMR) duplication of each file within each die. For the most mission critical data such, as the SpaceRISC configuration tables and software, we utilized Quad-redundant with byte level TMR (QTMR). As the SpaceRISC reads a file it will read in three bytes at a time and perform a series of bit-wise AND/OR operations on the data set (1). The output of this operation is then byte-wise ANDed to the input data set (2). To mitigate the possibility of two bit flips in the same bit position resulting in a false positive output, the voted results are compared to the input values and if any of the values do not match the voted output the system moves onto the next mirrored copy of the file at the current file offset. In the event that all copies indicate some kind of error we will use the voted output from the last test. This coupled with four checksums per page and checksums on all files helps to detect multiple bit flips in the NAND Flash before they are used by the system.

$$\text{TMR\_RESULT} = (d0 \,\&\, d1) \mid (d0 \,\&\, d2) \mid (d1 \,\&\, d2) \qquad (1)$$

$$\text{TMR\_ERROR\_n} = (\text{TMR\_RESULT} \,\&\&\, d0) \parallel (\text{TMR\_RESULT} \,\&\&\, d1) \parallel (\text{TMR\_RESULT} \,\&\&\, d2) \qquad (2)$$

Due to that fact that QTMR is partially implemented in software it would require extending the boot time to utilize this method for larger files such as those for Xilinx bitstreams and the PowerPC OS and ramdisks. To minimize our boot time and thus increase our availability larger images are store in a Quad Mirrored fashion with DMA read and transmit assist. When a flash read request if received by the SpaceRISC the request is validated and a NACK packet is sent in response if any errors are detected otherwise the SpaceRISC will setup a flash read response and transmit the packet header information, the hardware in the Aeroflex FPGA will read data from the flash and place it directly into the transmit buffer while also calculating the data checksum which will be added to the header checksum to allow for general validation of the response packet. The data transmitted also include the Out Of Bounds area of the NAND flash that is used to store error correcting codes for the NAND Flash page. Software in the PowerPC will then check that the page is valid or request a new page from the next mirrored copy.

*Xilinx Configuration Scrubbing*ù The task of monitoring the programmable configuration bits within the Xilinx FPGA is typically handled by an outside controller. The

Aeroflex FPGA designs and processing load on the SpaceRISC were considered to be at full capacity. The SpaceCube v1.0 Xilinx FPGAs contain an internal TMR self configuration scrubber that utilizes the ICAP and FRAME_ECC [12]. The Aeroflex FPGA is responsible for enabling this service. The scrubber core reports status to the Aeroflex FPGAs that it is actively scrubbing, if it has detected and corrected an SEU, or if it has found an uncorrectable error as a result of a Multiple Bit Upset.

D. FPGA Design and Software Design Methodology

FPGA development for the SpaceCube Xilinx FPGAs requires the standard Xilinx tool chains (ISE, EDK). We have developed a baseline FPGA design that includes the necessary framework for an embedded system using the PowerPCs on the SpaceCube. This baseline system is given to developers as a starting point for porting a new application to the SpaceCube environment. Similar FPGA designs have also been developed for the ML403 and ML410 Xilinx development boards. This allows for a cheaper development cycle for application engineers prior to targeting the SpaceCube system.

The PowerPCs within the Xilinx FPGAs on the SpaceCube currently support standalone code, Linux, VxWorks, and RTEMs operating systems (OS). The SpaceCube software team has modified an existing Linux OS and fine-tuned it to support the SpaceCube build environment (SpaceCube Linux).

The SpaceCube system is easy for application engineers to target and allows for a fast development cycle. We have supported more than 10 projects inside and outside of GSFC using this development approach. All cases have resulted in a seamless application port to the SpaceCube hardware system.

4. MISSION USE CASES

This section will present four examples of how the SpaceCube v1.0 system was adapted to support different missions. For each mission, we will describe the mission and its objectives, the corresponding SpaceCube hardware requirements and changes, FPGA and application descriptions, integration and testing, operations, and an assessment on overall development effort.

A. Relative Navigation Sensors

On May 11, 2009, STS-125 Space Shuttle Atlantis, lifted off from Kennedy Space Center (KSC) with new instruments, gyroscopes, and flight computers for the Hubble Space Telescope. The HST Servicing Mission 4 (SM4) saw almost 37 hours of astronaut Extra-Vehicular Activity (EVA) time to install the instruments and hardware, and overcome many obstacles in servicing the observatory. Along for the ride on this mission, installed in the back of the shuttle payload bay on the Multi-Use Logistics Equipment (MULE) carrier, was a technology flight experiment called the Relative Navigation Sensors (RNS) system [1, 3, 4, 6, 9].

The RNS system, which was a driving technology requirement for the HST Robotic Servicing and De-orbit Mission (HRSDM), consists of three cameras, a GPS module, two redundant Mass Storage Modules (MSM) that each contains four hard drives, a Telemetry Module (TM), a SpaceCube v1.0 system as the payloadôs central avionics and a dual redundant ground terminal. The two main objectives of RNS were to record imagery of HST during rendezvous and deploy operations, and also to demonstrate the capability of providing real-time tracking and position estimation on HST with the SpaceCube processing system. The RNS flight hardware is pictured in Figure 8.

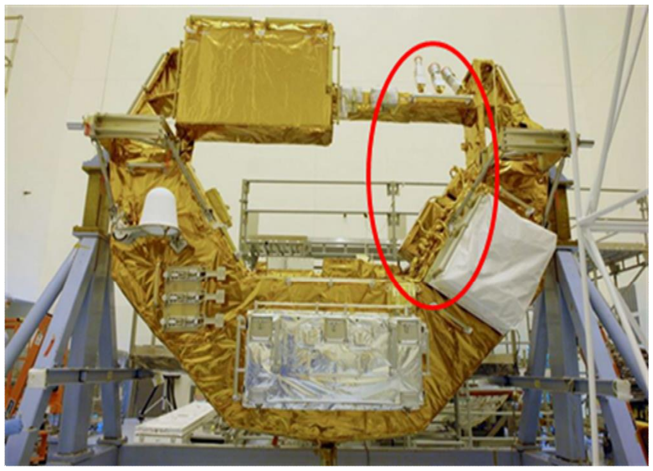


Figure 8 - RNS Payload Installed on MULE Carrier

RNS SpaceCube Systemð The RNS SpaceCube consisted of two SpaceCube processor cards (SCuP), two power slices, and a custom card called the Video Interface Module (VIM). Processor card 1 hosted two different position estimation and image tracking algorithms, the Goddard Near Feature Image Recognition (GNFIR) and Ultra Lethal Targeting by Optical Recognition (ULTOR). Processor card 2 handled multiple tasks including GPS, MSM, TM, Automatic Gain Control (AGC), C&DH and shuttle KU-band in a single Xilinx.

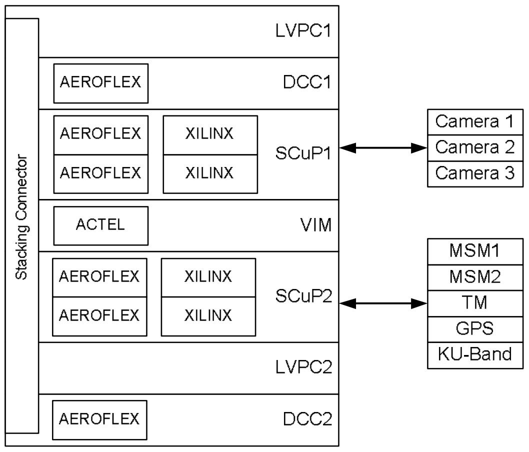


Figure 9 - RNS SpaceCube Diagram

The VIM was responsible for compressing images. The compressed images were stored during critical operations and sent to ground operators via the processor card 2 shuttle KU link. This SpaceCube, pictured in Figure 2 prior to RNS payload integration, was approximately 5-in. x 5-in. x 7-in. in size and required a nominal power of 37W (7-8W per processor card). A high level SpaceCube diagram is shown in Figure 9.

*FPGA Design*ô Three of the four Xilinx FPGAs in the SpaceCube were 60-70% utilized. The fourth FPGA was for design contingency, but was never needed and remained unprogrammed. Two FPGA designs used one PowerPC and the ULTOR application FPGA design used both PowerPCs. The FPGA designs consisted of the required embedded system peripherals, internal card-to-card infrastructure, RNS interface peripherals such as the custom camera core, an internal triplicated self scrubbing configuration module, and hardware acceleration co-processing cores.

A major part of the RNS experiment on HST-SM4 was the GNFIR pose estimation application. One of the two Xilinx FPGAs in the processor card 1 hosted the GNFIR application. In order to meet real-time processing requirements, the GNFIR system had to operate at 3 Hz. Initially, GNFIR was developed exclusively in software and run on the embedded PowerPC405 processor in the Xilinx. However, the performance using the processor alone was insufficient and GNFIR could only operate at 0.125Hz. In order to improve the application performance, the Floating Point Unit (FPU) FPGA IP core was added to the PowerPC, which resulted in a 4x speedup to 0.5Hz. We developed the custom Edge core in FPGA to accelerate some of the more compute intensive operations in GNFIR that resulted in an additional 6x speedup that enabled the application to operate at the required 3Hz [4]. The Edge core provides an FPGA implementation of the edge detection, gradient direction computations, and centroid computation on the camera image data. The edge detection is performed using a Sobel operator and computes the gradient vector at each image pixel by performing a convolution of two 3x3 filter kernels in the horizontal and vertical dimensions with the image. The gradient magnitude is then computed, and the edge data is scaled by a factor selected via a command register. The gradient direction is calculated using a CORDIC arctangent module. The centroid of each input image was also computed. Each pixel of the input image that is above the threshold in intensity is considered a significant pixel, and the centroid is the average coordinate location of all the significant pixels in the image. The desired threshold value was configurable through a command register by the software driver. The edge core processing engine is fully pipelined and can produce an edge/angle pair at the same rate as the camera data pixels are supplied to it. This data was buffered in a read FIFO in the FPGA core that was connected to the PowerPCôs Processor Local Bus (PLB). This allowed the processor to transfer the data to memory at a high rate using Direct Memory Access (DMA). The reconfigurable nature of the radiation tolerant Xilinx FPGAs in the SpaceCube v1.0 allowed the new Edge core to be developed, integrated, and tested in a matter of months.

