

Reconfigurable Avionics for Hubble Servicing Missions

E. Cheung¹, W. Clement² and R. Bietry²

¹Jackson and Tull, Seabrook, MD 20771

²Clement Engineering, Arnold, MD 21012-2220

³Orbital Sciences Corporation, Greenbelt, MD 20771

Abstract

Every few years, the Hubble Space Telescope (HST) is serviced by space walking astronauts who install new instruments or replace worn subsystems. Carriers that fit into the Cargo Bay of the Shuttle protect these new components, furnishing power and a stable thermal environment for their ride to space. This paper describes a reconfigurable avionics system that is meant to be the next generation of carrier avionics. The reconfigurability of the new avionics system will be a useful feature as the manifest of hardware has historically changed on short notice in order to react to on-orbit emergencies with HST. This system will be flown on the Fifth Servicing Mission (designated SM-4) to HST in 2005.

I. INTRODUCTION

The HST servicing program is centered on NASA's space shuttle. The shuttle was used to deploy HST, and has been used to service it four times to date. For each of these servicing missions, NASA has had to outfit the shuttle specifically for the tasks involved. This consisted of building Space Support Equipment (SSE) carriers, and interconnecting them to the shuttle's avionics system. Each mission is unique, but can be implemented by utilizing a set of building blocks, similar to a modern erector set. The building blocks include both mechanical structures and avionics boxes. Since the requirements for these missions change each time, the building blocks need to be flexible and re-configurable. With this in mind, we have been building up a modular avionics box called the Enhanced Power Distribution and Switching Unit (EPDSU) over the course of each mission. The EPDSU is a modular chassis, with connectors designed to match the standard interfaces to the shuttle's. The chassis has space for ten modules to implement the specific functions required for each mission. The first EPDSU flew on HST SM-2, and since then, additional units have been replacing other single function avionics boxes on the SSE for SM-2, 3a and 3b. The original chassis was configured with a series of Power Control modules. Each module can be configured internally through a series of patch and fuse plugs to support the specific function required. The first EPDSU, flown on second HST Servicing Mission (SM-2) contained four modules, and was used to supply redundant power, control and monitoring functions for the Second Axial Carrier (SAC), the carrier that brought the NICMOS instrument to HST. Since then, two

additional EPDSUs have been built and flown, replacing all original SSE power controlling boxes. To date, these boxes have been dumb boxes, powering and monitoring specific functions without internal computers.

The EPDSU was originally designed to allow for inclusion of intelligent modules to offload control and monitoring functions from the shuttle's General Purpose Computer (GPC), eliminating the need for 20-year-old avionics boxes currently used for this interface. For HST SM-4, we are upgrading an EPDSU for use on the Super Lightweight Interchangeable Carrier (SLIC) that will be used to carry a new Wide Field Camera (WFC) to HST. The upgrade will implement this intelligent function in the form of redundant Telemetry Modules.

In developing the requirements for the Telemetry Module we analyzed how the avionics were used on each of the previous missions, and what flexibility would be desired to allow the modules to replace the remaining outdated avionics interface boxes used on our other carriers. A key factor in this analysis was the ability to re-configure the avionics without opening the box, and incurring a re-qualification program. This requirement was well understood internally, since we have consistently been required to make configuration changes late in the program to meet the demands of a very large and dynamic program. This flexibility is not well understood by people outside the HST program and unfamiliar with the unique roll and requirements of the SSE.

Based on the requirements, a trade study was performed to determine the overall architecture of the module. The study lead us to a System On a Chip (SOC) implementation, designed as an intelligent peripheral. Since we have low volume requirements, and require reconfiguration, we implemented it in a single Field Programmable Gate Array (FPGA). This included both the application specific glue logic, and the processor based intelligence. This was possible due to the ever increasing size of FPGA's. We chose a Xilinx RAM based FPGA to allow for complete re-configuration of the software, processor and hardware without opening the module, or even removing the module from the EPDSU chassis. The design implementation choice was VHDL, and the design was implemented with commercial VHDL/FPGA tools. This choice reduced the development time and made implementation flexible, but carries a verification cost in understanding possible single point errors and error mitigation. Since HST is in low earth orbit, we needed the module to function in that ionizing radiation environment. The Xilinx FPGAs are available in a radiation-hardened

