# Strategies for Fault-Tolerant, Space-Based Computing: Lessons Learned from the ARGOS Testbed[1][2]

M. N. Lovellette, K. S. Wood
Naval Research Laboratory
4555 Overlook Ave SW
Washington, DC 20375
202-404-7460
lovellette@xip.nrl.navy.mil

D. L. Wood[3]
Praxis Inc.

J. H. Beall[3]
St. John's College Annapolis

P. P. Shirvani, N. Oh, E. J. McCluskey
Center for Reliable Computing, Stanford University

*Abstract*—The Advanced Space Computing and Autonomy Testbed on the ARGOS Satellite provides the first direct, on orbit comparison of a modern radiation hardened 32 bit processor with a similar COTS processor. This investigation was motivated by the need for higher capability computers for space flight use than could be met with available radiation hardened components. The use of COTS devices for space applications has been suggested to accelerate the development cycle and produce cost effective systems. Software-implemented corrections of radiation-induced SEUs (SIHFT) can provide low-cost solutions for enhancing the reliability of these systems. We have flown two 32-bit single board computers (SBCs) onboard the ARGOS spacecraft. One is full COTS, while the other is RAD-hard. The COTS board has an order of magnitude higher computational throughput than the RAD-hard board, offseting the performance overhead of the SIHFT techniques used on the COTS board while consuming less power.

TABLE OF CONTENTS



## 1. INTRODUCTION

The Advanced Space Computing and Autonomy Testbed (ASCAT) on the Advanced Research and Global Observation Satellite (ARGOS) provides the first direct on orbit comparison of a modern radiation hardened (RAD-hard) 32 bit processor with a similar Commercial Off The Shelf (COTS) processor. This investigation was motivated by the continuing need for higher capability computers for spaceflight use than can be met with currently available radiation hardened components.

Radiation hardening has traditionally been required for components to accommodate the space environment. Component radiation susceptibility encompasses two distinct effects, total dose susceptibility (TDS) and Single Event Upset (SEU) susceptibility. TDS is permanent damage caused by interaction of charged particles with the devices, leading to higher leakage currents that eventually cause failure. Typically the annual radiation dose for a system in low earth orbit (LEO) is 1-5 kRADs/year. Many of the smaller LEO missions have total dose requirements in the 20 -50 kRAD range. SEUs are transient faults caused by the passage of a single charged particle and typically manifest themselves as a bit-flip -- an undesired change of state in the content of a storage element. Radiation induced SEUs are not restricted to the space environment, but also have been observed at ground level [1].

TDS may be enhanced by use of intrinsically radiation hard processes such as bulk CMOS or silicon on insulator (SOI), or by shielding more susceptible devices with a high Z material such as tungsten. Radiation hardening for SEU

[1] U.S. Government work not protected by U.S. copyright.

[2] This work was funded by the Office of Naval Research and by the Ballistic Missile Defense Organization.

[3] Under contract to NRL Code 7620.

tolerance can involve substantial changes to the device topology to mitigate charged particle effects. Because of the time required to modify commercial designs, these components lag behind today's commercial components in performance. It is not uncommon for the radiation hardened version of a part to be released five years after the initial commercial part, producing obsolescence on arrival. The need for state-of-the-art high performance computing systems for space flight applications, coupled with mission cost containment, provides a strong motivation for investigating alternatives to traditional radiation hardened devices.

Use of COTS devices has been suggested as a way to accelerate the development cycle and produce cheaper, faster systems [2]. It is possible to screen COTS components for appropriate levels of TDS, and shield them as required, but there is no similar fix for SEUs. Software-Implemented Hardware Fault Tolerance (SIHFT) techniques can provide low-cost solutions for enhancing the reliability of these systems without modifying the hardware [3][4][5].

There are several metrics that may be used to quantify system performance. Two selected for this study are the *number of unrecoverable* errors and the *total system availability*. These are partly related in that an unrecoverable error will lead to a loss of system availability, at least for the code module that sustains the error. The latter is probably the best choice of metric for a processor used as a payload controller or a data processor. It should be possible to implement spare processing modules that could immediately take over for one affected by an SEU. Very high reliability tasks, such as spacecraft control functions, typically have not been drivers for increased processor throughput.

The Unconventional Stellar Aspect (USA) experiment on the ARGOS satellite was typical of space systems whose data gathering capabilities greatly exceed their downlink bandwidths [6]. When the USA mission was manifested, space and power were reserved for the addition of one or two high performance computers to process data in flight. The concept was to provide an early flight test opportunity for several of the then active processor development programs. A survey was conducted to identify candidate processors in conjunction with the Integrated Science and Technology (IS&T) Office of the Ballistic Missile Defense Organization (BMDO). Four potential processors were identified, the Harris RH3000, TRW and Honeywell RH32s and the IBM RAD 6000. Of these candidates, the RH3000 program was able to work within the schedule constraints of the USA program and deliver a processor card. This left an opening in the USA electronics for an additional board, and at the suggestion of one of the authors, E. J. McCluskey, the idea testing of an approximately equivalent COTS processor to the RH-3000 was implemented, but this decision came late in the program and had to be implemented swiftly.

The Naval Postgraduate School developed an SBC based on the IDT-3081 processor. This processor was selected because of previous experience with it on the Micro Processor and Photonics Testbed [8] and Clementine [9] programs. This board contained completely COTS technology with the exception of its interface with the rest of the instrument, but provided as much as 25 times the computing power of the RH-3000 for less power. The board was fabricated and delivered to NRL for operating system development and interface testing with the USA brass-board (nonflight electronics) at a time when the rest of the USA experiment had already shipped to the space vehicle contractor. The IDT-3081 board underwent an abbreviated board level environmental test, thermal cycling and vibration, at NRL and was integrated with the USA instrument at Boeing. It then completed the nominal integrated test flow with the instrument including system level acoustic and thermal vacuum testing. The compressed design and fabrication schedule precluded any effort to include the capability to recover autonomously the state of the system after a major error. The response of the IDT-3081 board was thus limited to preparing for system recovery.