Figure 10 shows the high-level diagram of the embedded system design used to implement the GNFIR application. GNFIR ran under the SpaceCube Linux OS.

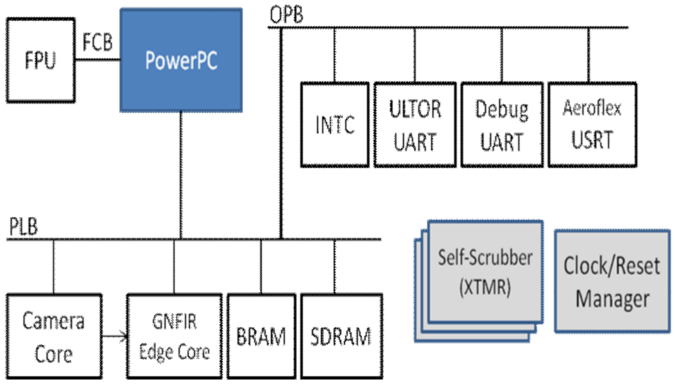


Figure 10 - GNFIR FPGA High-Level Design Example

The ULTOR application FPGA was proprietary, designed by Advanced Optical Systems. The ULTOR and GNFIR applications continuously exchanged HST position estimate data. This was implemented to help speed up the process of acquiring and locking onto the image in order to enter tracking mode. The ULTOR application PowerPC ran under the VxWorks OS [4].

The C&DH FPGA design included an AGC algorithm with supporting FPGA core to dynamically adjust the camera brightness of the image as lighting conditions changed. A custom interface core was necessary to extract streaming data from the GPS receiver. The C&DH design also included a Floating-Point Unit (FPU) core, UARTs for communication with the MSMs and TM, and a KU core to stream continuous data to the shuttleôs KU transponder system. The C&DH PowerPC ran under the SpaceCube Linux OS.

*RNS Testing*ô Preparing the RNS payload for flight was a considerable task due to the number of instruments, interfaces, and configurable operation modes, along with the challenge of obtaining high confidence that the system would track a school bus sized satellite in the space environment. This involved a series of test campaigns at four NASA centers. Three trips with were made to the Flight Robotics Laboratory (FRL) at the Marshall Space Flight Center (MSFC). Testing at MSFC involved a full RNS engineering-level system and a full-scale aft bulkhead and a tenth-scale mockup of HST. Full system integration and operation capabilities including image recording, position estimation and tracking, and AGC were incrementally tested during subsequent trips. As issues arose during testing at the FRL, full advantage was taken of the SpaceCubeôs reconfigurability to fix problems quickly. The command and telemetry capability and KU downlink were tested at Houstonôs Johnson Space Center (JSC). Numerous tests at JSC included shuttle interface testing, ground terminal verification, and mission operations simulations with the entire shuttle ops team. All typical payload integration and environmental testing of the flight system was conducted at GSFC. The most notable test at GSFC took place prior to delivery to Cape Canaveral. With the RNS flight payload integrated onto the MULE carrier, a crane was used to maneuver the full-scale HST bulkhead to test the close proximity rendezvous and deploy sequences (Figure 11).

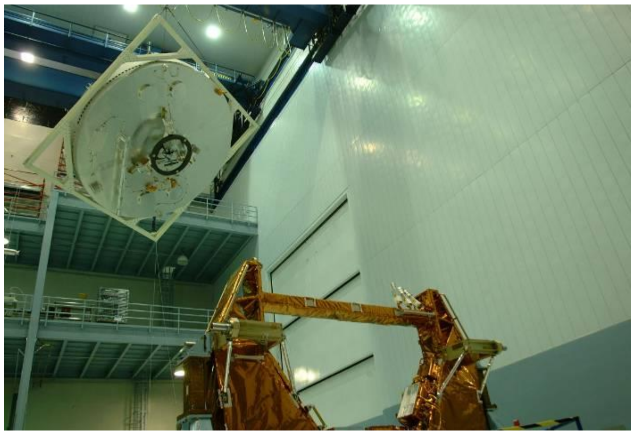


Figure 11 - RNS Flight Payload Testing at GSFC

Final testing and shuttle integration took place at KSC. Two final software updates to the SpaceCube were conducted that fixed minor bugs that were found during ongoing testing at GSFC on the RNS engineering development units. To support all of the different tests, two SpaceCube EDUs in addition to the SpaceCube flight box were built for the project. The reconfigurability of the SpaceCube FPGAs and software were absolutely necessary in addressing the many issues that arose during application development, interface integration, and operation sequence testing. RNS would have missed schedule deadlines if the avionics did not have the ability to quickly adapt to required changes. However, after environmental testing, the FPGA designs were locked down.

Operations—Payload operations were conducted by the RNS team from the Payload Operations Control Center within Houstonôs Mission Control facility. RNS was successful in achieving all of its on-orbit objectives. The GNFIR position and attitude estimation algorithm successfully tracked HST for 21 minutes during rendezvous using the long range camera between ranges of 50 to 100 meters, and also tracked HST during deploy for 16 minutes using the short range camera at a 2 to 3 meter range [4]. RNS recorded a total of 6+ hours of HST imagery (~750GB data) during rendezvous and deploy. GNFIR feature tracking during rendezvous and deploy are shown in Figure 12.

Figure 13 shows a split screen of an image of HST during rendezvous from the long range camera and the real-time GNFIR solution computed on the SpaceCube at that given time.

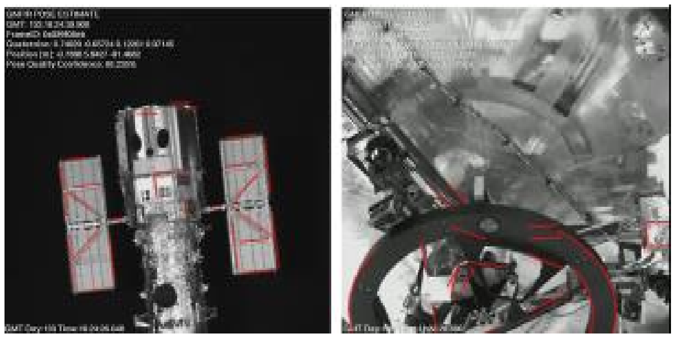
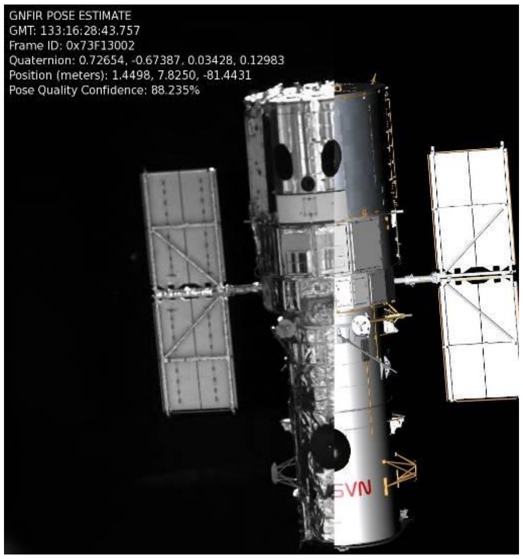


Figure 12 - Real-Time GNFIR Feature Tracking of HST


```
GNFIR POSE ESTIMATE
GMT: 133:16:28:43.757
Frame ID: 0x73F13002
Quaternion: 0.72654, -0.67387, 0.03428, 0.12983
Position (meters): 1.4498, 7.8250, -81.4431
Pose Quality Confidence: 88.235%
```
Figure 13 - GNFIR Estimation Overlay of HST Location

RNS also recovered 100,000+ compressed images over the course of the mission using the video compression capability in the SpaceCube. An example of a compressed image that was downloaded during mission operations is shown in Figure 14.

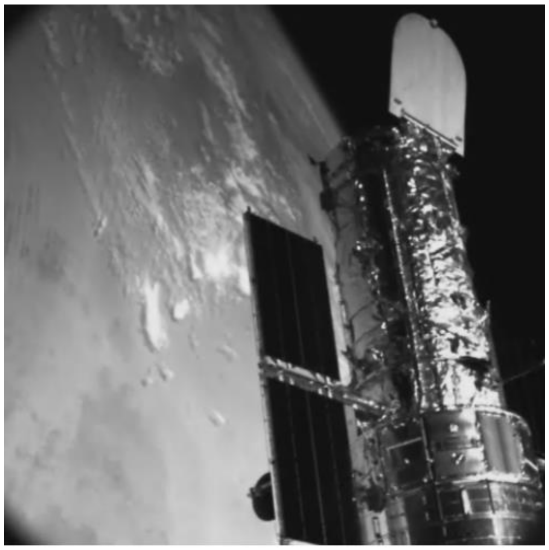


Figure 14 - Compressed Image from HST Release

The RNS system infrastructure did not allow for on-orbit FPGA reconfiguration of the SpaceCube, but did allow for software parameters to be updated in the processor cardô s flash. The AGC parameters in flash were successfully updated to tune the algorithm for the deploy operations. In total, SpaceCube was powered for 60 hours during this mission. Two SEUs were detected and successfully repaired by the scrubber. During a routine KU image downlink, the C&DH PowerPC experienced an SEE, at which point it stopped functioning. The PowerPC WatchDog Timer in the Aeroflex FPGA successfully detected that the heartbeat had stopped and reprogrammed the FPGA, at which point the KU data dump was resumed.