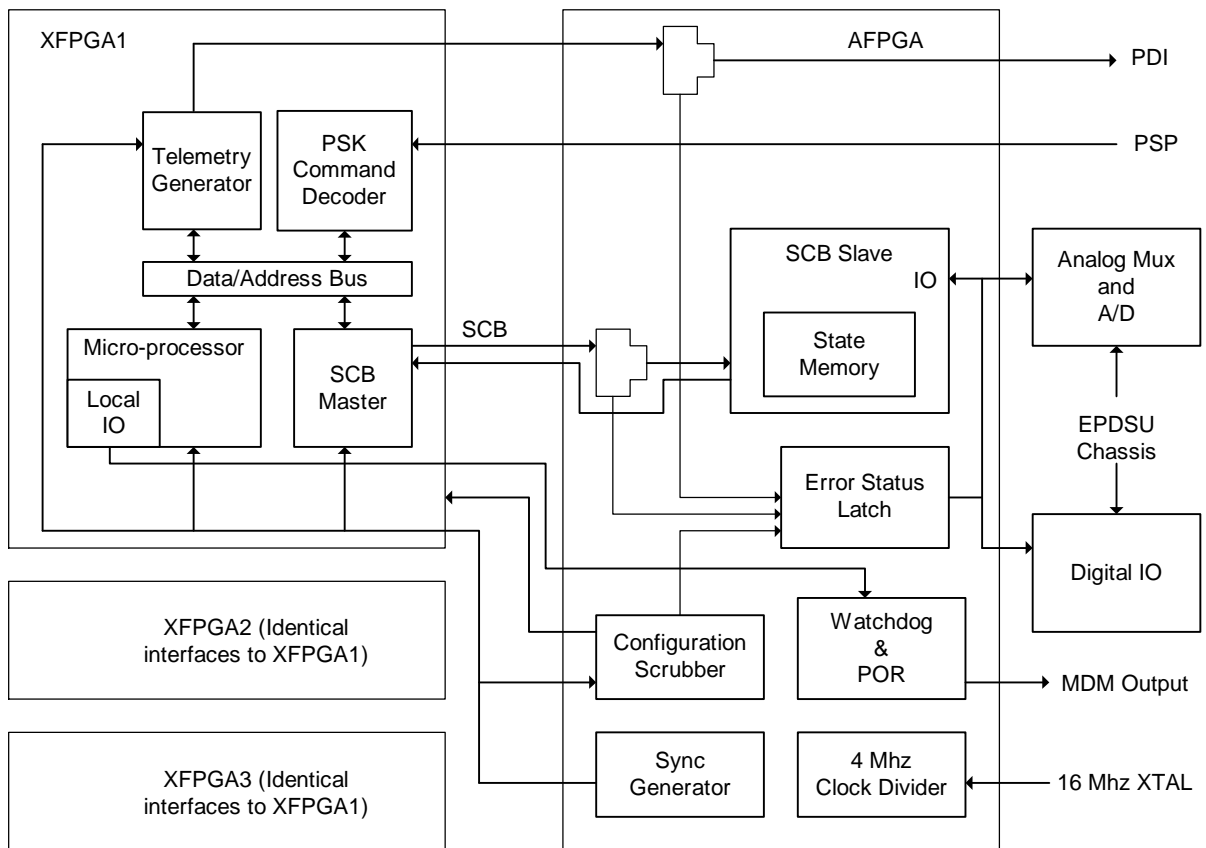


Figure 1. System Diagram of the SLIC Telemetry Module.

version, but they are not hard against Single Event Upsets (SEUs). The FPGAs are subject to SEUs that can change the value of a RAM cell that contains either configuration or state information, causing a malfunction. We mitigated the sensitivity to SEUs in the configuration information by scrubbing the configuration information. We mitigated the sensitivity to errors in each chip by running three identical FPGAs in parallel, and voting on both the configuration and operational data. This approach significantly reduced the need to eliminate errors in any single FPGA, since all three chips operate independently, and are constantly voted, preventing a error from propagating, and correcting the FPGA that has an error as soon as it is detected. The remainder of this paper describes the approach we developed to mitigate that sensitivity, and allow us to utilize re-configurable FPGAs in the Telemetry Module.

II. SYSTEM DESCRIPTION

A. Overall System Architecture

On the highest level, the SLIC Telemetry Module (STM) performs the following functions for the carrier hardware:

- Using the A/D converter and the analog multiplexers, reads analog channels such as voltages, currents and temperatures.
- Reads discrete switch status such as doors and latches.

- Sends telemetry down to the ground via the Space Shuttle.
- Decodes commands from the Space Shuttle or ground crew to control heaters and loads.

These functions are performed by four FPGAs on a logic board – three of which are reprogrammable without opening the STM. The reprogrammable FPGA that was selected is the Virtex XQVR600 from Xilinx. Unfortunately this FPGA is not completely SEU immune. As a result, the manufacturer recommends several techniques to overcome these problems such as Triple Module Redundancy (TMR) [1]. This architecture instantiates three copies of a design, and uses majority-wins type voters for the outputs. We decided not to follow this approach due to our inability to be confident that the effect of a single SEU event would be limited to one copy of the design inside the single FPGA. It seemed possible to us (due to our lack of knowledge of the details on the silicon level) that a single SEU could simultaneously affect more than one of the three ‘strings’, causing the disruption of the operation of the avionics.