## 2. THE ARGOS TESTBED

*ARGOS, USA and ASCAT*

The ARGOS satellite is a multi-instrument platform built and operated by the US Air Force for the DoD Tri-Service Space Test Program (STP). STP provides a path to flight for payloads with DoD relevance produced by government laboratories. The ARGOS space vehicle weighed approximately 5400 pounds at launch and carries 9 experiments ranging from an arc-jet propulsion demonstration to space and upper atmosphere remote sensing. ARGOS was launched from Vandenberg Air Force Base into a 850 km 98.7° orbit on 23 February 1999. This polar orbit, slightly higher than usual for LEO satellites and crossing both the radiation belts and the South Atlantic Anomaly (SAA). presented radiation tolerance challenges for electronic parts. The total dose expected behind 2.5 mm of Aluminum shielding for the three year duration of the mission was 5 - 20 kRads [7].

The USA experiment is an X-ray timing instrument that provides energy and time of arrival to 1 μs precision for each incident photon. The polar ARGOS orbit is not ideal for an X-ray detector such as USA and the operational time period of the X-ray sensors was generally limited to the low latitudes between +45° and -45° and sometimes over the south pole. The X-ray sensors could also not be operated in the SAA, but by the nature of their operation, provided excellent diagnostic data in the low background portions of the orbit.

The USA instrument control processor is a Harris radiation

hardened version of the Intel 8086 running at 4 MHz. This part is a good example of the difficulty of obtaining space-flight worthy parts. In 1992, when the USA system architecture was created, the current state of the art for the 80X86 processor family was the 80486 running at 33 MHz. The other rad-hard processor was the GVSC-1750, which provided somewhat more capability than the 8086, but was much more expensive and had a much less complete software development environment. To accommodate the unique background environment the USA instrument would experience and to allow data packing modes to be tailored to the source features of interest, there was the strong desire to perform the USA data processing in a reprogrammable processor rather than in hardware. However, the X-ray data processing requirements for USA were beyond the capability of the Harris 8086. The actual sensor data formatting and telemetry frame building was performed by a RAD-hard Analog Devices ADSP-2100 digital signal processor.

*RH-3000 Rad-hard board*

The RH-3000 SBC was provided to USA by the Mission Development Branch of the Naval Research Laboratory Naval Center for Space Technology. The principal components of the RH-3000 processor are the RH-3000 CPU, RHFPA and RHMD which together function as the RH-3010 floating point coprocessor for the RH-3000 CPU, and two RHSC cache control devices. The RH-3000 CPU is clocked at 10 MHz. The board has 2MB of Silicon on Insulator (SOI) SRAM memory with hardware error detection and correction which is capable of correcting single bit errors and detecting double bit errors through a 4 bit symbol error correcting code. The address and data buses both have parity bits. The RH3000 is configured as a "master/shadow" system. Each processor component is duplicated and each calculation is therefore performed twice. The results are compared after each operation. A mismatch between the master and shadow processors causes an exception resulting in a system halt and reset. Startup code and operating system are stored in 128Kbytes of EEPROM. The interface to the rest of the USA electronics is through a 16Kbyte dual port memory device. The board also contains 32 Kbytes of SRAM for both the data and instruction caches. Unfortunately a hardware bug resulted in read-modify-write instructions periodically failing, requiring the board to be operated in uncached mode with a substantial performance penalty.

*COTS IDT-3081 Board*

The processor selected for the COTS board was the IDT-3081. This device is a single chip implementation of the MIPS RISC architecture which provides an instruction set compatible with the R30XX family of RISC CPUs. The IDT-3081 provides 16 Kbytes on chip instruction cache and 4 Kbytes on chip data cache. Both the instruction and data caches are equipped with parity, but initial versions of the fault-tolerance experiments did not make use of it. The bulk of the IDT-3081 operation time has been accumulated with the caches enabled. The COTS board also contains 2 Mbytes of memory, in the form of four Cypress Semiconductor CYM1465LPD-70C modules, 512K x 8 bits. *This memory has no hardware error detection or correction capabilities.* Nonvolatile storage is provided by 2 Mbytes of EEPROM configured as 8 segments of 256 Kbytes each. The COTS board interfaces with the USA instrument controller through a dual port RAM device in an identical fashion to the RH-3000 board.

## 3. THE ARGOS MISSION

The ARGOS satellite was launched 23 February 1999. The USA instrument was powered up on 30 April 1999. The ground rule for the computational portion of the experiment was that it would not interfere with the other ARGOS payloads or with the USA X-ray experiment. The computer power up and first software loads took place approximately 20 May 1999, after all other experiments on the spacecraft had been initialized. The first applications run were simple memory tests. These were used as much to debug the command and telemetry systems as to actually gather data from the instrument.