*Development Effort*ð The manpower and schedule to deliver the RNS flight box is significantly greater than subsequent missions using the v1.0 system. This development cycle accounted for all of the NRE required when building a new hardware system with supporting FPGA and software. This includes all electrical engineering design, mechanical design, thermal design, radiation and parts engineering, systems design, anti-fuse FPGA design, Xilinx framework core development (PowerPC, SDRAM, scrubber, etc.), and software development for the SpaceRISC and PowerPCs. This phase of the development took two years and required the equivalent of approximately 20 people/year, or 40 man-years. Next, implementing RNS specific applications involved PowerPC software development, Xilinx FPGA core development, intra-box infrastructure testing, independent box verification, environmental testing, and post-delivery support. The RNS implementation phase for SpaceCube was a simultaneous effort that lasted three years and required the equivalent of approximately 10 people/year, or 30 man-years.

B. MISSE-7

A SpaceCube system was launched to the ISS in November 2009 as part of the Materials International Space Station Experiment 7 (MISSE-7) [16, 22]. MISSE-7 is installed on the ISS Express Logistics Carrier (ELC), specifically ELC-2. The main objectives of the MISSE-7 SpaceCube was to (1) demonstrate reliable use of the commercial devices, in this case Xilinx FPGAs and embedded PowerPCs, for a long duration in the space environment, (2) demonstrate continuous and reliable execution of computation-intensive science data applications utilizing SpaceCubeô s Radiation Hardened by Software (RHBS) technology, (3) demonstrate the ability to reconfigure the FPGA and software with new design files sent from ground.

MISSE-7 SpaceCube System—The flight spare hardware from RNS was used to develop this payload. The MISSE-7 payload transmits and receives telemetry and commands to ISS through the Communication Interface Box (CIB) over a RS485 bus with individual experiment hardware enables. Two processor/power slice pairs were configured as independent experiments with separate command and telemetry interfaces. A new MISSE-7 interface slice was required within the SpaceCube modular stack to fulfill all hardware requirements and to support the two independent SpaceCube experiments. The flight box that was delivered was the same physical size as RNS, but only required 28W of power (14W per processor/power slice pair). A high level diagram of MISSE-7 is shown in Figure 15.

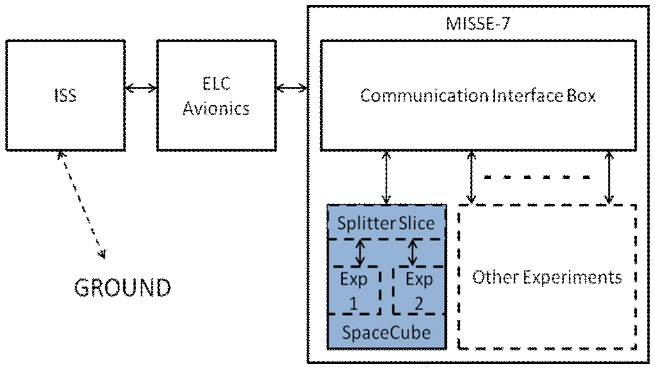


Figure 15 - MISSE-7 System Diagram

FPGA/Software Applications—There is significant re-use of the FPGA and software design from the RNS mission. The initial FPGA design contained framework cores from RNS along with new cores specific to the MISSE-7 experiment. We tested preliminary versions of our RHBS methodologies. One methodology involves running identical applications in two PowerPC in separate FPGAs. Mirrored C&DH applications on both FPGAs coordinate through the SpaceRISC to execute incoming commands and respond to telemetry requests. Each C&DH app receives and processes incoming commands and requests for telemetry from the CIB and then transmits the parameters to the SpaceRISC. Once the SpaceRISC receives a set of parameters from one C&DH app it sets a timer to wait for parameters from the second C&DH app. The SpaceRISC validates the received parameters. If valid parameters were received from both C&DH apps it grants one of the applications the right to process the parameters based on a round robin approach. If only one valid set of parameters were received in the timeout window it grants the right to process to the app with the valid parameters. If none of the parameters are valid then no rights to execution are given.

This RHBS technique allows the C&DH system to generate telemetry and process command free from error. This also allows the system to operate when one FPGA is down.

Along with the C&DH app, the RHBS demonstration experiment continuously runs a Lunar Lander task using data stored in SDRAM. The Lunar Lander application performs part of the calculations needed for an autonomously controlled vehicle to safely land on the surface of the moon avoiding unsafe terrain at the landing area. The results are sent to the SpaceRISC inside of the Aeroflex. The SpaceRISC software, which had to be modified to support MISSE-7, allows for timing windows that each processor has to send incremental results. Different modes are supported for rolling back a processor task if it fails to send data or if the processors become out of sync. The SpaceRISC also has the ability to completely ignore a processor string. A high-level diagram of how the FPGA embedded system is configured is shown in Figure 16 [5].

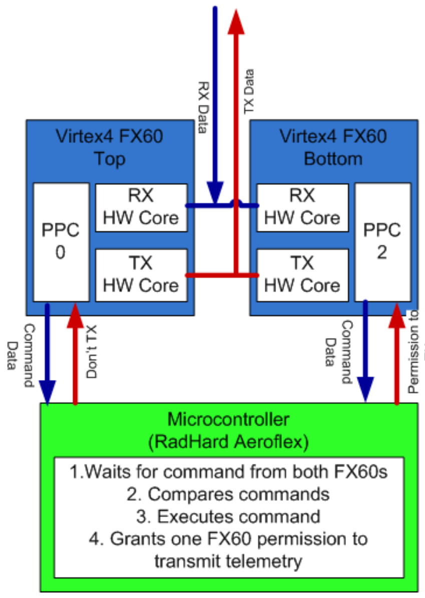


Figure 16 - MISSE-7 SpaceCube Processor Design

New FPGA cores were developed to support the MISSE-7 experiment. The hardware acceleration Sobel Edge core from RNS was slightly modified to support the Lander application. The CIB UART core was designed to handle all CIB communication and significantly assist software in robust packet handling. The spare PowerPCs in each FPGA were utilized to run continuous tasks. An identical task was run in a MicroBlaze processor. Each processing string had command and telemetry capability to the main C&DH application running in the primary PowerPC. Both of these secondary processing systems were clocked by separate redundant DCM structures. Each redundant DCM consists of two DCMs that periodically (apprx. 1 minute) switch control of driving the clock net. Additional logic is in place to detect a DCM string failure, switch over to the redundant DCM, and reset the failed DCM string. A block diagram of the FPGA designs is shown in Figure 17.

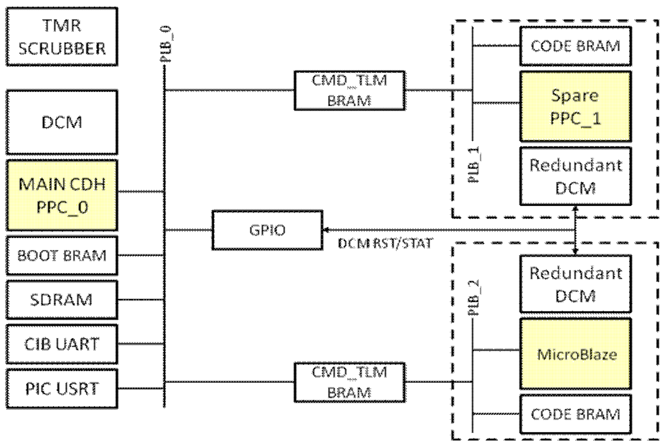


Figure 17 - MISSE-7 Xilinx Embedded System Design

The radiation-tolerant SDRAM memory modules on the processor card are still prone to SEUs, and can require additional Error Detection and Correction (EDAC) techniques depending on the radiation environment of the mission. The MISSE-7 experiment provided an opportunity to test memory EDAC on the SpaceCube v1.0 system. Several SDRAM schemes were evaluated and eventually the (16,8) quasi-cyclic (QC) code described in [21] was chosen for the flight FPGA. The (16,8) code corrects all double errors and can detect all triple-adjacent errors in the data. The EDAC technique was incorporated into a standard SDRAM memory controller IP core. The encoding and decoding logic was added to the combinatorial logic path of the data in the memory controller, so that it would not require additional clock cycles. The additional logic delay is sufficiently small that it does not reduce the data rate of the SDRAM. In the SpaceCube v1.0, there are two 16-bit wide, 256MB SDRAM modules attached to each Xilinx FPGA. For the MISSE-7 experiment, the pair of modules was combined to form a single 32-bit wide interface and the (16,8) scheme was employed in each device. This yields an effective data width of 16-bits with the remaining bits devoted to the parity bits. One limitation of the memory controller is that it only detects and corrects errors at a memory address when it is accessed.