Instead of the strict TMR approach, we decided to employ a modified version, using three *separate* reprogrammable FPGA’s of identical design. Denoted as ‘XFPGA’ in Figure 1, they communicate only with a non-reprogrammable (anti-fuse) Actel 54SX72. The latter is denoted ‘AFPGA’, and contains functions that are considered unlikely to change such as the Input/Output (I/O) ports and data storage. Also, it is sufficiently SEU immune to allow its use without redundancy.

All I/O functions performed by the XFGPA occur via serial streams. For example, the command path from the Space Shuttle occurs via the Payload Signal Processor (PSP) interface [2], and the serial telemetry stream to the Shuttle and the ground is via the Payload Data Interleaver (PDI) interface [2]. Similarly, all actual I/O to the analog and discrete I/O subsystem is via the Serial Communications Bus (SCB). The serial streams from the XFGPAs (PDI and SCB) are combined by voters in the AFPGA. In this manner, if one XFGPA fails to operate, its erroneous output will be outvoted and suppressed by the data from the other two functioning XFGPAs. This system provides tolerance to single SEU events.

The voting of the serial stream is feasible because the three XFGPAs are synchronized at all times. All three run off the same 4Mhz clock and remain in phase once synchronized. The synchronization is repeated regularly to overcome effects due to SEUs, as will be explained below.

Each individual XFGPA contains a command processor, a telemetry generator, a Microprocessor and its associated peripherals, and the SCB module mentioned above. Each of these is explained in the following sections.

B. The Voters

There are four voters in the AFPGA, two of which are denoted by the symbol  in Figure 1. They combine the state of three inputs into a single output by determining which logic state is in the majority. The voters are instantiated for the PDI and SCB busses. By using voters, the result appears to be a single XFGPA that is robust to single SEU events.

In addition to the three data lines that control the level of the output, the voters also have two control lines that determine their mode. If both control lines are logic low, normal voting of the three data inputs takes place. In any other state, the voter will connect a particular one of the three data inputs directly to its output. This "Selector Function" is useful for test, debug and verification, but is disabled for flight.

Besides the data output, the voters also issue status information in the form of discrete lines that indicate which (if any) input is in the minority. This allows counters to track errors that occur as a result of SEUs in the system. The mismatch is checked on the rising edge of the system clock. In this manner temporary invalid results due to propagation delay are ignored.

C. The Serial Communications Bus (SCB)

As explained above, the core functions are implemented in the XFGPAs. These functions, such as the Analog to Digital (A/D) converter are very I/O intensive, requiring numerous digital lines. This could have been implemented using parallel busses leading from the XFGPAs to the AFPGA, along with a corresponding number of parallel voters in the latter unit. Instead of this rigid and cumbersome method, a bank of general purpose parallel I/O ports and memory

locations were implemented in the AFPGA, controlled by the SCB module in the XFGPA.

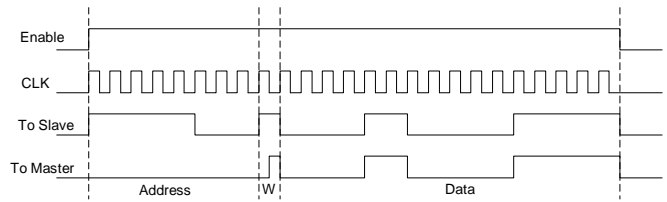


Figure 2. The Serial Communications Bus lines.

The SCB is a simple serial bus using four lines: Enable, Clock (CLK), Data to Slave and Data to Master. Each SCB connects an XFGPA with the AFPGA, using a master in the former and a single slave in the latter. The master SCB in each XFGPA controls all the lines except 'Data to Master', which is controlled by the slave. Voters in the AFPGA combine the lines controlled by the master, so that the slave is connected to only a single set of SCB lines.

The data sent to the slave consists of an 8-bit address and 16-bit data argument along with a control bit for read/write. By convention, addresses below 0x10 (16 decimal) are physical I/O ports. Writing to these locations will set bits on the AFPGA's outputs, while reading from them transfers data from the AFPGA's pins to the XFGPA via the 'Data to Master' line. Locations 0x10 and above refer to storage locations, and are used for holding data. The latter implements a 64x16 bit memory bank that is unlikely to be upset by SEU events, and provides a robust method for the XFGPAs to store data. The clock line runs at half the system clock speed, or 2 MHz. As a result, each SCB message takes 12.5 microseconds to transfer.