Even though nothing changes physically in the hardware, flight operations differ enormously from operations in ground tests. The required sequences take longer because of the additional communication links and any happenstance can produce further delays. All of this adds up to a qualitative difference in the character of operations. The software upload requirements of the ASCAT testbed placed the experiment toward the extreme end of the uplink requirements spectrum for STP missions. The uplink rate was 2 Kbits/second and a typical upload for the IDT-3081 was 52 Kbytes compressed (100 Kbytes uncompressed) at the beginning of the mission. Time was needed to debug the logistics of moving a large block of code reliably up to the vehicle. ARGOS typically had five or six contacts per day. Each contact was about ten minutes in duration. Code uploads could begin only after other vehicle commands had been sent and data transmission to the ground had been started. With the overhead imposed by the communication system and scheduling around other vehicle tasks, it often took a day to perform a complete code upload. The magnitude of the task was increased because early in the mission additional code loads were not committed to the onboard nonvolatile storage. This was done to prevent a fault leaving the system in a non-recoverable state. As the level of experience with the system increased, the EEPROM memory was used to store frequently executed code modules reducing the upload requirements to approximately 25 Kbytes, compressed, and a wider range of software experiments were then conducted, often with several running concurrently. These fall into two broad classes, memory tests and error detection/correction experiments.

Figure 1. Software Test Uplink Flow

The procedure required to perform a software test in flight was quite lengthy; Figure 1. It was possible at all only because the processors were not in a flight critical position. Many of the fault-tolerant techniques were developed before launch using simulations and artificial fault injections. Once a fault-tolerant method was identified for use, it was ported to the ARGOS environment. This module was first loaded and run on the USA electronics test bench setup at NRL. The test setup includes fully functional brass-board copies of the USA instrument control processor, the IDT-3081, and the RH-3000, as well as an ARGOS space vehicle simulator. At this point, the software modules were expected to run smoothly for at least a day without indicating any errors. Once a software module had passed the bench test, the module object file was converted to the mission operations upload format (CUTF) and dispatched to the mission operations center. The time from the beginning of bench testing to the time that the software module began running in flight was about two weeks. A software experiment on ARGOS wrote its results into the space vehicle solid-state recorder (SSR) once per orbit. The SSR was then dumped to a ground station between one and six times a day. Depending on the location of the downlink ground station, the SSR dumps were made available from the mission operations center within one to fourteen days. The downlink data was then passed through a telemetry filter application to extract the ASCAT results in a useable format.

Thus, the nominal software experiment took about a month from beginning of development to execution. The time to gather meaningful results, however, was longer since many of the ASCAT software modules require long run times to produce statistically significant numbers of errors. The test software modules were also not exempt from bugs. Most software modules have had several major revisions. The results presented in this paper are limited primarily to the most recent versions of the software.

## 4. Software Experiments

Fault tolerance software experiments divide into two broad categories, systems diagnostics to define the primary fault categories and the subsequent fault-tolerant tasks designed to mitigate those faults. The primary systems diagnostics were memory test routines that continually checked for SEUs in the system, with characterization upon detection.

*Operating Systems and Infrastructure*

We selected the VxWorks operating system (OS) from Wind River Systems to provide low-level system services for the ASCAT processor boards. The RH-3000 port of VxWorks was provided by the Naval Center for Space Technology. This port was already in existence in the early phases of the ASCAT experiment. The VxWorks port for the RH-3000, therefore, is somewhat older than the reset of the ASCAT software and is based on VxWorks version 5.0.1. The OS binary image is installed in the RH-3000 EEPROM in a compressed format along with a small bootstrap module. The OS port for the RH-3000 has some custom features to handle the special fault-tolerant hardware features included in the processor chip set. The IDT-3081 VxWorks port was provided by NRL at a later date. The IDT-3081 OS port is based on VxWorks version 5.3.1. As with the RH-3000, the IDT-3081 OS binary image is stored in EEPROM along with a simple bootstrap module. Because the IDT-3081 board contains a more generous amount of EEPROM, the OS image is not compressed. At power on or reset, the bootstrap module for both boards initializes the hardware, inflates or copies the OS image into a known location in SRAM, and jumps to the OS entry point.

Both the RH-3000 and IDT-3081 OS ports also contain software interfaces the dual-port memory. These low-level drivers allow the USA instrument control processor to send the boards commands, file uploads, and time and attitude information. The dual-port memory drivers also allow applications on either board to send telemetry information to the USA instrument control processor. In addition, the later work on the IDT-3081 system software allowed a compression feature to be built in to the file upload code. This feature is not available for the RH-3000. The OS memory images are quite large in comparison to the 2 Mbytes SRAM available on the boards. The OS code segment occupies about 340 Kbytes of memory, and the OS static data segment occupies about 135 Kbytes of memory. In addition, the OS internally allocates and frees memory dynamically at run time when system objects are created or destroyed. When the ASCAT experiment began, the OS images and bootstrap modules were the only on-board software; all other applications needed to be uploaded from a ground station.

*RH3000 System Diagnostics*

Because the RH3000 OS was ported to the board much earlier than the IDT-3081 OS, the RH-3000 system file loader did not contain code to allow for compressed upload modules. The resulting reduced upload bandwidth severely

limited the size of the upload modules which could be reliably uplinked to the RH-3000 board. Since the IDT-3081 provides a much better testbed for the fault-tolerant software experiments, it was decided early on in the experiment to allocate most of the available upload bandwidth to the IDT-3081 uplinks. The RH-3000, however, was regularly loaded with two test applications.

The first RH3000 application is a simple memory test which uses a fixed pattern of alternating 1's and 0's. The fixed pattern is written a large (512 Kbytes) memory buffer at startup. The memory test then scans the test buffer, looking for a change in the known pattern. The second application is a test of the floating point unit. This application has included a large, static buffer of values. The application continuously calculates the trigonometric sin value of each of the input values. The resulting table is then checked against the known correct values.

