TREMOR: A TRIPLE MODULAR REDUNDANT FLIGHT COMPUTER AND FAULT-TOLERANCE TESTBED FOR THE WPI PANSAT NANOSATELLITE

Ryan Angilly ECE B.S.
Department of Electrical and Computer Engineering
Worcester Polytechnic Institute, Worcester, Massachusetts
Advised by Professor William Michalson

Abstract
Elevated levels of radiation in Low Earth Orbit (LEO) can cause several unexpected behaviors in digital logic. These behaviors, known as Single Event Effects (SEEs), manifest themselves in two ways: unexpected short circuits (Single Event Latch Ups), and erroneous bit flips (Single Event Upsets). Protecting memory from SEEs is usually done via some type of SECDEC controller, and protecting IO can be done in a number of ways -- the simplest of which entails using upper level protocols to verify data integrity. Several techniques are currently employed to deal with SEEs in microprocessors including radiation hardening, radiation shielding, software redundancy, and hardware redundancy. TREMOR uses a hardware solution based on an architecture known as Triple Modular Redundancy to achieve SEE tolerance. This paper discusses the TREMOR FPGA system and how it will be used to synchronize the processors and ensure that no erroneous data propagates to the system bus. It will also discuss how the flexibility of this design will allow TREMOR to become a new test bed for various implementations of the TMR architecture.

TABLE OF CONTENTS

1. INTRODUCTION

Early last year, WPI received a grant from NASA and the Air Force Research Labs Space Vehicles Directorate (AFRL) to participate in the third NanoSat Competition (NS-3). The grant called for building a nanosatellite – no more than 45cm tall and 45cm in diameter – on a fiscal budget of $100,000 and a time frame of two years. WPI was given this grant for three reasons. First was the promise to research the possibility of using powder metallurgy techniques for manufacturing spacecraft. Second was the desire to design a system capable of calculating spacecraft orientation based on multiple GPS measurements over a short baseline (~30cm). Third was the creation of a flight computer impervious to SEEs and built from Commercial Off The Shelf (COTS) parts. Out of this third objective, the **TR**ipl**E** **MO**dular **R**edundant Flight Computer, TREMOR, was born [1].

Since the satellite will be traveling in a Low Earth Orbit (LEO) of ~380km, elevated levels of radiation caused by the lack of atmosphere increase the probability of Single Event Effects (SEE). SEEs come in two types: Single Event Upsets (SEU) and Single Event Latch-Ups (SEL). In an SEU, a high-energy particle travels through a p-n junction in such a way that it causes electron-hole pairs to form around the junction and results in a temporary current flow across the junction. If