It corrects the data before it is presented to the PLB, but does not automatically write the corrected value back to memory. This means that over a long duration, multiple upsets could accumulate. However, when an error is detected in the data, an interrupt is sent to the PowerPC and the memory controller fills a FIFO with the memory address where the corrupted data is. This allows the processor to scrub the memory during idle periods by performing a read operation at the addresses buffered in the FIFO and then writing the data back.

FPGA and Software On-Orbit Reprogamability—Another driving requirement was to support the ability of ground operators to upload new FPGA configuration files, PowerPC software and SpaceRISC software files, then execute a command sequence to reprogram the system. Due to the extremely low communication bandwidth capability to MISSE-7 and the polling schedule of the CIB, this is an extremely tedious and time-intensive process. All files are first compressed on the ground using the GZIP utility.

New software and FPGA configuration files are uplinked to the SpaceCube using ground commands that write the new files to the SpaceCubeȏs onboard Flash memory in 512 byte chunks. A flash write of 512 bytes is comprised of a series of six ground commands. The SpaceCube C&DH app strips out and buffers the data in each of the six commands in SDRAM, using a CRC in the sixth command of the series to validate the 512 byte chunk of data. Once the CRC is validated, it sends a Flash write command to the SpaceRISC containing the chuck of data. It then performs an automatic Flash read command and sends the data to ground in the next telemetry packet so that it can be verified that the flash write executed properly. If one of the six commands is received out of order, the C&DH resets its buffer, reports the error in telemetry and waits for a new series of packets to arrive.

Due to the hundreds to thousands of commands needed to uplink new configuration files, an automatic command generation feature was built into the MISSE-7 ground software application. The ISS Experiment Control Center (IECC) was developed in house at GSFC and is based on the GSFC Instrument remote control framework [8]. To uplink a file to the SpaceCube the IECC user provides a file name, a SpaceCube flash address, and a file offset if the user is in the middle of a file uplink.

The IECC parses the file, calculating the size of the file and the number of packets needed to transfer the file. It then generates a flash block erase command to clear the flash location that will be written to. The IECC then starts generating the six command series needed to uplink one 512 chunk at a time. The IECC only transmits the next command in the series once it receives telemetry that the last was received and validated by the C&DH app. It will automatically retransmit packets to account for dropped packets and Loss of Signal to ISS. When the series of six packets completes, it waits for, then validates against the readback telemetry of the 512 chunk. If the readback is invalid it will log the issue and pause the process for debugging by the user. If the readback is valid it generates the next series of six commands. It continuous this process until the whole file is uplinked.

Once the support files and new compressed configuration and/or software files are uplinked, the user executes a series of commands to reprogram the FPGA and embedded processors.

The support files are encoded with the physical flash address location of the new FPGA and embedded processor configuration files along with other parameters needed to perform a reconfiguration.

- **RT_Config:** contains addresses for new Flash Image Table, new top and bottom FPGA files
- **Flash_Image_Table**: points to new boot scripts and new PowerPC SW
- **PPC_SW_Boot_Script**: points to a slot in the new flash image table which points to the new PPC SW
- **Compressed Primary PPC SW**: contains updated main app including the new C&DH. Its uncompressed automatically during bootup
- **Compressed experiment PPC SW**: contains new experiment source code that operates on the second embedded PPC which is designated for experiments. (This code is stored in flash but loaded into BRAM by the main PPC app)
- **Compressed FPGA configuration**: a file containing a FPGA bitsream that works on both the bottom and top FPGA.

To decompress the new compressed FPGA configuration file or other compressed support file, a special ground command is used. The user populates the flash decompress command with a source and destination flash address. When the C&DH app receives the command from the ground it calculates the file size, and verifies that the compress file is not corrupt, then writes the uncompress data to the new flash location. Last, it reports the outcome of the decompression in its telemetry.

Below is the series of ground commands that are executed to initiate a SpaceCube reconfiguration.

1. Decompress compressed FPGA configuration file and save in a new flash memory location.

2. Read back first and last 512 bytes for crude verification of successful decompression and flash write.

3. Command SpaceRISC application to utilize new RT Config file located at a given flash address.

4. Command SpaceRISC to Reload RT Config file

5. Command SpaceRISC application to reconfigure FPGAs

The SpaceRISC app will use the new FPGA files that are pointed to by the new RT_Config file. Once the FPGA is configured, the embedded processor boot loader will ask the SpaceRISC for the flash location for its software. The SpaceRISC will provide the new flash addresses provided by new Flash image table. This will result in the new PowerPC software files to be loaded.

To accommodate and mitigate anomalies in the reconfiguration process, only one FPGA per SpaceCube system is reprogrammed at a time. If an anomaly occurs preventing one FPGA from being reconfigured, the C&DH app will continue to operate nominally on the other FPGA allowing for the reconfiguration to be reverted to the óGoldô configuration. A power cycle of the SpaceCube also results in a reversion back to all óGoldô configurations.

Operations—Primary MISSE-7 SpaceCube payload operations are performed at GSFC. Operations are conducted through MSFCôs Huntsville Operations Support Center (HOSC), which manages the telemetry and command links to ISS attached payloads.

Operations are conducted using two main application suites: the HOSCôs Telescience Resource Kit (TREK) and GSFCôs IECC. TReK serves as gateway to ISSôs payload data stream and provides telemetry and command streams from GSFC to the HOSC. The IECC sits on tops of TReK as an advanced secondary payload telemetry and command processor. The IECC is built on GSFCôs Instrument Remote

Control (IRC) framework. The IECC has the following features:

- User generated custom displays via XML
- Client/Server capabilities supports end users
- Interactive and automated commanding
- Real-time event detection and geolocation
- Interactive event mapping and IRC plug-in
- scripting for real-time complex telemetry processing

The IECC displays real-time Health and status telemetry, plotting critical temperatures. It has a feature that monitors SEU telemetry which autonomously time stamps and geotags the SEU events.

The SpaceCube on the MISSE-7 payload is shown in Figure 18.

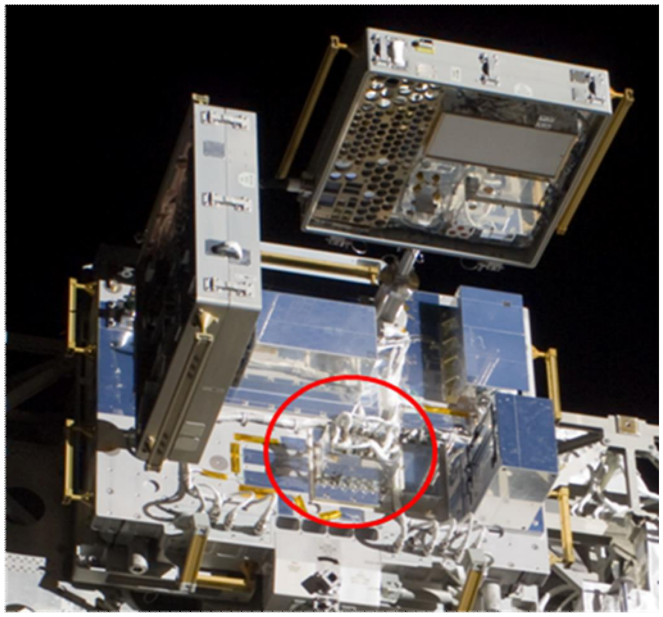


Figure 18 - SpaceCube/MISSE-7 Installation on ISS

Results—The MISSE-7 SpaceCube payload has been continuously operating for four years at the time of this paperôs submission.

We have had only one anomaly on 12/9/12 at 4:59pm EST that required power cycling the payload. In this instance, one of the two SpaceCube experiments appeared to have stopped sending data and was not recoverable through reset and reconfigure commands. Nominal operations were resumed after the power cycle. There is not enough data to determine if the CIB was involved or if it was solely a SpaceCube problem. No further issues have been observed.

We have not experienced a processor reset as a result of a watchdog timeout. Our data shows that the PowerPC processors have been up and running for more than 99.999% of the time. Further data analysis is needed to confirm 100%.

The overall average SEU rate that we have collected on the four FPGAs is 0.09 SEU/Day/FPGA. A 10-month sample of where SEUs have occurred are geotagged and depicted in Figure 19. Each color represents one of the four FPGAs.

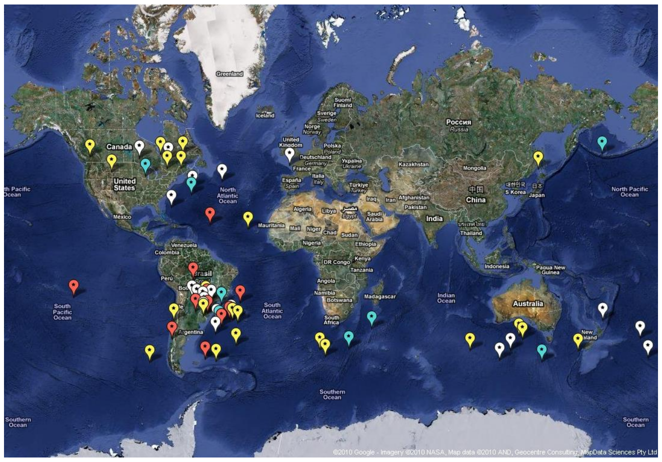


Figure 19 - MISSE-7 SpaceCube SEU Map

We have noticed a few scrubber runaway occurrences. The SEU count for a single FPGA starts incrementing at a fast rate. It will last for a period of hours to days. We have not noticed any adverse effects to the underlying applications running.

Updated SEE results will be presented at the conference.

