Simple Methods for Error Detection and Correction for Low-Cost Nano Satellites

Kjell Arne Ødegaard, Amund Skavhaug

Simple Methods for Error Detection and Correction for Low-Cost Nano Satellites

Kjell Arne Ødegaard and Amund Skavhaug

Department of Engineering Cybernetics,
Norwegian University of Science and Technology,
Trondheim, Norway
kjell.arne@odegaard.net, amund.skavhaug@itk.ntnu.no

Abstract. The objective of this paper is to propose a low-cost, robust Error Detection And Correction (EDAC) solution for use in applications such as nano satellites, where price is a primary concern. Different methods have been evaluated, with the main result mitigation Single Event Effects causing bit-flips in system memory utilizing Bose-Chaudhuri-Hocquenghem (BCH) codes. The general implementation is resource intensive and the algorithm has been adapted to the embedded platform. The codes have been implemented on a low-cost microcontroller with a real time operating system and faults have been injected during run-time to emulate a radiation environment. The performance impact and dynamic behavior of the algorithms is studied with third party trace analysis tools.

1 Introduction

The gateway to space for research institutions and commercial actors has traditionally been associated with a very high cost. Recent year's development of small, inexpensive satellites known as pico and nano satellites can change this by considerably lowering both the price point of satellite construction and launch.

An interesting development along these lines has been the introduction of the CubeSat platform. To help universities worldwide perform space research the CubeSat platform was developed in 1999 by, among others, California Polytechnic State University and Stanford University. The CubeSat programs goal is to provide practical, cost-effective and reliable launch opportunities for small satellites and their payloads through a standardized platform measuring form $10 * 10 * 10\ cm$ to $10 * 10 * 30\ cm$ [10] [13] [14]. The community also maintains an overview of available launch providers, including contact information, a service that simplifies launch tremendously.

The small standardized form factor makes it more feasible to combine the CubeSats with other payloads, keeping the launch costs low. The co-launch with other payloads is facilitated in the CubeSat standard by providing pre-authorized specifications for materials, physical launch stress and separation of satellite and launch vehicle in orbit.In addition, the satellites often use Commercial of-the-shelf (COTS) electronic components, further decreasing satellite costs.

This paper aims to investigate low-cost methods to increase mission lifetime of small COTS based satellites. The theory and methods that are used are well known, but the application is novel. The CubeSat community is composed of a large number

of universities, private firms and even high schools [13]. One of the primary goals with CubeSats is to provide an educational platform. A consequence of this is that the teams working on the satellites have varying degrees of competence, and a robust *design* becomes even more important. The StudSat project at NTNU started as far back as the early 2000s [9] and have launched two satellites. The first exploded during launch and communication was never achieved with the second satellite. This history clearly states the concern both for low cost and dependability for the current satellite.

The use of COTS based solutions allows for fast development with modern tools and enables the designers to get full advantage of the economy of scale with cheap and plentiful components and development tools. Due to the typically shorter lifespan of these satellites compared to traditional endeavors, it is possible to use newer, more innovative and even unproven components and designs without running unacceptable financial risks. This is interesting as it allows for rapid development and advancement in an otherwise conservative industry.

The majority of the reported work in this paper has been to study the satellite and its systems, as well as suggesting solutions to the problems that are likely to be encountered. Due to cost concerns, availability and needed simplicity due to students, CubesSats [13] [14] are usually based on the use of COTS components. A number of different factors, that will be detailed later in this paper make these components vulnerable to the environment in space. In this paper we explore measures to alleviate the impact of these factors to the reliability, availability and survivability of the satellite.

Fig. 1. NUTS - NTNU Test Satellite

1.1 Problem

One of the main challenges for space applications is the hard radiation operating conditions [3] [5]. Radiation hardened electronic components and fault tolerant hardware have been used in space systems for a number of years to either ensure error free operation or to mask the occurrence of errors from the operation of the system. In the context

of a CubeSat, however, the main challenges of high reliability system design are slightly different. It is still desirable with a high reliability system, but the budgetary constraints are much stricter than for commercial or government satellites.