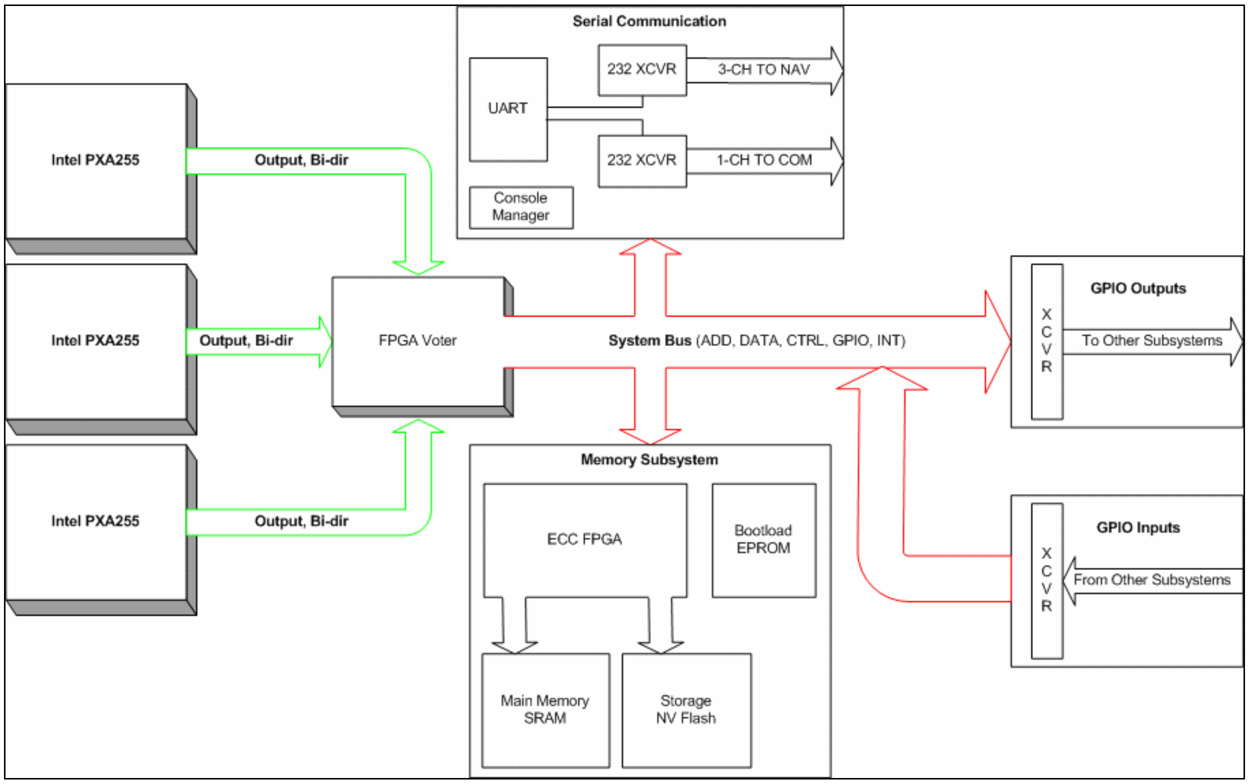


Figure 1-TREMOR Flight Computer Block Diagram

the current flow is large enough, this can manifest itself as a bit value flipping from a logical high to a logical low or visa-versa. In an SEL, a high-energy particle travels through a p-n junction much the same as an SEU. However, if the particle travels through a NAND gate (PNPN), it is possible for a self-amplifying short circuit to occur, causing increased power consumption, heat dissipation, and the possibility of permanent damage to digital circuitry. Three factors are directly related to the probability of SEEs:

1. Element Size: As transistors get smaller, they become more susceptible to SEEs as the amount of charge in the device decreases.
2. Clock Frequency: As clock frequency increases, small perturbations in transistor charge can translate into complete state changes over a short clock period.
3. Cross-Sectional Area: Since transistors are susceptible to SEEs, a larger area of transistors will result in a high probability of SEEs [2].

Another effect of radiation is Total Ionizing Dose (TID). Unlike SEEs, the TID is a rating that denotes how much radiation a device can accumulate before total failure. The TID for the simplest device is usually around 10krads, and the radiation experienced in LEO is around 5krads per year. Since the TREMOR mission is only 6 months, there is effectively no need to worry about TID affecting the parts of the flight computer [3].

Statistics have shown that at 380km, SEEs can be expected at a rate of one every 15 minutes on an average flight computer. Therefore, in order to create a flight computer that can claim it is impervious to SEEs, the memory subsystem and processing subsystem must be protected to ensure that SEEs are detected before they are allowed to propagate and risk danger to the system. The flight computer must also be able to correct these errors before the next SEE occurs.

2. TREMOR FPGA DESIGN

In the TMR architecture, three identical processors operate in lockstep – executing the same instruction at the same time – and instead of outputting their signals unabated to the system bus as in normal uniprocessor systems, the processors output to an arbiter that compares the outputs of each processor to make sure that radiation has not caused bits to erroneously flip either in the processor cache or in the pipelines of the processor.

Figure 1 is a block diagram of the TREMOR flight computer, which shows how the Intel PXA255 processors are completely isolated from the rest of the system via the TREMOR FPGA System. All of the functionality of the TREMOR FPGA comes from several state machines, which govern how it interacts with the rest of the flight computer. These state machines are:

1. TMR Voter State Machine (TMR VSM): Responsible for voting on processor outputs, sending data to the system bus, resetting processors that have been affected by radiation, and entering the Dual Modular Redundant Voter State Machine when an SEU occurs.
2. DMR Voter State Machine (DMR VSM): Responsible for voting on the output of two processors when the TMR VSM has detected an SEU. It stays in this state until the processors are ready to be power cycled.
3. Processor Synchronization State Machine (PSSM): Responsible for ensuring the processors are synchronized on power-up so that they can run in lockstep.
4. General Purpose Input State Machine (GSM): Responsible for synchronizing general purpose inputs pins so that changes on the pins arrive at the processors at the same time.
5. Bi-directional Voter State Machine (BVSM): Responsible for voting on data pins when they are outputs, and passing the data through to the processor when they are inputs.
6. Random Error Generator State Machine (REGS): Responsible for helping to prove the viability of TREMOR by introducing errors into the inputs of the various Voting state machines.