During Write operations, the 'Data to Slave' line holds the data being written into the Slave. The 'Data to Master' line has a special function in this case. It acts as a "bent pipe", sending the *voted* data right back to the master. With this mechanism, each XFGPA obtains a voted copy of the data and at the end of the SCB transaction it will know if the result that it intended to write was outvoted, and correct its version if needed.

The approach of using a serially controlled peripheral to implement a bank of general purpose I/O and memory makes future expansion easier. It is not necessary to change the board layout, or to add more voters in the AFPGA. The overhead in time that each transaction requires is not a problem due to the relatively slow speed of the discrete I/O in the STM.

Besides physical I/O locations and the memory bank, the SCB slave also has locations to allow the AFPGA to communicate data to the XFGPAs. These are, for example, memory locations that map to the latches that hold the voter mismatch information. If a voter input line is in the minority, it is judged to have an error, and this occurrence sets a latch in the SCB slave that can be read by the XFGPA. As will be explained later, this information is downlinked to the ground to gather SEU performance data.

D. Microprocessor

The Microprocessor is implemented in the XFPGA as another VHDL module. Software that runs the sequential operations of the STM such as the gathering of analog data and the decoding of commands is implemented in C code. Software tools automatically convert the C code into a lookup table in VHDL, which is synthesized along with the rest of the XFPGA modules.

The Microprocessor uses the same instruction set as the PIC family from Microchip Corporation. As a result, we are able to use a low-cost and proven compiler and Integrated Development Environment software for our development. A drag-and-drop tool converts the compiled HEX file into VHDL code for easy code development.

The Microprocessor controls its peripheral bus (shown as “Data/Address Bus” in Figure 1). This bus allows control of the Command Decoder, SCB Master and other peripherals inside the XFPGA. C code drivers have been written for these peripherals, allowing easy interface using high-level code for the I/O functions.

The box labeled “Local I/O” in Figure 1 is the exception to the rule that each XFPGA performs all I/O serially. The following are implemented in this portion of the system, and explained further in sections below:

- Watch dog stroke. The C code strokes a watchdog inside the AFPGA that verifies all three processors are running.
- Although the VHDL code for each XFPGA is identical, each Microprocessor knows its identity by reading input pins whose logic levels are determined by the board layout. This allows commands to be targeted to a particular Microprocessor, such as the “Suspend” command to simulate the freezing of a Microprocessor due to an SEU event.
- Reset of the Command Decoder. See the section on the “Reset and Synchronization”.

The Microprocessor C code executes in a continuous loop, its execution timing controlled by the PDI Telemetry Generator. Once the processor detects that the Generator has started a new telemetry frame, it starts its normal processing. This includes the decoding of commands, and the sampling of the analog and digital telemetry. The latter data gets populated into the Telemetry Generator’s Dual-Port RAM (more on that below), for downlinking to the ground. This takes only a portion of the time it takes for the Generator to output one frame of telemetry, after which the Microprocessor C code goes into its “idle” mode, waiting for a new frame to start.

E. Processor Watchdog

As explained above, the C code strokes a watchdog in the AFPGA. Upon expiration, the watchdog does not cause a system reset, as that function is already performed by the Synchronization subsystem. Instead, the watchdog produces an output for the Space Shuttle computers to alert that

subsystem of a problem. The notification goes out in the form of a single discrete I/O line to the Shuttle.

The stroking is synchronous to the telemetry frame of the STM, occurring about 4 times per second. If a single Microprocessor stops stroking the watchdog for more than two seconds, that unit is flagged “dead” in the telemetry stream. Note that two XFPGAs are still performing nominally, outvoting the failed unit. If the failure persists for greater than 600 seconds, the Space Shuttle is notified. An astronaut will then shut down the current side of the STM, and transfer to the redundant side. If, on the other hand, more than one XFPGA gets flagged as “dead” at any time, the notification will be issued instantly, bypassing the 600-second delay.

F. PSP Command Decoder

The PSP interface from the Space Shuttle is a Phase Shift Key modulated 2 kbps data stream onto a 16kHz carrier [2]. The input sinusoidal waveform is threshold detected by an RS422 receiver, and connected to the AFPGA for distribution to the three XFPGAs. The most critical aspect of this system is the Phase Locked Loop (PLL), which needs to recover the original 16 kHz signal used to modulate the stream.

After threshold detection, the input Digital PSP Signal is EXOR’ed with the PLL output, and then digitally filtered to obtain the raw data stream. This is then cleaned up with the clock recovery circuit, and shifted into a parallel register. The input data is then searched for the synchronization header “0xEB90” to detect the start of a command sequence. Once that is found, the command bytes are inserted into a FIFO for pickup by the Microprocessor. CRC checking of the command stream is performed by the VHDL code to reduce the workload on the Microprocessor as much as possible.