*IDT-3081 System Diagnostics*

Memory tests are perhaps the most important of the ASCAT applications. The first application uploaded and executed on either board was the simple memory test. It was chosen as the first application for a number of reasons. First, the memory test is relatively easy to write and test. The small code size results in a small upload size. The SRAM on the IDT-3081 board, by virtue of its gate count, is the most vulnerable component to radiation SEUs. Thus, a small memory test estimates the error conditions without itself being subject to significant internal errors. If we assume that all SRAM will exhibit similar error probabilities, it should then be possible to gauge the error probabilities for different applications by examining the amount of SRAM an application uses. If the error rates observed during the application's execution period are in the same range as the estimated error rates, then it follows that the errors encountered by the application are most likely the result of SRAM SEUs. If the observed error rates differ significantly from the estimated then components other than SRAM may be contributing to the application's overall SEU rate.

Two different memory tests were run over the course of the experiment on the IDT-3081 board. The first memory test uses a fixed pattern of alternating 1's and 0's. The fixed pattern is written to two large (256 Kbytes) memory buffers at startup. The memory test then scans the test buffers, looking for a change in the known pattern. One buffer is accessed through the data cache and the other buffer is accessed by bypassing the data cache to determine if the cache memory shows a noticeable effect on the memory error rate. The second memory test uses a single, uncached test buffer, but periodically re-initializes the buffer with a different test pattern. For example, the memory test uses all 1's, all 0's, and different cases of alternating 1's and 0's. The variable test patterns are designed to determine what effect the patterns have on the measured memory error rate. Differences in the error rates for different patterns may lead to conclusions about how the underlying memory chip hardware design affects the reliability performance.

*IDT-3081 Operating Systems Augmentations*

When the measured SRAM SEU rate of ~6 x $10^{-7}$ SEU/bit-day is applied to the OS static memory allocation figures, the following error rates are expected in OS SRAM segments:

OS code segment estimated SEU rate = 1.9 SEU / day
OS data segment estimated SEU rate = 0.7 SEU / day

These expected rates indicate that the OS SRAM SEUs are probable enough to warrant attention. All test applications rely on the underlying OS services to manage memory, processor time, and I/O. Early attempts at running test applications showed the IDT-3081 system as a whole was not stable. In particular, uploading new versions of test applications was not very successful. It became obvious, from memory test calculations and flight experience that OS memory segments needed protection. Otherwise, actual test applications would never run long enough and reliably enough to gather useful data.

This issue was addressed early in the experiment[10]. Any solution must be transparent to the on-board OS image. Replacing the OS image stored in EEPROM is infeasible for two reasons. First, the binary image is too large to upload using ARGOS ground stations. Second, even if a replacement were possible, the lack of OS source code makes the implementation of more sophisticated fault tolerance features nearly impossible. We eventually adapted the general technique of Error Correcting Codes (ECC) to the IDT-3081 environment. ECC codes are well-known methods for autonomously correcting errant bit patterns for mass storage and communication devices. Each ECC implementation provides a certain level of detection and correction capability. Usually, those methods that provide better detection and correction ability also require greater resources such as more memory storage or longer encode/decode times. Normally, the ECC codes are created by a hardware encoder, which generates the correction bits based on the bits of the word is being committed to storage. The correction bits are stored along with the actual data. When the data word needs to be retrieved from storage, a hardware decoder checks the data word bits against the correction bits. Depending on the sophistication of the ECC code, the decoder may be able to detect one or more bit errors in the data word encountered. Again depending on the sophistication of the ECC code, the decoder may either autonomously correct the data error, or it may signal that it encountered an uncorrectable error. Usually, the decoder detects errors when data is read out of storage. However, this policy can lead to several bit errors accumulating in stale locations that are not read frequently. The buildup of bit errors may surpass the detection and correction of

capabilities of the ECC code. Periodic *scrubbing* can help solve this problem. It forces a check of all of the stored correction bits within a certain minimum time interval. The scrubbing eliminates the stale storage locations, but it also imposes a performance penalty because the scrubber contends with normal application software for access to memory.

Keeping with the idea of testing COTS hardware in the space environment, the IDT-3081 board does not provide hardware ECC protection for its SRAM. We decided to implement a cyclic ECC code with a software encoder and decoder. This code provides single bit error detection and correction. It may also detect errors of two or more bits but then cannot provide corrections. The ECC software is implemented as a high priority VxWorks task running every 5 seconds. When ready, the ECC task preempts any running test applications and performs a scrub of the OS code segment. SEUs in the code segment are corrected, and a telemetry report is generated giving byte address, bit number, GPS timestamp, and the ARGOS space vehicle location. The ECC task also contains a general-purpose interface that allows test applications to register a code or constant data segment for protection. In these cases, the ECC task is configured either to scrub these segments periodically (along with the OS code segment) or to perform a correction check on demand.

*IDT-3081 Fault Tolerant Applications*

Although the memory test and ECC applications provide good tools to measure overall hardware performance, it is envisioned that a general purpose, space-based computer would run applications that use more conventional algorithms. One objective for space-based fault-tolerant computing is that the error detection and correction mechanisms should be transparent to the implementation of the algorithm. For instance, it should be possible to write an application to analyze instrument data without the programmer being burdened with knowledge of the complete details of the various error detection and correction methods. Many of the transparent methods available involve duplicating a software calculation multiple times. When not all of the results agree, a transient error occurred in one of the calculations. Duplication can be done at task-level by the programmer or by the OS. Three copies of a task all perform computations on a data set, and the results are not accepted unless two out of the three tasks agree. It can also be done at instruction level during program compilation. We have developed a technique called *error detection by duplicated instructions* (EDDI) that uses the latter approach. Figure 2 below shows a sequence of instructions (a) and how it is transformed for EDDI (b). Computation results from master and shadow instructions are compared before writing to memory. If the two values do not match, the program jumps to an error handler that will cause the program to restart. Details of this technique can be found in [11].

```
ADD R3, R1, R2          ; R3 <- R1 + R2
MUL R4, R3, R5          ; R4 <- R3 * R5
ST  0(SP), R4           ; store R4
```