The TMR Voter State Machine has several simple logic circuits implemented within it. In order to determine the majority of three binary signals, the circuit in Figure 2 is used as the basis for the TMR VSM.

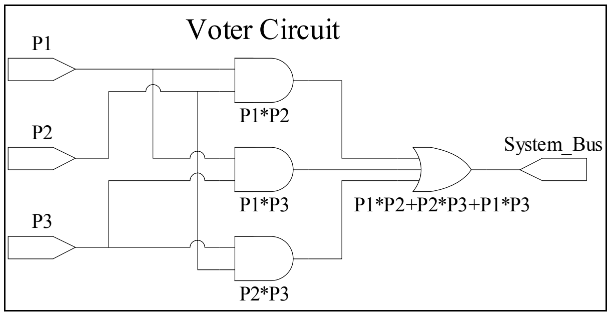


Figure 2-Majority Voter Circuit

In Figure 2, the output is guaranteed to be equal to the majority of the processors, whether all three agree, or if only two agree [4].

The circuit in Figure 3 is used to determine if any of the processors are in disagreement. If a processor is in disagreement, it is reset, and the other processors are notified that it has been reset due to an SEE. These notifications are done through a series of dedicated IO pins. Each processor has three pins as dedicated inputs. If the TREMOR FPGA resets processor 1, then a pin will be driven high on the remaining two processors. This will be detected by software, and a sequence will begin to store the processor states so the system can reset and re-synchronize. This will be discussed more in a later section.

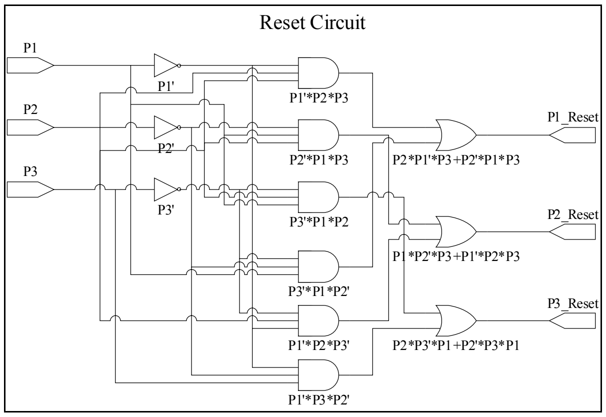
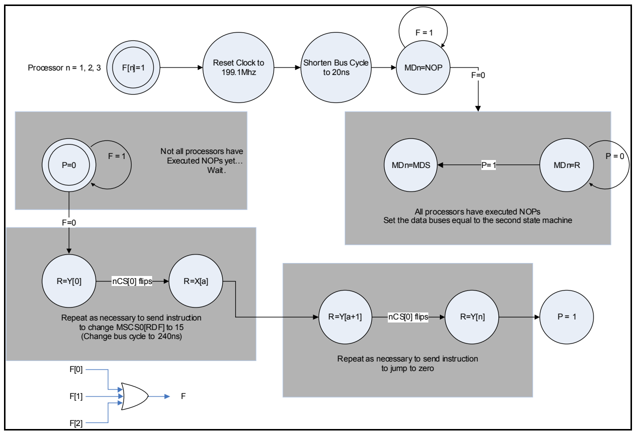


Figure 3- Minority Reset Circuit

When an SEU is detected, the TMR VSM enters a degraded mode called Dual Modular Redundant Voting. In the DMR VSM, the voting is done in a similar way, except only two values need to be compared. The two values are passed through a logical AND, and the value is passed immediately to the system bus. If the two values do not agree, then an SEU has occurred, and the system needs to be immediately power cycled. Once the operating system has finished storing the state of the processors, it notifies the FPGA via a dedicated pin and the DMR VSM cycles power to all processors. Its last job is to start the Processor Synchronization State Machine (PSSM) once power is restored.

The creation of the PSSM was due to a 100.02 microsecond uncertainty in the time it takes the Intel PXA255 to fetch the first instruction after nRESET is unasserted. After power is applied to the PXA255, a 10ms delay must be created before taking the processor out of reset to let the internal PLL stabilize to 99.5Mhz. The processor must then receive instructions to change its clock speed to 199.1Mhz. This 199.1Mhz clock coupled with the 100.02 microsecond delay means that one processor could be 20,000 clock ticks ahead of another. Since TMR architecture requires that the processors run in lockstep, this is obviously unacceptable. The answer was the creation of the Processor Synchronization State Machine (PSSM). Figure 4 is a top level representation of the PSSM [5].