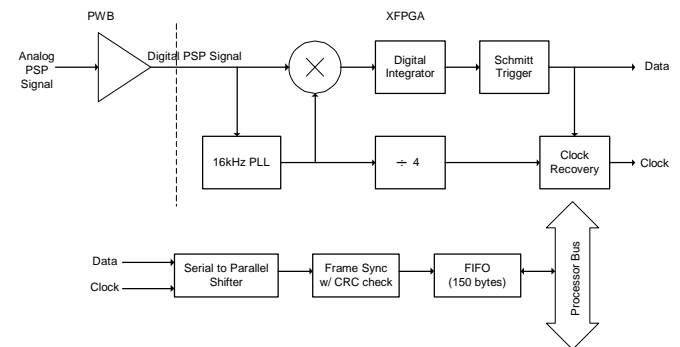


Figure 3. PSP Command Decoder.

G. PDI Telemetry Generator

The PDI telemetry generator consists, at its highest abstraction, of a telemetry generator engine and dual-port block RAM. The engine, written in VHDL and implemented in the XFPGA, is highly configurable. Parameters include the size of the telemetry frame, telemetry bit rate, encoding scheme (NRZ-L, -M, -S, Biphas-L, -M, and -S), and header synchronization pattern. The default configuration is a 128-byte frame, 4 Kbps, NRZ-L, with a 3-byte sync pattern 0xFAF320. The last two bytes of each telemetry frame hold

the frame CRC, calculated using the standard CCITT generator polynomial 0x1021.

The telemetry generator only reads from the dual-port block RAM. That RAM is populated through its other port by the microprocessor. The microprocessor knows when a new telemetry frame begins (and hence, when to start repopulating the block RAM) by watching for address rollover from the telemetry generator. By design, the microprocessor's operations require less time than a telemetry frame, so the microprocessor has some idle time while waiting for the start of the next telemetry frame. Also, since the first three bytes of each telemetry frame begin with the same synchronization pattern, the microprocessor has ample time to begin collecting analog and digital data to populate the block RAM and always remain ahead of the telemetry generator. Thus, the data in each frame is always current.

In order to increase flexibility and reliability in the telemetry generation process, the microprocessor's populating of the block RAM and telemetry generator function are independent. For example, the telemetry generator itself computes the CRC from whatever bytes it reads from the block RAM rather than relying on a microprocessor-generated CRC. That is, regardless of what the microprocessor may have written into the last two bytes of block RAM, these are discarded and the VHDL-generated CRC is inserted. Thus, even if the microprocessor could not collect data as quickly as the telemetry generator could send it out, there is no fear that an old CRC would be sent simply because the microprocessor did not have time to refresh it.

For our system, the telemetry frame consists of status bytes showing the current system configuration, frame counters, the last two commands properly decoded, bi-level data showing EPDSU status, EPDSU current, voltage, and temperature analog readings, and status bits showing comparison errors among the three XFPGAs. This last data is collected by the AFPGA and is made available to the XFPGAs through the SCB. Its purpose is to give insight into the occurrence of SEUs.

H. Reset and Synchronization

The simple and instantaneous voting of the outputs of the three XFPGAs requires their precise synchronization. The "Sync Generator" in Figure 1 performs this function. This module resets all subsystems inside the XFPGA except for the Command Decoder. A pulse is issued shortly after Power Up as the Power-On-Reset, and then every 512 complete telemetry frames thereafter (about 131 seconds). The occurrence of the sync signal has the following effects:

- Master reset on the Microprocessor, resulting in a reboot. This action mitigates SEUs that may have corrupted an important register in the Microprocessor, which would have caused it to malfunction (stack pointer, program pointer, register, etc). This causes the Microprocessor to jump back to the top of instruction memory and start running code as if it were just turned on. As a result of this operation, all registers are corrected. Data that needs to be preserved across telemetry frame boundaries

(such as command counters, frame counters, etc) is kept in the SCB slave memory. Note that the sync occurs during the "idle" period of the C code, and does not disturb its actions.

- Reset of the telemetry generator. Note that the sync signal is much shorter than one telemetry bit, and causes the telemetry generator to issue the first bit of the first byte. Since the interval is timed precisely with the telemetry frames, the sync does not disrupt telemetry. This operation mitigates corruption of the memory that controls or holds the contents of telemetry.
- Reset of SCB Master. Any corruption due to SEUs is cleared as a result of this. Since this occurs during the idle period of the C code processing, this does not corrupt a transaction.
- Configuration check of the three XFPGAs. This removes the effect of SEUs in the configuration memory of the devices. See the section on "Scrubbing" below.
- Reset of other minor modules inside the XFPGAs, completing the synchronization of the entire three XFPGA system.