(a)

```
ADD R3, R1, R2          ;master instruction
ADD R23, R21, R22       ;shadow instruction
MUL R4, R3, R5          ;master instruction
MUL R24, R23, R25       ;shadow instruction
BNE R4, R24, Err        ;compare results
ST  0(SP), R4           ;master result
ST  offset(SP), R24     ;shadow result
```

(b)

Figure 2 - EDDI Example

The EDDI method can detect some of the control-flow errors. To further enhance the detection coverage for this type of error, we have developed a technique called *control-flow checking by software signatures* (CFCSS). Signature monitoring is a well-known method for control-flow checking. In this method, a signature is associated with each program block. This signature is stored in memory and checked during the execution of the program. CFCSS is an *assigned* signature method where unique signatures are associated with each block during compilation time. These signatures are embedded into the program using the immediate field of instructions that use constant operands. A run-time signature is generated and compared with the embedded signatures when instructions are executed. Figure 3 below shows an example of instructions (a) and how it is transformed with CFCSS (b). In this example, R30 (any of the general-purpose registers of the processor can be used for this purpose) holds the run-time signature and is updated as execution moves from block to block. Upon entering a block, R30 is XOR'd with a constant to generate the signature of the current block. This value will be correct only if the correct sequence of blocks has been followed. The assigned signature of the current block is compared with the run-time value. If the two values do not match, the program jumps to an error handler that will cause the program to restart. Details of this technique can be found in [12].

```
ADD R3, R1, R2      ;A branchless block
MUL R4, R3, R5      ;  of instructions
ST  0(SP), R4       ;
```

(a)

```
XOR R30, R30, 0x3c  ;Gen. run-time signature
LDI R10, 0xb7       ;Load assigned signature
BNE R30, R10, Err   ;Compare the two
ADD R3, R1, R2      ;Continue normal
MUL R4, R3, R5      ;  sequence if
ST  0(SP), R4       ;  correct signature
```

(b)

Figure 3 - CFCSS Example

Both of these techniques are transparent to the original software algorithm since they are not implemented until compile time. The schematic below shows the system we

used for the ARGOS experiment, Figure 4.

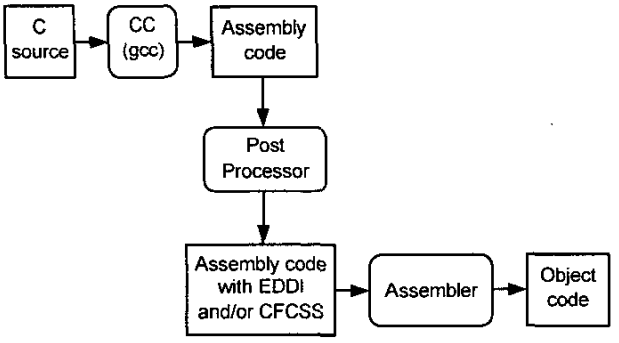


Figure 4 - EDDI and CFCSS Code Generation

*Sorting and FFT Algorithms*

In order to provide a realistic evaluation of these types of applications, a small set of algorithm tests was written. The two algorithms selected for test are sorting and the Fast Fourier Transform (FFT). Normally, these tests would be run on input data from a space-based instrument. In order to maintain the simplicity of the experiments, however, the test applications simply use built-in tables of sample data. This method also has the advantage that the results of the algorithms are well known in advance, so it is easy to determine if the applications are in fact running correctly. Three test applications were developed using the EDDI and CFCSS techniques: an insert sort test, a quick sort test, and a FFT test. The original versions of these applications produced hardly any error reports. Since by this time, we had already concluded that the SRAM memory was the primary source of errors, the memory cross section that these applications occupied was increased dramatically. This was done in two complementary ways. The insert sort and FFT applications had the input data buffers duplicated multiple times, with the applications periodically switching to a new test data set. This is known as the *data expansion* technique. The quick sort application had the instruction code segment duplicated multiple times, with the application periodically switching to a new set of instructions. This is known as the *code expansion* technique. Once these expansion techniques were employed, the error reports from the applications increased dramatically.

The insert sort, quick sort, and FFT test applications all use the EDDI and CFCSS methods to detect errors within their own code and data memory segments. These two methods can also detect errors that occurred within the IDT-3081 processor itself. An EDDI or CFCSS error causes the monitor task to restart the errant test application. If the error was caused by an error in the processor or in the memory data segment, this first restart is successful. If the test application produces another error immediately on first restart, it is most probably due to an error in the memory code segment. In this case, a request is sent to the software ECC to check for errors in the code segment, and a second restart is attempted. In case the second restart fails, the task is considered unrecoverable and is dropped from the list of active test applications (the test application is suspended). Since the memory where the OS and applications run is not completely reliable, it is very difficult to isolate the exact cause of the error. One can always make the case that an apparent processor error was in fact a memory error that occurred in an unprotected region. A first analysis of test application errors is to examine number of errors detected versus the amount of memory used. If the error rate is comparable to memory test error rates we tentatively conclude that the errors are being produced in memory.

There are also several pieces of support code loaded along with the test applications. A software monitor provides watchdog services for the test applications. When a test application times out, the monitor task will attempt to restart the errant program from the beginning. An application is given two retries before it is considered unsalvageable and removed from the list of active programs. Also, an error log module is loaded. It collects reports from the test applications and software monitor, arranges them into a common format, and passes them to telemetry.