Processor n = 1, 2, 3
F[n]=1
Reset Clock to 199.1Mhz
Shorten Bus Cycle to 20ns
MDn=NOP
F = 1
F=0
P=0
F = 1
Not all processors have Executed NOPs yet... Wait.
MDn=MDS
P=1
MDn=R
P = 0
All processors have executed NOPs
Set the data buses equal to the second state machine
F=0
R=Y[0]
nCS[0] flips
R=X[a]
Repeat as necessary to send instruction to change MSCS0[RDF] to 15 (Change bus cycle to 240ns)
R=Y[a+1]
nCS[0] flips
R=Y[n]
P = 1
Repeat as necessary to send instruction to jump to zero
F[0]
F[1]
F[2]
F
Figure 4-Processor Synchronization State Machine

There are two stages to the PSSM. First, there are three asynchronous instances of the PSSM, one for each processor. The first stage is responsible for getting the processor into a state where it is ready to be synchronized. The PSSM starts immediately after the nRESET signal is unasserted by the FPGA and waits until the processor tries to fetch the first instruction. Instead of voting on the outputs of all three processors, each instance of the PSSM sends the PXA255 several instructions that are hard coded into the state machine. First, the PSSM sends instructions to change the clock speed to 199.1Mhz. Next, it waits 10ms for the PLL to stabilize [6].

The default bus cycle time of the PXA255 is 250ns. In order to get the processors synchronized below 250ns, the next state of the PSSM sends instructions to change the bus cycle time of the processor to 20ns. It then starts to send the processor NOPs, and sets a flag telling the other two instances of the PSSM that it is now in a loop sending the processor a NOP every 20ns.

Once all three instances of the PSSM are in a NOP loop, we know that the processors are synchronized to within 20ns of one another. At this point, the three PSSMs synchronize and start sending instructions to all three processor in unison. In this second stage, the processors are sent instructions to change their bus cycle times to the most efficient value (currently 180ns) and jump to zero. After the processors are instructed to jump to zero, the TMR VSM starts, the next fetch is voted on, and the processors begin operation.

Even though the processors are synchronized by the PSSM, there is still an uncertainty of 20ns. This could be a problem when programming the general-purpose input pins as interrupts. Figure 5 illustrates the problem.

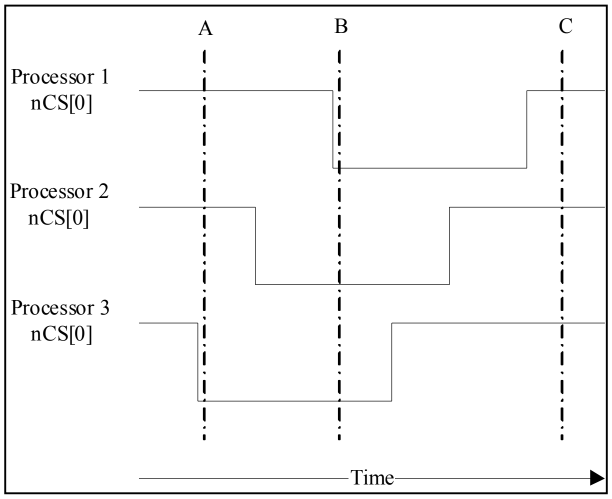


Figure 5- GPI Timing Problem

When the processor enters a bus cycle, it will defer servicing interrupts until the bus cycle is complete. If an interrupt were to reach the processors at Time A in Figure 5, the lagging processors would begin to service the interrupts and would abandon the bus cycle they have yet to start. Meanwhile the third processor would remain in the bus access and would result in the TREMOR FPGA interpreting the situation as an SEE. To avoid this situation, the General Purpose Input State Machine (GSM) was created. The GSM does not let the inputs change at the processor until all three processors are in a bus cycle – Time B in Figure 5. This not only ensures that processors will service the interrupt, but it also ensures that the interrupt will be serviced at the same point in the instruction queue for each processor. This is an added benefit due to pipeline questions that were raised late in the design. If the interrupts were to come in at Time C on Figure 5, the processors would service the interrupt and there would be no SEE. The cache of the PXA255 can be setup to use a write-through caching scheme. This increases the number of bus cycles as the cache tries to remain coherent with main memory.