The MISSE7 SpaceCube was an essential part to making MISSE7 a success. The SpaceCube was considered the most reliable experiment and thus was utilized as an indicator to the health of the misse7 payload. During integration testing the SpaceCube also uncovered an anomaly with the CIB that helped characterize operational performance.

The MISSE7 SpaceCube system continues to be a successful and valuable payload because it is a prime showcase of the reliability, flexibility and high-performance of SpaceCube technology. The SpaceCube team was involved in the full life cycle of this payload, from requirement writing, to hardware design, hardware assembly, software development, environmental testing, integration, and post launch operations. This payload has provided significant lessons learned that have laid a strong foundation for all work that followed it.

Development Effort—The development cycle for the MISSE-7 box was drastically less than that of the RNS box. This is mainly due to minimal NRE required to build the hardware. The only new piece of hardware was the communication and power adapter slice. This phase of the development took 9 months and required the equivalent of approximately 3.5 people/year, or 3 man-years. The application development phase took 1 year and required the equivalent of approximately 5 people/year, or 5 man-years. After payload delivery (2/2009), one FPGA and three software updates were made to fix issues that were found during payload testing and to add enhanced features. This system was on a very short delivery schedule in order to meet payload integration milestones. Being that SpaceCube is reconfigurable, it allowed us to meet the delivery deadline by delivering the system with all essential functions, but to continue development for later upgrades.

C. DPP/Argon

The Satellite Servicing Capabilities Office (SSCO) at GSFC began efforts in 2009 to improve the agencyȏs capability to robotically service satellites in space. Two simultaneous flight projects were spawned from this effort (1) Robotic Refueling Mission (RRM) and (2) Dextre Pointing Package (DPP). RRM was launched to ISS in 2011 and has been completely successful in demonstrating the capability to tele-operate tools in space to do robotic servicing tasks such as gas fitting removal, refueling, screw removal, and thermal blanket manipulation [20]. DPP was a more advanced follow-on mission to RRM that would demonstrate passive and active relative navigation sensing by autonomously controlling the ISS Dextre robot to point to and track vehicles within proximity [17]. Due to budget constraints, the SSCO had to downgrade DPP to a AR&D ground demonstration called Argon. Argon integrates essential AR&D components and unique algorithms into a system that autonomously images, visually captures and tracks dynamic and static target [19].

The Argon system show in Figure 21 consists of two RNS cameras, a star tracker, a Visual Navigation System (VNS), an Inertial Measurement Unit (IMU), an Infrared camera, a wireless Ethernet module, Power Control Unit (PCU), a suite of situational awareness cameras, and the SpaceCube as the payload avionics and onboard processor. 1553, Ethernet, and wireless 802.11 Ethernet are the main communication channels. The main objectives were to demonstrate a robotic AR&D system that couples the functionality of a collection of cameras, sensors, computers, algorithms, and avionics to independently track an uncontrolled target at different ranges. Once the AR&D system has locked onto the target, Argon will safely guide the robot through precise rendezvous and docking maneuvers [19].

SpaceCube Hardware Changes—A few modifications were necessary to support the increased requirements of the Argon system. A new Video Compression Module (VCM) slice for the SpaceCube was built to handle the new interface requirements of the situational awareness cameras (NTSC). The VCM is Xilinx-based, which was a huge upgrade in reconfigurability and functional potential compared to the Actel-based VIM slice on RNS. New DCC boards were made to fix timing parameters within the Ethernet circuit to guarantee functionality with ELC. Finally, both processor card front-panel connector configurations were changed to increase the amount of I/O available. This required new processor housings.

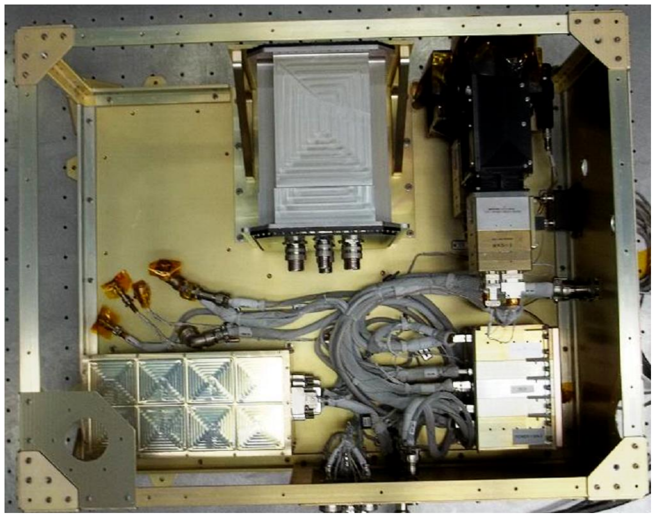


Figure 20 - Argon Assembly, SpaceCube Lower Right

FPGA Design and Application Description—The SpaceCube applications heavily leveraged the RNS work as a starting point. The FPGA designs were adapted to accommodate the added Argon requirements. The AGC and GNFIR algorithms were improved from RNS. Two additional AR&D algorithms with supporting FPGA hardware accelerator cores were added, Goddard FlashPose and JSC Cooperative 3D Pose. The SpaceCube 1553 and Ethernet interfaces also required development and test to obtain reliable operation.

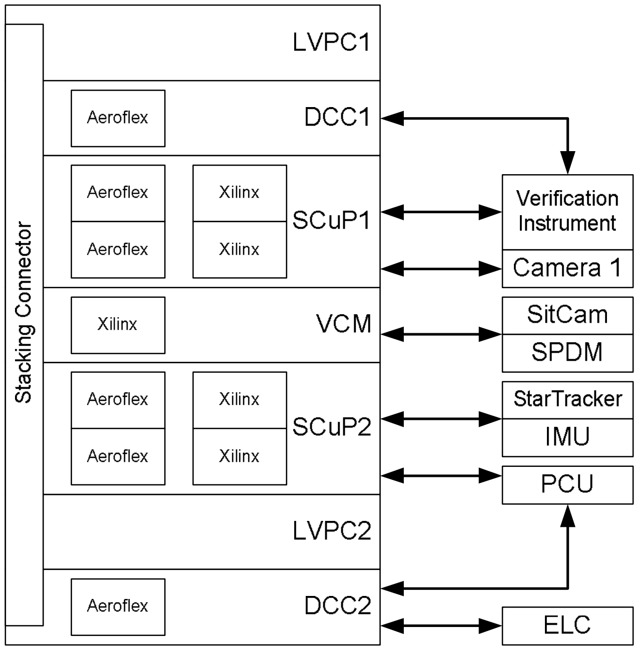


Figure 21 - Argon SpaceCube Diagram

Argon required a significant amount more processing power than on RNS. As a result, all 8 PowerPCs were utilized running the SpaceCube Linux OS. A custom software bus was implemented that utilized the LVDM transceivers to communicate between cards via the internal stacking connector. During application development, Ethernet was used for quickly loading new FPGA configuration and software files to flash storage via the internal PowerPC bus architecture. For the four FPGAs, slice utilization ranged from 80-95% and BRAM utilization ranged from 60-90%. This configuration of the SpaceCube requires 43W of power.

Testing—Static and dynamic system testing with Argon occurred at GSFCõs Satellite Servicing Center, the Naval Research Laboratory (NRL), and at Lockheedõs Space Systems facility in Denver. Argon was successful in demonstrating its stated objectives with a flight-ready system. Figure 22 shows a picture of open- and closed-loop system testing of Argon. The Argon package is attached to the blue Fanuc robot arm on the far left. Argon tracks the motion of a non-cooperative, tumbling satellite, which is the gold mockup mounted on a motion-based Rotopod platform on the far right [18-19].