Note that the sync generator does not reset the Command Decoder. This is because commands arrive asynchronous to the STM, and it would be unacceptable to lose commands simply because they arrived in the middle of a sync event. The Command Decoder is instead reset by logic in the C code that notices that the command counter on the other XFPGAs is incrementing while its own is not. The Microprocessor then uses a line in its "Local I/O" to reset the Command Decoder.

SEU MITIGATION

A. Scrubbing XFPGA configuration

Xilinx RAM-based FPGAs allow the device to be reloaded with its configuration data without halting operation. This is technically termed scrubbing. Because of the issue of half-latches (discussed later), the only way to ensure that all faults have been corrected is by erasing the device and then fully reloading. Consequently, the device must, of necessity, halt its operation. This consideration, coupled with overall system robustness, led to the use of three independent XFPGAs rather than a design which incorporated TMR within a single part. Erasing and fully reloading the configuration data of a device is termed full reconfiguration.

During the reconfiguration process, while new configuration data is being loaded, there is an increased probability that an SEU will corrupt the data. Conversely, the longer a device operates without being reconfigured the higher probability that it will have been corrupted. To balance these competing issues, we decided on an approach that checks the XFPGAs for corruption often but only reloads them when they are erroneous. The approach we have selected also allows us to minimize SEU probabilities on the EEPROM whose total dose and SEU susceptibility increase

when powered, although for our short mission duration, we do not actually power down the EEPROM between reloads.

Xilinx RAM-based FPGAs allow one to read configuration data from the device as well as load it. Our approach to module integrity is to routinely perform comparisons of the configuration of all three XFPGAs. This comparison is performed byte-by-byte. If during the readback any XFPGA's data disagrees with that of either of the other two devices, that FPGA is flagged as needing reconfiguration. At the end of the readback cycle, those FPGAs indicating failure are reloaded sequentially.

Documentation concerning XFPGA readback and reconfiguration is found in Xilinx Application Notes (XAPPs) [4] and [5]. Configuration is the easier task because the job of the Actel FPGA is merely to clock data out of the Xilinx PROM and write it to the XFPGAs. All startup and initialization sequences and data necessary to get the XFPGA configured and running are part of the data stream held in the PROM. The only two concerns are how to prepare the XFPGA to accept the data and how many bytes to clock out of the PROM for a given size XFPGA. As [5] points out, the XFPGA is prepared to receive commands and data by sending it a disable sequence followed by an abort sequence. The disable and abort sequences are merely a series of clock cycles (rising edges) with the configuration lines in a particular state. Table 4 in [4] then provides the information concerning how many data bytes to send. For our device, the XCV600, there are 3,607,968 configuration bits, or 450,996 bytes. After the last byte is clocked in, a one-clock-cycle disable sequence completes the task. Full configuration requires about 225 msec (per device) using our 4 MHz system clock.

XFPGA readback is only a little more difficult because one must first send the XFPGA a command sequence telling it that it is to output configuration data. That complete command sequence is provided in Table 13 in [4] with the missing item, the number of 4-byte configuration words being requested, supplied by Table 14 in [4]. These commands are prefaced with the same disable and abort sequences used for configuration, in order to prepare the XFPGA to receive the command. From Table 14 in [4], for the XCV600, there are 108,810 configuration words, so we must clock the data out, a byte at a time, using 435,240 clock cycles. Though the data clocked out is a mixture of configuration data and reserved words used for retrieving flip-flop data from the device, since all three of our XFPGAs are operating in lock-step, we merely compare all the received bytes for mismatches. This eliminates the need for a mask file which would be required if we were to compare the XFPGA contents against data stored in the PROM. The mask file is as large as the data file, so it would also increase the PROM storage requirements as well as the logic to interface with the mask file.

In the current design, the Actel FPGA's readback and configuration logic remains in idle until it reaches timeout or is strobed by the synchronization pulse. When brought out of idle, the Actel logic first checks to see whether any of the three XFPGAs indicates that it lacks configuration data (as

noted by a latched "done" line from the XFPGA). If any show they are not "done", readback is skipped and the logic jumps directly to the configuration portion. This is the sequence which is followed on power-up. If there is no indication of an unconfigured XFPGA, the Actel performs readback and comparison on the three units. If all are in agreement, then the timer is restarted and the readback and configuration logic returns to idle. The timeout period is set for 5 seconds if devices remain unconfigured or 180 seconds if all received their configuration data and reported "done". The synchronization pulse which occurs every 512 telemetry frames (about 131 seconds) arrives before the 180 second timeout, so this timeout period only forces readback and configuration if the synchronization pulse should be absent.