However, at Time C, the processors, although being synchronized to within 20ns, are still operating with a 4-instruction margin of error. At Time C, all processors would service the interrupt. They would execute the interrupt, fetch instructions, and alter the caches. The caches would remain coherent between the processors, and there would be no problem – until the processors returned from the interrupt. If, in the course of operation, the interrupt had requested an address that replaced Cache Line A from Figure 6, processor 1 would come back, not find the instruction in cache, and try to fetch it from memory. This would be interpreted by the TREMOR FPGA as an SEU and the system would enter degraded mode immediately.

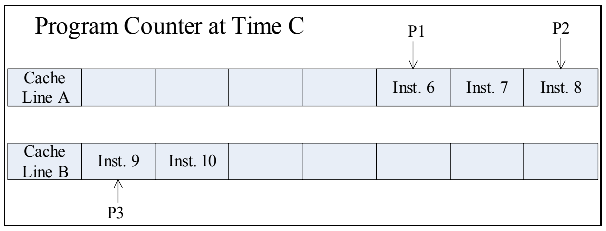


Figure 6- Cache Page Example

There is an interesting tradeoff associated with the behavior that was just discussed. If it turns out that if the interrupt service latency associated with this scheme turns out to be too much, we can allow interrupts to occur during times of no bus access with the understanding that an SEE could occur. The system would service the interrupt immediately, and there would be no added latency. If an SEE did occur, it would be corrected with no harm done to the system and the flight computer would only suffer a small performance degradation. Luckily, the TREMOR system will be able to test both of these scenarios with a simple reprogramming of the FPGA.

The processor not only has to vote on outputs, but it also has to vote on bi-directional pins like the data bus. In order to accommodate this special case, a version of the VSM was created that used the output from the RDnWR voter to detect if the processor is reading or writing. The RDnWR pin on the PXA255 reflects what state the data bus is in – whether it is reading or writing. The Bi-directional VSM (BVSM) then enables and disables the appropriate tri-state buffers based on the outputs of that state machine. Figure 7 is a schematic of the BVSM.

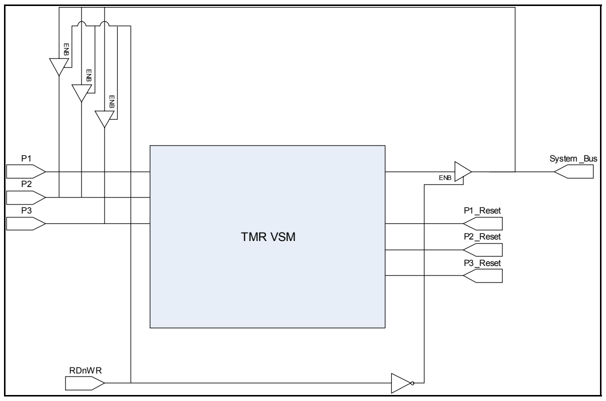


Figure 7-Bidirectional VSM

The timing of the RDnWR signal is such that we can use it as an input to this voter with no problems. Any changes in the direction of the data bus are reflected in RDnWR well before the changes actually take place.

The Random Error Generator State Machine (REGS) is not vital to the operation of TREMOR, but it is a valuable machine to help aide in debugging the system. REGS will be able to randomly modify input pins to the various voter state machines at regular intervals to verify the stability of TREMOR in radiation environments.

4. TREMOR/OS INTERACTION

TREMOR is currently being slated to run a release of RTLinux. There are several instances where the Operating System needs to interact with the TREMOR FPGA.

When an SEU occurs, the TREMOR FPGA notifies the processors by driving a pin high. This pin is configured to be an interrupt on all of the processors. When the signal is driven high, the operating system saves all the processor states and process contexts in FLASH. It also sets some bits in FLASH to let itself know on reset that there are processes waiting to be restored. Once all the processes and states are saved, the operating system sends a signal to the FPGA that it is ready to be power cycled. Power is removed from the processors, the processors are cycled, they run through the PSSM, and begin executing the bootloader software at the reset vector.