In addition to being considerably more expensive, radiation hardened components traditionally lag behind their non-hardened equivalents in performance. This means that one gets a less capable system at a higher price point. At he same time, high availability design for CubeSats is usually not so important since the system does not control critical applications, but rather performs data collection tasks of an exploratory nature. It is important to receive correct data and to know if the satellite has suffered a malfunction, but the timeliness is of less importance. This means that on line redundant backup components can be omitted as long as we ensure that the system does not malfunction critically (i.e. fail without coming back up again). By using software methods, combined with some *simple* measures of redundancy for the most important subsystems, it is therefore possible to get higher performance, more flexibility and lower price, all without hot standby redundant backup components. The reason for this software approach is twofold. The most important systems in the NTNU Test Satellite (NUTS) have already been realized in hardware, and a redesign at such a late stage is not desired by the project management. The second reason is that we want the proposed solutions to be relevant for projects that do not have the resources to build a conventional high reliability system.

When considering the different reliability measurements it is important not to impact the performance of the rest of the system to an unacceptable degree. If we can accept restarts and possible data loss when mitigating the effects of Single Event Phenomena (SEP), it is possible to mask the errors from the operation of the system by power cycling, checkpointing and Error Detection and Correction (EDAC). Power cycling implemented in the power supply and backplane logic clears Single Event Latchups (SEL) form components while checkpointing and EDAC clears faults from Single Event Upsets (SEU) in memory. This further promotes safe operation and increases the likelihood of not loosing mission critical or payload data. Student satellites do not have access to the established solutions because of budget constraints, and have to rely on ingenious solutions and COTS hardware to have a usable system even in extreme conditions.

The problems with COTS components in space are numerous, as detailed by NASA [3]. In brief, radiation effects known as SEP, can occur when cosmic radiation strikes certain parts of the semiconductor material, as outlined by Fig. 2. If the cosmic ray has enough energy it can alter the electrical charge and thereby alter the digital value in the component. This is known as a bit-flip and can corrupt saved data in addition to causing instability in the system. The expected number of errors estimated by NASA is $10^{-5}\ errors/bit-day$ [3]. For NUTS [11] [12] this results in hundreds of errors per day in RAM and and up to a thousand errors per day in the flash data-banks. The expected radiation level is 10-100 Gy per gram of silicon per year with an orbital inclination between 20 and 85 degrees [3]. The large variance stems from the fluctuations in the solar cycle which determines the flux of both solar and galactic radiation. NUTS will have an even higher inclination and therefore even higher worst case radiation levels can be expected. With the total dose failure level of flash memories from 50-150 Gy

and microprocessors from 150-700 Gy [3] of radiation, both can experience failure of a permanent nature during the satellite's mission lifetime.

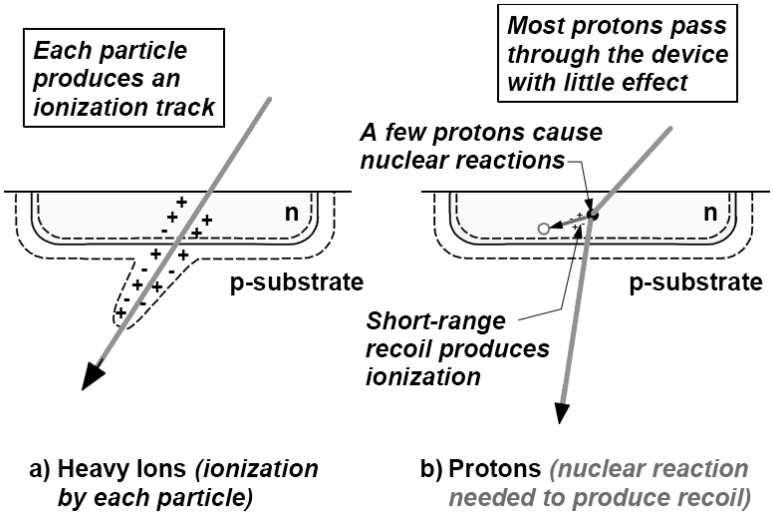

Fig. 2. Mechanisms for Heavy Ion and Proton SEU effects [5]