The one configuration error which cannot be detected by readback is the issue of half-latches. Xilinx synthesis tools use half-latches as an economical means of initializing certain data. Corruption of half-latch data by an SEU is not detectable by read-back of the configuration. Their use in a design can be eliminated by a procedure developed by Xilinx. However, if half latches remain for some reason in a design, and if after several scrub cycles a particular XFPGA continues to show errors in its SCB or PDI outputs, a command can be sent to force a scrub of that particular unit. If the problem was corruption of a half-latch, the erasure and reconfiguration of the device will clear it up.

B. SEU Robust Data Storage

As mentioned before, any data that is needed across telemetry frame boundaries, such as command and telemetry frame counters, are kept in the SCB Slave memory. This memory is considered robust to SEUs that are expected for HST's orbit, and prevents the corruption of data. Data that is used only for one telemetry frame, such as analog data, is rewritten every time, and will only result in a temporary mismatch during voting. Other data in the XFPGA that is not rewritten such as program counters and stack pointers is reset when the Synchronization pulse occurs. With this collection of mitigation strategies, all known SEU mechanisms are addressed.

C. SEU Probabilities

SEU statistics have been measured for the 18V04 EEPROM, and the XQVR1000 device [3]. Although our device is 40% smaller, we can use this data to determine the upper bound on the probability of a corruption in the configuration RAM of the XFPGA using the Poisson Distribution:

$$p(x;m) = \frac{e^{-m}m^x}{x!} \quad (1)$$

where : x=number of occurrences, and
m=mean occurrences per period

The expected interval between SEU hits on the configuration memory is 1.1 days [3]. We first find 'm' in (1), which is the mean SEU per scrub period (131 seconds), presuming a utilization of 0.5 of the FPGA:

$$m = 1.51E-3 * 9.05E-1 * 0.5 = 6.86E-4 \text{ (SEU/scrub period)}$$

Substituting into (1), the probability of no errors ($x=0$) during a scrub period, for a given XFPGA, is:

$$p_0 = p(0;m) = e^{-m} = 0.9993$$

And the probability of 1 or more errors, for a given XFPGA, is therefore:

$$p_1 = 1 - p_0 = 1 - e^{-m} = 0.00068$$

For a three-XFPGA system, we now find the probability of more than one corrupted XFPGA in a scrub period. This is the probability of any two, or all three corrupted:

$$p(\text{system}) = 3(p_0 p_1^2) + p_1^3 = 1.41E-6$$

Finally, we can calculate the probability of a successful 14-day mission:

$$= (1 - p(\text{system}))^{\text{days} * \text{scrubs per day}} = 0.99999^{14 * 659} = 98.7\%$$

Thus the probability of no more than one corrupted XFPGA during any scrub period over the duration of a 14-day mission is 98.7%. The main parameters that control this probability are the scrub interval and the utilization of the XFPGA. To gain some insight into the sensitivity of these factors, a parametric analysis is performed, and summarized in Figure 4. As might be expected, the probability of success deteriorates as the scrub interval is lengthened or as utilization is increased.

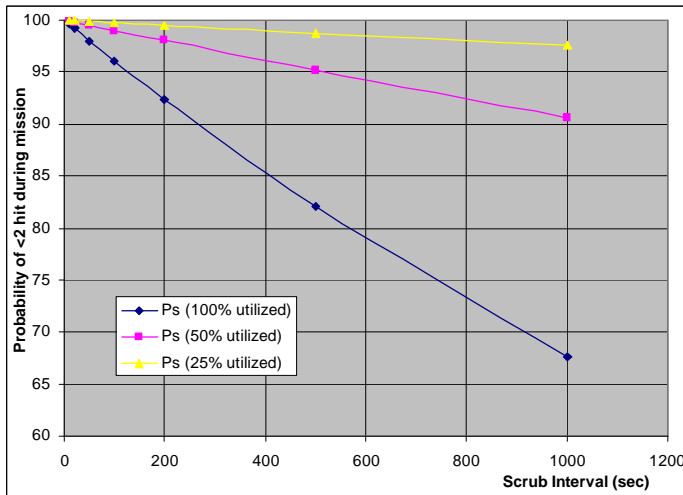


Figure 4. Effect of scrub interval and utilization on the probability of a successful mission.

By using similar analysis, we obtain the bounding results of the other SEU effects. The results are summarized in Table 1. Only the case of half-latches has no automatic mitigation scheme.

During the synthesis of the FPGA, the Xilinx tools on occasion use half-latches to store logic levels needed to terminate unused internal inputs. The half-latches are set during configuration and startup, and no means exist to either read or reset their state. It is possible for an SEU event to change their logic level, which will alter logic.

Unfortunately, the value of the half-latch cannot be corrected with scrubbing alone.