The bootloader is responsible for uncompressing the kernel from the PROM and storing it in main memory. It is then responsible for jumping to the start of the operating system. However, it also checks to see if the system was reset due to an SEE. If the system was reset to the SEE, it finds the processor states and process contexts in the FLASH and restores them. Since power is not cycled to the memory, there is no need to copy the kernal from PROM to main memory again. In fact, the operating system will be able to pick up exactly where it left off. To the operating system, it will appear as if the following psuedocode was executed:

```
1:  save_processor_state();
2:  save_process_contexts();
3:  int pc = read_program_counter();
4:  pc = modify_and_store_pc(pc);
5:  notify_FPGA_ready();
6:  /* Power is cycled */
7:  iret();
```

Lines 1 and 2 save the state of the system. Line 3 reads in the current program counter. Line 4 modifies the program counter before it is stored so that when it is popped out of FLASH it will lead right to line 7 and allow the interrupt to return to wherever it was called from. As soon as line 5 is executed, power is cycled to the processors. The bootloaded software, after realizing there is a context to be restored, is responsible for manually restoring all register values, general purpose IO pin directions, interrupt pin assignments, power management registers, and clock management registers. It will then send the processors an instruction to jump to PC and send it to line 7 in the psuedocode.

5. TREMOR Current Status

From the beginning, the TREMOR project was destined to be a multi-year design project. To date, the current TREMOR team has accomplished the following:

Schematic Completion: All the necessary schematics have been completed.

Part selection: All of the parts for TREMOR have been selected, and a full compatibilty analysis has been completed. This includes voltage compatiability, fan-out analysis, loading compatibility, and timing analysis.

PANSAT Interfaces Defined: The interfaces to all other PANSAT systems have been defined.

FPGA Operational Outline: Although nothing can be considered complete until a working model has been built, the TREMOR team believes that the foundation laid for the TREMOR FPGA system is a strong one. Extensive failure modes analysis has led to a system that can deal with voting on output pins and their failure modes; voting on bidirectional pins; synchronizing the processors to within acceptable parameters; buffering inputs such that any changes arrive at the processors at the same point in the instruction stream; and communicating state information across FPGAs, since the TREMOR FPGA system is too large for a single FPGA.

Extensive Docmentation: The Configuration Management document tree that was modified to create this MQP follows the WPI PANSAT Configuration Management plan and will allow for a smooth transition to the next team.

6. TREMOR Future Work

The flight competition review is schedule for January 2005. By that time, TREMOR must be operational. This section outlines what is left for TREMOR to be considered complete.

Finalizing Power Delivery System: The FPGA must have the ability to control the power to the processors in much the same way that the Power Subsystem must have the ability to control the power to the entire flight computer. These power relays must be selected, either by the TREMOR team independently, or by the Power team, before layout can be finalized.

PCB Layout: The PCB need to be layed out, ordered and populated.

Radiation Tests: WPI's nuclear reactor is capable of recreating the orbital environment for TREMOR. A finish product can be irradiated by the reactor in an attempt to detect SEEs and see if TREMOR behaves as expected.

7. Conclusions

The TREMOR flight computer will allow the software of WPIs PANSAT Nanosatellite to operate completely oblivious to radiation effects. By the end of this project, TREMOR will have proven itself as a robust solution to the radiation effects problem and will act as a test-bed for future researchers to explore radiation tolerance in their labs.

8. Acknowledgements

I would like to thank my teammates, Daniel DeBiasio and Mitchell Lauer, for their dedicated hard work in this project. I would also like to thanks Ritesh Shuklah for his help in evolving TREMOR from a simple flight computer to a fault-tolerance test bed. Lastly, I would like to thank Professor William Michalson for his guidance through the project, and on this paper.

9. References

[1] Air Force Research Laboratory, "AFRL Internal Cargo Unit User's Guide," Space Vehicles Directorate, Kirtland AFB, NM, 2003.

[2] P. Shirvani, "Fault-Tolerant Computing for Radiation Environment," Center for Reliable Computing at Stanford University, 1997.

[3] M.N. Lovellette, "Strategies for Fault-Tolerant, Space-Based Computing: Lessons Learned from the ARGOS Testbed."

[4] S. Mitra, E. McCluskey, "Word-Voter: A New Voter Design for Triple Modular Redundant Systems," Center for Reliable Computing at Stanford University, 2000.

[5] "Intel® PXA255 Processor Electrical, Mechanical, and Thermal Specification," Intel Corporation, Hudson, MA, March 2003.

[6] "Intel® XScale Microarchitecture for the PXA255 Processor," Intel Corporation, Hudson, MA, March 2003.