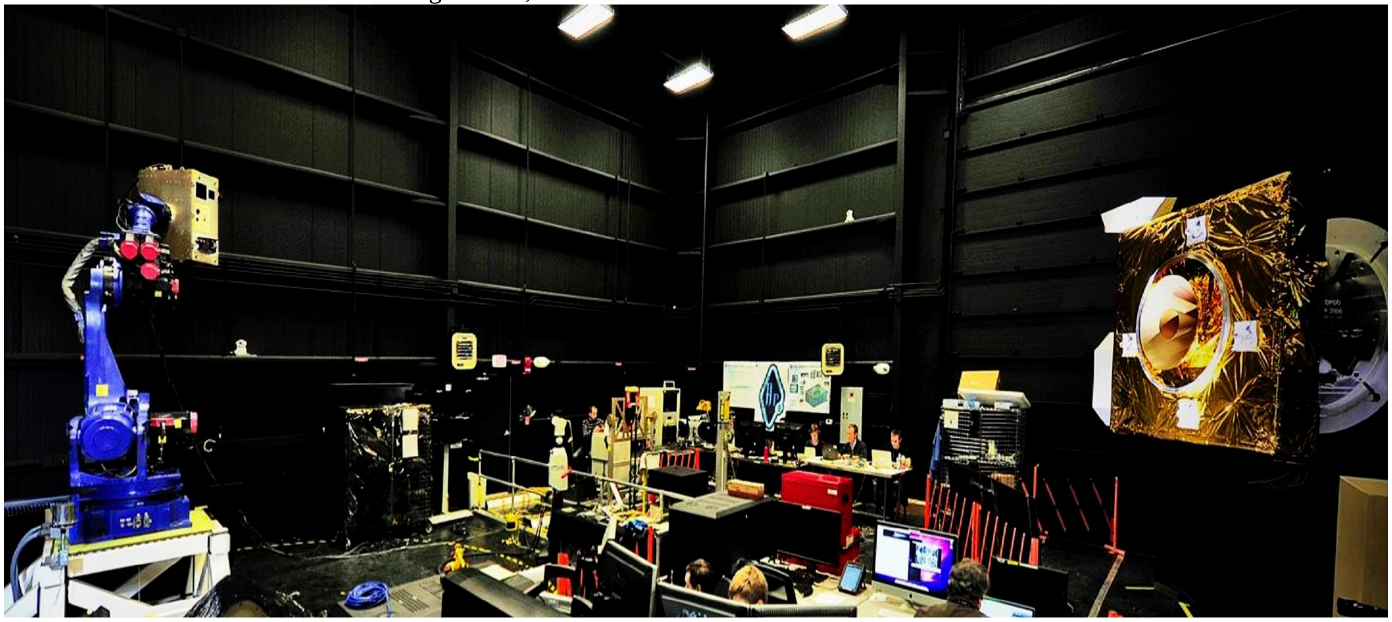


Figure 22 - Argon Testing at the GSFC Robotic Satellite Servicing Center

Development Effort—The hardware required some NRE to build new DCCs, a new VCM card, and slightly modify the processor card connectors. This phase of the development was programmatically slow, taking 18 months and required the equivalent of approximately 4 people/year, or 6 man-years. The application development phase was more involved to accommodate the additional interface requirements and demonstration objectives. It took 2 years and required the equivalent of approximately 9 people/year, or 18 man-years. The Argon system development was very dynamic as the internal architecture was in constant flux. The SpaceCube system was heavily leveraged for its ability to adapt to the changing requirements by reconfiguration of the FPGAs and software.

D. SpaceCube CIB on STP-H4

The DoD Space Test Program (STP), managed by the Air Force, was responsible for the payload processing of MISSE-7. They were impressed by the reliability and capability of the MISSE-7 SpaceCube during system integration testing and by its on-orbit performance. STP requested that Goddard deliver a SpaceCube v1.0 system to replace the legacy CIB system from MISSE-7 for a new ISS payload called STP Houston-4, or STP-H4. The SpaceCube CIB (SC_CIB) gives the STP-H4 payload the ability to offer experiments higher bandwidth data connections since SpaceCube supports an Ethernet interface compatible with the ISS High Rate Data Link (HRDL) via ELC avionics. STP-H4 is installed on ELC-1. The STP-H4 payload pallet is shown in Figure 23. The SpaceCube CIB is seen on the bottom right.

For STP-H4, SpaceCube CIB supports six experiments via RS422 interfaces. One of the experiments is called ISS SpaceCube Experiment 2.0 (ISE 2.0), which is a GSFC experimental payload based on an Engineering Model of the SpaceCube v2.0 processing system [23].

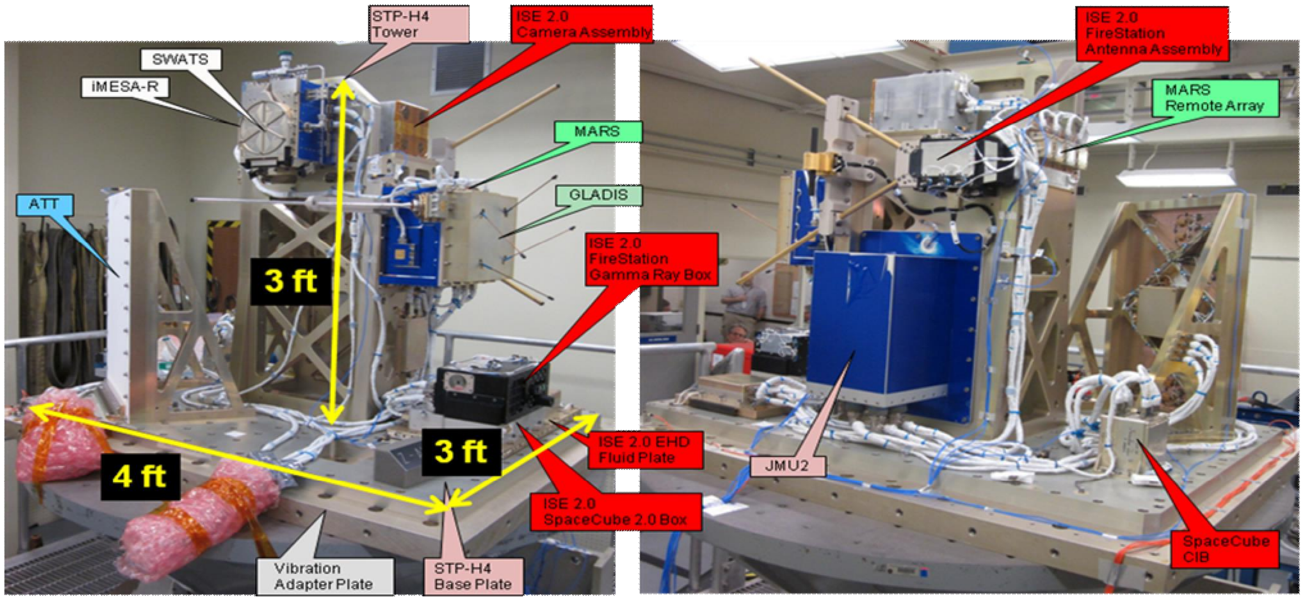


Figure 23 - STP-H4 System Integration

SpaceCube Hardware Description—The SpaceCube CIB is a base system, which as described in section 3 is one processor slice and one power slice [8]. The hardware used for the SC_CIB is a true reflight of one of the processor and LVPC cards from the RNS SpaceCube flight box that flew on Shuttle Atlantis. The DCC board was taken from the flight box developed for the DPP/Argon campaign. A new DCC board was needed to support the Ethernet interface required on STP-H4. The SpaceCube CIB draws 15W of power.

FPGA/Software Application Description—The main objective of the SC_CIB application is to provide a C&DH application between ELC and the payload experiments. A custom C&DH application for the PowerPC was developed for STP-H4. This application validates and forwards commands to the appropriate payload. The C&DH application schedules high rate telemetry (HRT) and low rate telemetry (LRT) requests from all payloads in addition to the CIB itself. The main interface is 1553, which is used for commanding, LRT, and health and status data. The SC_CIB collects health and safety data every second from all attached payloads and its own internal registers such as temperatures, voltages, command and telemetry packet counter, etc., to aid in the operations of the payload. The HRT is sent to the Ethernet interface which operates at a maximum theoretical bandwidth of 10Mbps. The high level interfaces of the SpaceCube CIB are depicted in Figure 24.

The SC_CIB FPGA design leveraged heritage cores and overall embedded architecture from prior missions, which was crucial in allowing for a fast application development cycle required to meeting the ambitious delivery schedule. The interrupt controller, USART, and scrubber cores are from RNS. The SDRAM DECTED EDAC core is from MISSE-7. The 1553, Ethernet MAC, and Ethernet PHY cores are from Argon. Likewise for software, design heritage was a key component in signing up for the fast delivery schedule. The Linux OS framework from RNS was used with all supporting FPGA core drivers. The SpaceRISC updates from MISSE-7 were incorporated to enable on-orbit reconfiguration of the FPGA. The C&DH application incorporates the flash file support and compression/decompression software from MISSE-7 that is also required to support on-orbit FPGA reconfiguration and software updates. The 1553 and Ethernet drivers were used from Argon.

Two new FPGA cores with supporting software drivers were developed to meet the CIB requirements. The TimeCore keeps an internal system time that is synchronized with the ISS broadcast time at a rate of 1Hz. This timestamp is included in all data packets sent to the attached payloads. The Time core is accurate to 1 byte of fine time, which is approximately 4ms.

The second core that was developed for CIB is the Payload Interface Core that is used to communicate with each attached payload via RS422. It validates incoming packets, searches for the sync header, validates header fields, and checks for a valid CRC. It strips out payload data and presents packet statistics to the C&DH software via a series of flags. It also generates all packets transmitted to the payloads. The software writes the desired packet type to a register and if itǿs a command it puts the command payload into a FIFO. The core then generates the packet header, fills in the payload data from the FIFO, and appends the calculated CRC. The core also manages payload response timeouts, by setting timers after packet transmissions and notifies software if the timer expired before a valid response was received. This core utilizes the TimeCore to timestamp all the packets sent to the payloads. It latches the time as it creates the packet to reducing latency to only the packet transmit time. At a high level it abstracts the payload interface to the software as a series of flags and payload data in FIFOs. This reduces the load on the software, allowing the system to quickly collect and transmit the data to the ISS.

Only one FPGA and one PowerPC were used to implement the CIB requirements. The FPGA design utilized approximately 40% logic and 30% BRAM resources. To avoid the accumulation of SEUs that could cause potential issues, a design that only contains the configuration scrubber is implemented on the second FPGA.

CIB Testing and Integration—The SpaceCube CIB went through environmental testing at GSFC prior to delivery to STP-H4 in Houston, TX. GSFC continued to support the system level tests and integration. A 1553 Remote Terminal address bug was uncovered during 1553 validation testing. The software patch to fix the bug was tested at GSFC and the SpaceCube CIB in Houston reprogrammed within 48 hours of discovering the issue. A second software patch was later performed to improve overall functionality as a result of ongoing testing at GSFC.

Following flawless system integration in Houston, the payload was sent to KSC. A risk reduction payload test was performed that included validating communication with the ELC Ethernet interface. After correcting a minor issue in the harness, the SpaceCube CIB successfully streamed 1.2 GB of data at an effective rate of 1.5Mbps. Environmental testing on the system and final end-to-end tests occurred prior to shipment to Japan for launch vehicle integration.