One means of mitigating this problem is to eliminate their use in the design by using a procedure developed by Xilinx on the resulting EDIF file. Or, an alternative method is to command a reconfiguration from the ground, as this toggles the “PROGRAM” line on the FPGA (completely blanking it before rewriting). This resets the state of the half-latches during the subsequent configuration cycle. In addition to the FPGA, SEU statistics for the 18V04 EEPROM are also known, and their results are also shown in Table 1.

VERIFICATION AND TEST

One important issue is how to detect that one or more XFGAs experience failures during ground test. After all, it is the point of the redundant system to suppress single errors. Errors must not be masked if they occur during ground test as they are not expected under those circumstances. At any time in the operation of the STM, if a voter sees one input in the minority, an error is latched, and a count is incremented. These error counters are downlinked, allowing feedback of the system state (both during ground test and flight).

The following error counters are downlinked, one for each XFPGA:

- Watch Dog time-out (Microprocessor halted/suspended).
- Error in SCB bus.
- Error in PDI telemetry stream.
- Error in XFPGA configuration memory determined by the Scrubbing system.

One additional feature is the ability, through a particular command, to “suspend” one of the Microprocessors. Although all three Microprocessors receive the identical command, each knows its identity number, and reacts to this command only if the number matches. This essentially causes that unit to go into an infinite loop in the C code and allows testing and verification of the watch dog functions, and the subsequent resynchronization on the next sync pulse. Since the sync pulse functions as a hard reset, it will start the processor back up. On the next sync pulse the three processors will be synchronized once again, and fault tolerance is restored.

On the board level, using the JTAG interface, the configuration memory of the XFGAs can be corrupted to test the scrubbing function. There is also a PSP command that forces the scrubbing and configuration of a particular XFPGA from the ground.

On the system level, the test operators can force the voters to select one of the XFPGA’s serial streams to verify correct operation of that unit. This function is also a valuable debugging tool.

Table 1. SEU Statistics Table.

SEE	Mitigation	Net Mission Effect
Corruption of FPGA Configuration Memory	Scrub on 131 second interval	Probability of <2 XFPGAs corrupted = 98.7%
Corruption of FPGA Block RAM	Use only for temporary storage	Expected time until >1 XFPGA hit is ~ 182k years
Toggling of FPGA CLB Flip Flops	Sync pulse every 131 seconds	Expected time until >1 XFPGA hit is ~511 years
FPGA Single Event Functional Interrupt/Power-On Reset (SEFI/POR)	Watch-Dog time-out causes power cycle or switch to other Side	Probability of no FPGAs corrupted = 92%
FPGA Half-Latches in design	Xilinx procedure and reconfiguration command	None
Corruption of EEPROM data	None	One corruption in 1.4 M years
SEU in EEPROM readout circuitry	Infrequent use of EEPROM. Compare does not use readout circuit.	If read once per minute, one corruption in 71 years.
EEPROM SEFI/POR	None	One corruption in 27M years

IV. CONCLUSION

The requirement for reconfigurability and the lack of a truly SEU immune reprogrammable FPGA has greatly driven the design of the STM. The benefit is the quick reprogramming of the functionality of the Avionics in order to respond to a change in manifest due to any on-orbit emergencies, but comes at the price of system complexity. Ultimately, our operational requirements were modest enough that we could have performed the same functions with a redundant state machine instead of using a Microprocessor running C code. However, the added complexity allows greater flexibility and modularity of the various functions inside the XFPGA.

In one configuration that is expected to be similar to the flight configuration, our utilization is as follows:

Table 3. XQVR600 Utilization.

Resource	Utilization
Block RAM	8%
Logic SLICES	53% ¹
IO Pins	18%

ACKNOWLEDGEMENT

The authors wish to thank Carl Carmichael from the Xilinx Hi-Rel group for his assistance during the design of the SEU mitigation strategies such as the scrubbing and the procedure for elimination of half latches. In addition, we also wish to thank members of the Design Review team including Rodney Barto and Richard Katz for their suggestions during the design reviews.

REFERENCES

- [1] C. Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs," Xilinx Application Note XAPP197, November 1, 2001.
- [2] ICD2-19001. Shuttle Orbiter/Cargo Standard Interfaces (CORE), rev L. Boeing North American, Inc. January 15, 1998.
- [3] GSFC Radiation Test Results. Internal memorandum from Christian Poivey (Code 561) to Rich Williams (OSC) dated 2/5/03.
- [4] "Virtex FPGA Series Configuration and Readback," Xilinx Application Note XAPP138 (v2.7), July 11, 2002.
- [5] C. Carmichael, M. Caffrey, and A. Salazer, "Correcting Single-Event Upsets Through Virtex Partial Configuration," Xilinx Application Note XAPP216 (v1.0), June 1, 2000.

¹ C code takes up 27%, and processor core 18%.