## 5. RAD-HARD BOARD RESULTS

One great surprise of the mission is that even though the RH3000 board uses radiation-hardened technology, the test applications show a small non-vanishing number of errors occurring over time, Figure 5. From day 1999/357 to day 2001/243, the memory test collected 7 errors over 543 days of actual run time. The test buffer was 512 Kbytes. This results in an error rate of $3.1 \times 10^{-9}$ errors/bit-day. While this low rate reflects the memory technology in use, it is higher than the $< 1 \times 10^{-10}$ rate expected for the board [13]. This is because the memory controller on the RH3000 board contains an ECC encoder/decoder pair, which should be able to automatically detect and correct single bit errors before the software memory test is able to see them. The ECC hardware has never generated a memory error report. Either the memory hardware ECC is not configured or functioning properly, or errors are occurring at a place somewhere other than the memory or processor (the memory bus buffer chips are a good candidate). It is interesting to note that all memory test errors collected from the RH3000 board involve bit flips from a '1' to a '0'. Also, all of the errors are located in only two bits out of the four bits set to '1' in each eight bit test word.

The sine table test application also exhibits errors on the RH3000 board. From day 2000/130 to day 2001/243, the sine test application collected 18 errors over 444 days of actual run time. Since the test buffer was 512 Kbytes, this results in an error rate of $9.7 \times 10^{-9}$ error / bit-day if only the effects of the memory exposure are considered. This error rate is more than three times higher than the memory test produces on the RH3000 board and is two orders of magnitude higher than expected. It is unclear whether additional errors from components other than the memory

lead to the higher error rate. The geographic locations of these errors correlates well with high particle regions such as the SAA and the radiation belts.

The RH3000 also sustained one autonomous reboot. This occurred in the northern radiation belt. this action is commensurate with the expected response of the RH3000 hardware to a miscompare between the two processors.

Figure 5 - RH3000 SEU's (green star) and reboot (red diamond) locations

Figure 6 - Summary of all IDT-3081 SEU's

## 6. COTS BOARD RESULTS

During the same time period the various IDT-3081 tests produced a total of more than 2000 SEU's detected and more than 50 task exceptions and reboots. These results, presented in Figures 6 and 7, also show strong correlation with regions of high particle background.

Figure 7 - Summary of all IDT-3081 task exceptions (blue crosses) and reboots (diamonds).

*Memory Test Results*

The IDT-3081 fixed-pattern, simple memory test collected 113 errors from 1999/244 to 1999/345 in 84 days of actual run time. The test buffer was 256 Kbytes, divided into a 128 Kbyte cached buffer and a 128 Kbyte non-cached buffer. The overall error rate is $6.4 \times 10^{-7}$ errors/bit-day. Of the errors that occurred, 54 were located in the cached region, and 59 were located in the non-cached region, showing little effect of utilizing the IDT-3081 data cache. One should note, however, that the memory test continually reloads and flushes the data cache, so this result may not apply to more general purpose software applications. For both cached and uncached tests bit flips from '1' to '0' are favored over SEUs from '0' to '1', Table 1.

Table 1- IDT-3081 Memory Test SEU Distribution

| | Errors 1->0 | Errors 0->1 | Ratio 1->0/0->1 |
|---|---|---|---|
| Cached | 36 | 18 | 2.0±0.5 |
| Non-cached | 32 | 27 | 1.2±0.3 |

The second memory test was written to further investigate the dependency of error rates on the particular bit pattern stored in memory. The IDT-3081 advanced memory test uses a 256 Kbyte test buffer and always accesses the buffer through the data cache. From 2000/146 to 2001/231, this memory test collected 290 errors in 228 days of actual run time. The resulting error rate is $6.1 \times 10^{-7}$ error/bit-day, which is very close the error rate produced by the simple memory test. This second memory test shows noticeable dependencies of the error rate on the particular bit patterns used in the test buffer. Each test pattern is a repetition of one of the following 16-bit test words: 0000 hex, FFFF hex, 00FF hex, FF00 hex, AAAA hex, AA55 hex, or 5555 hex. Each test pattern shows a different error rate, and each test pattern exhibits a different ratio of errors involving bit flips

from '1' to '0' verses from '1' to '0'. The ratios for those patterns involving alternating bit patterns are shown below.

Table 2 - Advanced Memory Test SEU Distribution

| Test Pattern | Ratio 1 -> 0 / 0 -> 1 |
|---|---|
| 00FF | 1.8±0.6 |
| FF00 | 1.4±0.5 |
| AAAA | 0.9±0.4 |
| AA55 | 1.3±0.4 |
| 5555 | 1.7±0.5 |

The differences in error rates between different patterns may be attributable to the details of the mechanical and electrical design of the memory chips [5].

*Fault Tolerant Applications Results*

The ECC task periodically scrubs the OS code segment, the monitor and error log code segments, the advanced memory test code segment, and the sort and FFT test application code segments, approximately 400 Kbytes of memory that is monitored for errors. From day 2000/131 to day 2001/233, the ECC task collected 590 errors in 287 days of run time. The resulting memory error rate is $6.3 \times 10^{-7}$ error/bit-day. This is in very close agreement with the memory tests although the error detection method is quite different.

The error data for the fault tolerant applications is summarized in Table 3. The three applications all utilize the same techniques, but are configured to give a small application with a large memory buffer, a large application size with a small data buffer, and extensive use of the IDT-3081 floating point unit.

The insert sort test application has a 4 Kbyte code segment and a 160 Kbyte data segment after data expansion. The insert sort test collected 145 EDDI errors from day 2000/135 to day 2001/226 during 203 days of actual run time. This results in a memory error rate of $5.4 \times 10^{-7}$ for the insert sort test application. The lack of other errors types is in consistent with the small code size of the application.