The FPGA design was locked after environmental testing at GSFC and never required an update post-delivery. The option to reconfigure the FPGA on-orbit exists if necessary.

Operations—The STP-H4 payload was launched to ISS on the JAXA HTV-4 vehicle in August 2013. The payload was activated shortly after arrival. The SpaceCube CIBôs telemetry is being monitored with the IECC and it has been operating nominally. All temperatures, voltages, and statuses are as expected. All attached payloadôs telemetry and commands are being transmitted without error.

The GSFC IECC has the capability to command and monitor the STP-H4 payload. The SC_CIB has successfully been sent commands to reset status and its internal counters.

SEE results will be compared to those of the MISSE-7 SpaceCube, and presented at the conference.

Development Effort—The agreement with STP-H4 put the SpaceCube CIB on a strict 12 month delivery schedule. The hardware did not require any NRE. Thus, the hardware build phase of the development was fast. It only took 11 months to build, test, and deliver the hardware. This phase required the equivalent of approximately 3 people/year, which is roughly 3 man-years. The application development phase required more people to implement the CIB-specific FPGA and software requirements. It took 12 months and required the equivalent of approximately 5 people/year, or 5 man-years. After delivery, the STP-H4 system integration required 2 people for 6 months, which increases the total application effort to 6 man-years. The hardware reuse, FPGA/software design heritage, and reconfigurable options of the SpaceCube allowed us to confidently deliver a product within the aggressive schedule requirement. The reconfigurability of the system was utilized after delivery to fix issues found during payload integration.

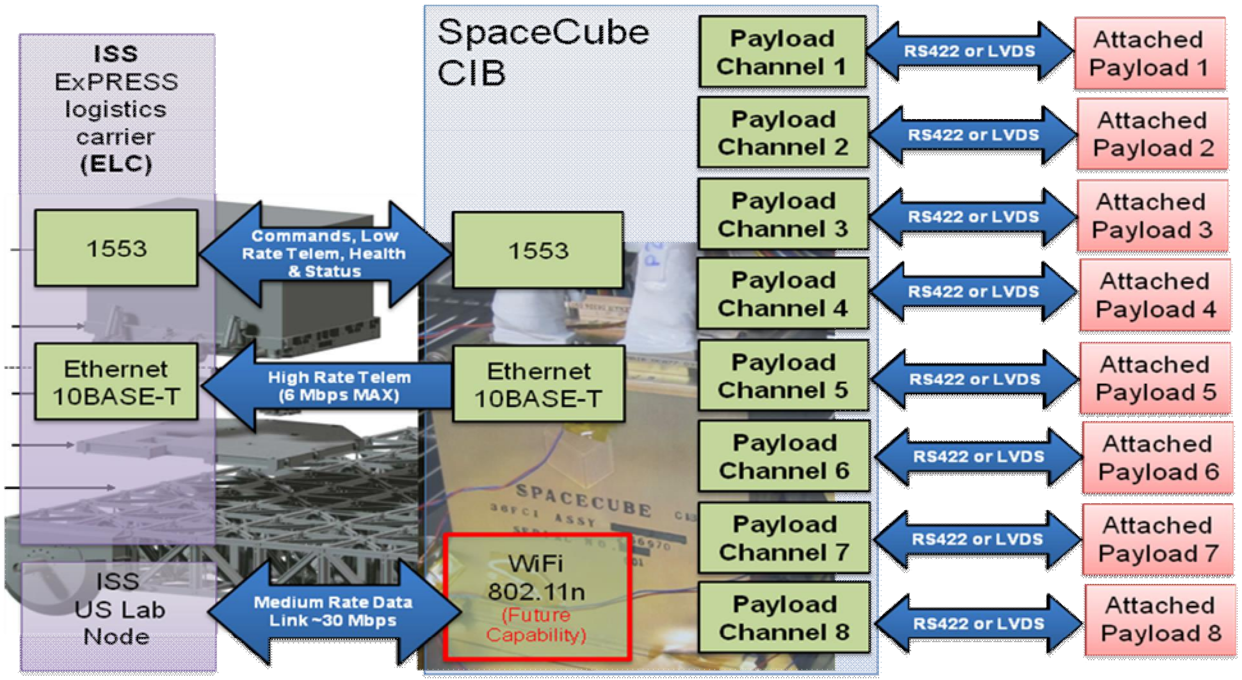


Figure 24 - SpaceCube CIB System Diagram

5. CONCLUSIONS

SpaceCube fits the need for a hybrid computing architecture for space. We have demonstrated reliable use in three separate missions including over four years of operation on the MISSE-7 payload. The computing power of the SpaceCube system provides at least a 10x performance increase over traditional space processors. We have shown how we have solved extreme data-intensive and computation-intensive applications within RNS and Argon by leveraging a multi-processing platform coupled with reconfigurable FPGAs. Traditional space processing systems cannot handle these advanced applications. The SpaceCube hybrid processing system enables break-through mission objectives such as AR&D and robotic servicing.

In addition, the SpaceCube is both reconfigurable and modular. We have shown how we have used these traits to quickly adapt to new missions and changing requirements. Each mission, aside from the SC_CIB, required a mission unique I/O card to meet requirements.

On the MISSE-7 SpaceCube, we have proven the ability to reconfigure the system in space flight with new FPGA and software design files sent from ground. The flexibility of SpaceCube allowed for an ad-hoc collaborative effort to be utilized in developing the new versions of software and FPGA designs that were used to reprogram it in space.

Within this paper we have also highlighted the development effort to build each of the systems in Section 4. This data is summarized in Figure 25 and Figure 26. The hardware NRE, FPGA NRE, and software NRE are significantly reduced after the RNS mission. Each of the follow-on missions only required engineering to build and test copies of the hardware, develop mission unique I/O cards, and integrate the new application requirements in FPGA and software. This reduction in NRE has a great benefit to program cost and schedule. The reuse and application of SpaceCube to different mission profiles is only possible due to its reconfigurability and modularity. As shown for the SC_CIB mission, schedule risk was reduced due to heritage design, hardware reuse, and the reconfigurable FPGAs and software features of the SpaceCube. These combined features are what enabled our confidence in delivering a working system within 12 months.

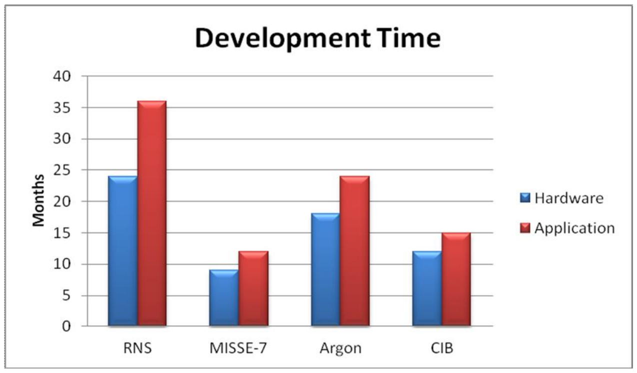


Figure 25 - Development Duration per Mission

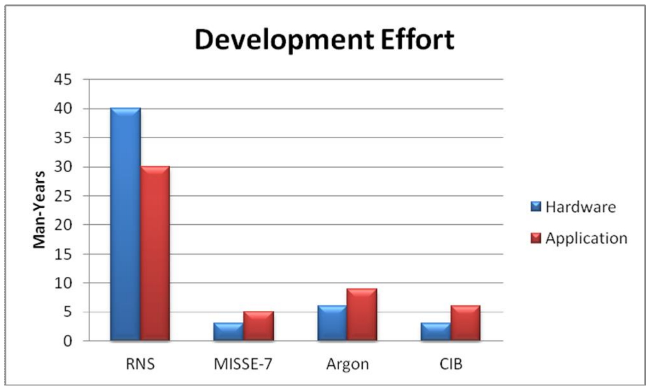


Figure 26 - Total Development Effort per Mission

The reduction of initial design NRE cost, the flexibility of the hybrid architecture, and the inherent low power and weight of the SpaceCube system is what makes the SpaceCube attractive to missions requiring an advanced avionics package.

6. FUTURE WORK

GSFC is currently supporting two new programs that will use a SpaceCube v1.0 system for on-board payload processing. GSFC is delivering another SpaceCube CIB for STP-H5, which is a follow-on project to STP-H4. The hardware will be identical to STP-H4, but will require some software modifications to support new experiments, including file transfer. Also, SSCO will use a SpaceCube v1.0 system to control the third phase of RRM. The RRM-3 SpaceCube will require an I/O card to handle analog monitoring and control of the payload systems, and will also require an added Ethernet interface to communicate with the ISS wireless 802.11 network.

REFERENCES

[1] T. Flatley, ñSpaceCube: A Family of Reconfigurable Hybrid On-Board Science Data Processors,ò presented at the NASA/ESA Conference on Adaptive Hardware and Systems, Nuremberg, Germany, June 2012.

[2] ñSpace Communications and Navigation.ò *NASA Goddard Space Flight Center FY 2006 Internal Research and Development Program, R&D Achievements*.ò [On-line]. pp. 8. Available: http://gsfctechnology.gsfc.nasa.gov/2006_AR_V6_FINAL_low.pdf [Aug. 8, 2013].

[3] ñSpaceCube.ò *Goddard Tech Transfer News*. Spring 2009. [On-line]. 7(1), pp. 3. http://techtransfer.gsfc.nasa.gov/newsletter/springHST_09.htm [Aug. 8, 2013].

[4] B. Naasz, J. Van Eepoel, S. Queen, C. Southward, J. Hannah, ñFlight Results of the HST SM4 Relative Navigation Sensor System,ò AAS Guidance and Control Conference, No. AAS 10-086, Breckenridge, CO, 2010.

[5] D. Espinosa, A. Geist, D. Petrick, T. Flatley, J. Hosler, G. Crum, M. Buenfil, ñRadiation-Hardened Processing System,ò U.S. Patent 8,484,590, issued July 9, 2013.

[6] D. Petrick, ñNASA Shoots SpaceCube Technology into Orbit,ò Xilinx Xcell Journal Customer Innovation Issue, pg. 27, 2010.

[7] ñ4 Row Uninterrupted Receptacle, Stacking.ò HMM Series Data Sheet. Internet: http://www.iehcorp.com/products/1/, [Aug. 8, 2013].

[8] D. Espinosa, J. Hosler ñGoddard Space Flight Center Attached Payload System,ò presented at the Military and Aerospace Programmable Logics Devices Conference, 2013.

[9] D. Petrick, ñApplication of SpaceCube in a Space Flight System,ò presented at the Military and Aerospace Programmable Logics Devices Conference, 2009.

[10] M. Berg, C. Poivey, D. Petrick, and K. LaBel *et al.*, ñRisk Reduction for Use of Complex Devices in Space Projects,ò *IEEE Transactions on Nuclear Science,* vol. 54, no. 6, pp. 2137-2140, Dec. 2007.

[11] C. Poivey, M. Berg, M Friendlich, H. Kim, D. Petrick, S. Stansberry, K. LaBel, ñSingle Event Effects Response of Embedded PowerPCs in a Xilinx Virtex-4 FPGA for a Space Application,ò European Conference on RADECS, Sept. 2007.

[12] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, and K. LaBel *et al.*, ñEffectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis,ò *IEEE Transactions on Nuclear Science,* vol. 55, no. 4, pp. 2259-2266, Aug. 2008.

[13] H. Quinn, P. Graham, *et al.*, ñOn-Orbit Results for the Xilinx Virtex-4 FPGA,ò Radiation Effects Data Workshop, IEEE, Tucson, AZ, 2012.

[14] G. Allen, G. Swift, and C. Carmichael, ñVirtex-4VQ static SEU characterization summary,ò Xilinx Radiation Test Consortium, Tech. Rep. 1, 2008.

[15] G. Allen, ñVirtex-4VQ Dynamic and Mitigated Single Event Upset Characterization Summary,ò Xilinx Radiation Test Consortium, JPL Publication 09-4 01/09, 2009.

[16] ñMaterials International Space Station Experiment ï 7.ò Internet: http://www.nasa.gov/mission_pages/station/research/experiments/653.html, Apr. 26, 2013 [Aug. 8, 2013].

[17] B. Naasz, M. Strube, J. Van Eepeol, B. Barbee, K. Getzandanner, ñSatellite Servicingôs Autonomous Rendezvous and Docking Testbed on the International Space Station,ò AAS Guidance and Control Conference, No. AAS 11-072, Breckenridge, CO, 2011.

[18] B. Naasz and M. Moreau, ñAutonomous RPOD Challenges for the Coming Decade,ò AAS Guidance and Control Conference, No. AAS 12-065, Breckenridge, CO, 2012.

[19] ñArgon,ò [On-line]. http://ssco.gsfc.nasa.gov/argon.html [Aug. 22, 2013].

[20] ñRobotic Refueling Mission,ò [On-line]. http://ssco.gsfc.nasa.gov/robotic_refueling_mission.html [Aug. 22, 2013].

[21] T. Gulliver and V. Bhargava, ñA Systematic (16,8) Code for Correcting Double Errors and Detecting Triple-Adjacent Errors,ò *IEEE Transactions on Computers*, vol. 42, no. 1, pp. 109-112, 1993.

[22] P. Jenkins *et al.*, ñMISSE-7: Building a Permanent Environmental Testbed for the International Space Station,ò Proceedings of the 9th International Space Conference Protection of Materials and Structures From Space Environment, Toronto, Canada, 19-23 May 2008.

[23] ñSpace Test Program-Houston 4-ISS SpaceCube Experiment 2.0 (STP-H4-ISE 2.0),ò Internet: http://www.nasa.gov/mission_pages/station/research/experiments/487.html, May 23, 2013 [Aug. 8, 2013].

[24] ñVirtex-4 Family Overviewò, [On-line]. http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf, DS112 (v3.1), [Aug. 30, 2010].

BIOGRAPHY

David Petrick *started his career at NASA in 2000. He has a wide range of experience building Xilinx FPGA-based systems for space flight including FPGA design, radiation mitigation testing, PCB design, reconfigurable system design, and mission operations. He was the lead design engineer on the SpaceCube v1.0 and v2.0 processor cards including embedded systems framework, FPGA core development, and electrical design, responsible engineer for the RNS SpaceCube system build, delivery, and shuttle payload operations, lead engineer for the MISSE-7 and ISE2.0 SpaceCube hardware deliveries, and systems lead on the SpaceCube v2.0 development effort. He is currently the Embedded Processing group leader within the Science Data Processing Branch and SpaceCube Program Technical Development Lead. He has a BSEE from the University of Pittsburgh and a MSEE from the Johns Hopkins University.*

Daniel Espinosa *started his career at NASA in 2003. He serves in the Science Data Processing Branch and is a key member of the SpaceCube Program. He has a BS in Computer Engineering and a MS in Electrical Engineering from the University of Florida. Dan's area of expertise is in Field Programmable Gate Arrays based reconfigurable high-performance embedded space systems. He has extensive experience with the ISS payload data interfaces and ISS payload operations. Dan has various experiences in the full life cycle of space flight including: requirement formulation, application and FPGA development, environmental testing, system integration, GSE design and mission operations. Dan was the lead developer for the SpaceCube MISSE-7 ISS payload and is currently the payload manager and lead operator. Dan is the SpaceCube CIB system lead for the STP-H4 ISS payload and planned STP-H5. Dan is also the avionics lead for the Robotic Refueling Mission phase 3.*

Robin Ripley *started working for NASA as a contractor for Orbital Sciences Corporation in 2005, then joined NASA as a full-time employee in 2008. She has experience working with both Actel and Xilinx FPGAs . She has worked on SpaceCube for various projects including RNS, CIB, DPP, and Restore doing design and testing at the FPGA and board levels, and integration and testing at the box and system levels. Her BSEE is from Virginia Polytechnic Institute and State University and her MSEE is from Johns Hopkins University.*

Gary Crum *received his B.S in Computer Engineering from Michigan State University. Mr. Crum began working with NASA in 2004 as part of a Senior Capstone Design Project which enabled him to secure an internship working on sensor fusion for robotic path planning and object avoidance. Mr. Crum then worked as a NASA contractor for Jackson and Tull where he lead the development work on the SpaceCube v1.0 Aeroflex System on a Chip and then transitioned to a NASA civil servant in 2008. Mr. Crum specializes in both embedded hardware and embedded software and is responsible for creating advanced SoC designs, IP Core's, device drivers, bootloaders and Flight Software. Mr. Crum has played a key role in all of the SpaceCube related missions. Mr. Crum is currently finishing his M.S. in Electrical and Computer Engineering with a concentration in Robotics at John Hopkins University.*

Alessandro Geist *received his B.S. in Computer Engineering from Johns Hopkins University and his M.S. in Electrical Engineering from the University of Maryland, College Park. He has worked at NASA GSFC in the Science Data Processing Branch since 2006. He has had significant experience implementing and accelerating on-board processing algorithms in FPGAs on a variety of missions including DBSAR, URAD, GNFIR/RNS, MISSE-7, and other internal research and development efforts. He has also been extensively involved in flight processor card design including being the lead design engineer of the SpaceCube 1.5 and co-lead design engineer of the SpaceCube Mini processor systems. In addition, he was the FPGA development lead on ISE 2.0 and the avionics lead engineer on the SMART sounding rocket mission. He is currently developing much of the FPGA infrastructure for the SpaceCube 2.0 processor system, and is also the experiment lead for ISEM on STP-H5.*

Tom Flatley *is currently Branch Head of the Science Data Processing Branch at the NASA Goddard Space Flight Center. Prior to this assignment he served as Branch Head and senior researcher in the Science Data Systems Branch (2005-2007), and Chief Technologist and Acting Associate Head of the Microelectronics & Signal Processing Branch and Electrical Systems Branch (2003-2004). From 1998-2002 he served as Chief Technologist and Associate Head of the Ground Systems Hardware Branch, and from 1993-1997 he served as head of the Flight Electrical Systems Section and Flight Component Development Group. Prior to this period he developed numerous flight and ground components and subsystems for various NASA missions, beginning in 1985. Mr. Flatley's current work includes the coordination of embedded science data processing technology development and hardware accelerated science data processing activities, serving as Principal Investigator on multiple flight processing experiments, with the primary goal of developing re-configurable computing technology and hybrid systems for flight and ground science data processing applications. He is also a key member of the GSFC CubeSat/SmallSat technology working group, manages numerous collaborations with government, industry and academic partners, and serves as liaison between* *technology developers and end users in the science community. Mr. Flatley received a 2011 NASA "Exceptional Engineering Achievement Medal" and the 2012 American Astronautical Society "William Randolph Lovelace II Award" for advancing spaceflight and space exploration technology through the development of SpaceCube.*