Table 3 - Fault Tolerant Application SEU Summary

| Error Type | Insert Sort | Quick Sort | FFT |
|---|---|---|---|
| EDDI | 145 | 30 | 79 |
| CFCSS | 0 | 8 | 1 |
| Timeout | 0 | 7 | 1 |
| Assertion | 0 | 3 | 0 |
| Processor Exception | 0 | 10 | 1 |
| Total | 145 | 58 | 82 |

The FFT test application has a 6 KB code segment and a 80 Kbyte data segment, with data expansion. The FFT test collected the following errors from day 2000/131 to day 2001/210 during 216 days of actual run time. This results in a memory error rate of $5.4 \times 10^{-7}$ for the FFT test application.

The quick sort test application has a 75 Kbyte code segment, with code expansion and a 8 Kbte data segment. The quick sort test operated from day 2000/131 to day 2001/227 resulting in 220 days of actual run time. The resulting total error rate for the quick sort test application is $3.9 \times 10^{-7}$. The large code size of the quick sort provided more opportunity for SEUs that were not detected by the ECC code and resulted in processor errors. These SEUs could have occurred in the application code between passes of the ECC task or in the data segments related to the application.

All of these test applications report error rates that are actually below the error rate for the memory test. The quick sort test application shows the most variety in error types reported. This is most likely due to the expanded code size of this application. The quick sort test application also produced three assertion errors. These occur when the application has finished sorting its test data set, but the results are not in fact sorted correctly even though no other type of error was seen. The insert sort and FFT application errors are almost all EDDI errors. This may result from the large data buffers employed by these tests. The insert sort application uses a data buffer which is twice the size of the buffer the FFT application uses, and the insert sort application produced about twice as many EDDI errors during approximately the same amount of run time.

The FFT application did report one error due to a data cache upset. The test applications always check the contents of their test data sets whenever an EDDI error is encountered. This check is done a second time with bypassing the data cache when a data error occurs. If the second check which bypasses the cache shows no sign of the error, then the error must have occurred in the data cache. The IDT-3081 data cache hardware parity, however, did not report an error at the time the FFT reported a software data cache error. This missing hardware error report is a mystery.

During the application run time shown in Table 3, the IDT-3081 board also automatically rebooted itself six times. Two of these reboot instances are known to be VxWorks OS critical exceptions. When the OS encounters an exception inside an interrupt handler, inside an exception handler, or inside the core of the kernel scheduler, it will reboot the processor to prevent an infinite recursion of exceptions. The causes of the other four automatic reboots are unknown.

All of the EDDI and CFCSS errors reported in Table 3 are considered successful corrections of SEUs. In all of these

cases, the test application was shutdown and restarted without any adverse effects on the rest of the system. The other errors shown in Table 3, while successfully detected, might not be considered successful corrections. For the assertions, the test application produced incorrect results. The timeouts and task exceptions usually resulted in the test application being suspended from execution. And obviously, the automatic reboots result in the stoppage of all of the test applications. When the six automatic reboots are added in to the figures from Table 3, the successful SEU correction rate is about 90%. The individual algorithms better suited to the software error detection and correction techniques had error correction rates in excess of 99%.

*SEU Orbital Effects*

The SEU pattern exhibited by the COTS board generally follows the distribution expected from the combination of the SAA and the radiation belts, but also shows an excess of SEUs after the vehicle has departed the SAA. This is evidenced by the asymmetry between the number of SEUs observed when the vehicle is on a north bound track as opposed to a south bound track. This effect could be caused by activation of material in the instrument by charged particles in the SAA, or by geomagnetic effects.

Table 4 - SEU data vs orbital direction near the SAA

| Latitude range | SEUs north bound | SEUs south bound |
|---|---|---|
| 40° N - 60° N | 3 | 2 |
| 6° N - 40° N | 15 | 1 |
| 0° - 6° N | 17 | 3 |
| 0° - 5° S | 11 | 13 |
| 25° S - 27° S | 34 | 40 |
| 30° S - 40° S | 60 | 107 |

Table 4 presents SEU data for the year 2000 for longitudes between 0° and 90° west. Each band of latitude shown has the full year of SEU data. Bands not enumerated have not yet been analyzed, this is the region from 5°S to 25°S in the core of the SAA. To the north of the SAA there is a clear excess of SEUs for north going tracks until the northern radiation belt is reached. To the south of the SAA there is also a statistically significant excess on south bound orbits, but the difference is not as large because of the proximity of the southern radiation belt. This asymmetry has at least two potential causes, the first is activation of the material in the vicinity of the processor card or on the processor card, and the second is a directional flux of particles from the SAA to the USA instrument. SEUs from activation could be produced by the decay of isotopes with half lives of several seconds to minutes. These isotopes could be produced by interactions of charged particles with either aluminum or the silicon, copper, magnesium or chromium used in the 6061 alloy [14], or other materials in close proximity to the board. More work is planned on correlation of error rates with known particle fluxes.

## 7. CONCLUSIONS

The performance of the IDT-3081 demonstrates that it is possible to increase the reliability of a COTS system operating in the space environment. This is illustrated by the improvement in the reliability provided by the ECC routines. The error detection and correction software techniques are not able to correct all errors on the IDT-3081 board, resulting in more crashes than the RH-3000 board experienced. The RH-3000, although it performed well for a first unit, was also hampered by this. One of the goals of the project was to use the boards to process X-ray data in flight. Because of the speed limitations imposed on the RH-3000 by the declocking and the inability to use the cache memory it was not able to process data fast enough to keep up with the sensor output. A second generation of the RH-3000 chip set has since been produced and our information is that the deficiencies that limited its use on ARGOS have been corrected.

In general, there are four classes of uncorrected errors: assertions, timeouts, exceptions, and reboots. Assertions occur when a test application completes its task and reports no internal errors, but produces incorrect results. This class of errors is fairly benign. Although the results are incorrect, the application continues to run, and more importantly, the rest of the system continues to run. Timeouts occur when a test application does not report completion of its task within a reasonable amount of time. Again, this class of critical error is not necessarily catastrophic. Usually, the rest of the system continues to run normally, but any results that the test application may have been working on are lost. Exceptions are those undetected errors causing the processor to encounter a potentially illegal hardware operation. Examples of exceptions are illegal processor instruction opcodes, unaligned addresses, and out-of-bounds addresses. The fourth class of uncorrected error is the automatic reboot. These occur when the processor is running applications and no apparent error is reported, but suddenly the processor reboots itself. Obviously, the reboots are severe since they halt the currently running application set and cause all of the results to be lost.

The total system availability of the IDT-3018 board can best be expressed by specifying the average time the board was capable of running before it required a restart. This was about twelve days when the board was operating with the three test applications plus the ECC, a significant improvement from the two or three days that it would run at the start of the experiment. The restarts were due to either a system crash or the accumulation of sufficient system exceptions to require a reboot. One of the drawbacks to using a proprietary operating system such as VxWorks is that it is difficult or impossible to add capability to clean up after system exceptions, and often the only recourse is to restart.

The RAD-hard board was the more reliable of the two, but it would not have been able to do the data processing job intended for it. The COTS board had more failures, but many of these failures were detected by the fault tolerance software while most of the system was still functioning, and could potentially be ameliorated by more sophisticated techniques. The ARGOS Testbed demonstrates that fault detection software works and points the direction of future investigation to more robust system recovery techniques.

## 8. ACKNOWLEDGEMENTS

The authors wish to thank Mr. Andrew Fox for supplying the RH-3000 board and Dr. Alan Ross and Mr. Kenneth Clark for supplying the IDT-3081 board. The authors also wish to thank Mr. Louis Lome and Dr. Richard Bleach for their sustained encouragement of the program from its inception and for many constructive suggestions.

## 9. REFERENCES

[1] Ziegler, J. F., et al., *IBM J.Res. Devlop.*, Vol. 40, No. 1, (all articles), Jan. 1996.

[2] LaBel, K.A., et al., "Commercial Microelectronics Technologies for Applications in the Satellite Radiation Environment," *IEEE Aerospace Applications Conf.*, Vol. 1, pp. 375-390, 1996.

[3] Shirvani, P. and et. al., "Software-Implemented Hardware Fault Tolerance Experiments; COTS in Space," International Conference on Dependable Systems and Networks, Fast Abstracts, pp. B56-7, June 25-28, 2000.

[4] Nahmsuk Oh, "Software Implemented Hardware Fault Tolerance," Ph.D. thesis, Stanford University, Dec. 2000.

[5] Shirvani, P.P., "Fault-Tolerant Computing for Radiation Environments," *CRC-TR 01-6* (Ph.D. Thesis), Stanford University, Stanford, CA, June, 2001.

[6] The USA X-ray Timing Experiment, P. S. Ray, K.S. Wood, G. Fritz, P. Hertz, M. Kowalski, W.N. Johnson, M.N. Lovellette, M.T. Wolff, D. Yentis, R. M. Bandyopadhyay, E.D. Bloom, B. Giebels, G.Godfrey, K. Reilly, P. Saz Parkinson, G. Shabad, P. Michelson, M. Roberts, D.A. Leahy, L. Cominsky, J. Scargle, J. Beall, D. Chakrabarty, Y. Kim, *Proceedings of X-ray Astronomy 1999, Bologna, Italy*

[7] ICD-907A P91-1/Unconventional Stellar Aspect (USA) Interface Control Document, Rockwell Aerospace, 1996.

[8] Dale, C.J., et al., "Fiber Optic Data Bus Space Experiment on Board the Microelectronics and Photonics Test Bed (MPTB)," *Proc. of the SPIE – The Intr'l Society for Optical Eng.*, Vol. 2482, pp. 285-293, April 1995.

[9] Regeon, P., Lynn, P., Johnson, M., and Chapman,J., "The Clementine Lunar Orbiter," 20th International Symposium on Space Technology and Science, May 19-25, 1996.

[10] Shirvani, P.P., N. Saxena and E.J. McCluskey, "Software-Implemented EDAC Protection Against SEUs," IEEE Transactions on Reliability, Special Section on Fault-Tolerant VLSI Systems, Vol. 49, No. 3, pp. 273-284, Sep. 2000.

[11] Oh, N., P.P. Shirvani and E.J. McCluskey, "Error Detection by Duplicated Instructions In Super-scalar Processors," to appear in *IEEE Transactions on Reliability* Sep. 2001.

[12] Oh, N., P.P. Shirvani and E.J. McCluskey, "Control Flow Checking by Software Signatures," to appear in *IEEE Transactions on Reliability* Sep. 2001.

[13] Harris private communication

[14] Erik Oberg, Franklin D. jones, Holbrook L. Horton, and Henry H. Ryffell, *26th Edition Machinery's Handbook*, New York: Industrial Press, 2000.

Author Biography: Dr. Lovellette is the Project Scientist for the NRL-801 experiment on ARGOS and served as the principal experiment spokesperson for all NRL experiments during ARGOS systems test at Boeing. He is currently involved with GLAST on-board processing and integration and test.