2 Error Detection and Correction, EDAC

Due to the random nature of the expected fault it is difficult to determine if the data variables are safe to use. To counteract faults we could store the variables multiple times and do a majority voting on the correctness or have an error correcting algorithm such as BCH [2, p. 155] codes to correct the faults at run-time. Since executing BCH codes in an individual task adds a layer of complexity, we have implemented a system task in order to manage the secure storage and recovery of protected data.

A specialized EDAC system task with practical interface functions eases the development by removing the sometimes complex algorithms from the other modules. A module based design is also favorable in programming because of the increased ease of maintaining and ensuring the correctness of smaller modules. This point applies even more for reliable systems [1, p. 202].

3 Checkpointing

Checkpointing is a proven solution in software system redundancy. This enables the system to roll back in the case of an error or initialize quickly and without losing critical

data in the event of a system restart [7]. It is important to ensure that the system is able to roll back multiple instances in case there is some unforeseen fault present.

Power cycling of faulty modules is implemented in the backplane. The modules of the satellite must therefore tolerate a sudden reset without losing any significant amount of progress or data (i.e. at least the loss of data must be known). It must be known that the reset is due to an error in operation as there are some events such as antennae deployment and detumbling that should only be executed once. Including these events in the saved system state will provide a simple measure of ensuring operational progress for the satellite.

4 Testing

The most realistic test would be to expose the system to a radiation environment and measure how the system holds up under real stress. While this might be desirable for the finished system it is not very useful when testing specific algorithms or sub modules in the system. The reason for this is that it is very difficult to control which modules is to be tested and next to impossible to replicate the exact error conditions in order to determine the severity of the fault.

Another alternative is to simulate random error occurrence via Joint Test Action Group (JTAG) port in the software running on the CPU boards [8]. This is somewhat better because the efficiency, e.g. of the error correcting code, can be determined directly since the number of inserted faults is known. Arguments against this testing regime is the lack of some realistic errors. Latchup, for instance, is hard to simulate in software.

With these considerations in mind, the preferred testing method is to simulate errors with JTAG injection of faults during runtime. This is the most economically viable option for us, while at the same time allowing for repeatable test runs and allowing us to focus on specific parts of the system.

5 Other Methods

In addition to EDAC and Checkpointing, a number of other features are being implemented. Master-Slave functionality allows for a spare control computer in case the main crashes. The Watch Dog Timer (WDT) ensures that the system does not deadlock forever, e.g. while interfacing with other system components. A periodic reset protects against any undetected failures that linger in the system. The ability to disable faulty modules completely (i.e. power down) safeguards against a malfunctioning module affecting the rest of the system. Finally, the ability to perform an integrity check on the program memory makes it possible to detect and possibly restore errors.

6 Scope

The scope of this work is limited to soft and transient faults. If the components malfunction due to effects such as charge distribution, Single Event Latchups or Single Event

Gate Ruptures, the power system and backplane is designed to cycle the power of the components. If components are damaged there are backups for the most important ones, for others the system will operate with reduced functionality.

This paper aims to investigate how to achieve high dependability in a simple system with the use of software methods only. The reason for this approach is twofold: The hardware for the most important systems have already been completed, and a redesign at such a late stage is not desired by the project management. Additionally we want the proposed solutions to be relevant for projects that do not have the resources to build a system with high dependability through conventional means.

7 Experiments

7.1 Functional Overview

Figure 3 shows the components of the system and a brief presentation of functionality is provided in Fig. 4. This is the principal design: The system is assumed to start in a normal state. The system does not, however, assume correct operation, and the first action after startup is to perform a CRC of program memory. If a fault is discovered the EDAC attempts to correct the data. If the error can not be recovered the system enters the checkpoint stages $(c_1, c_2, \ \ldots, c_n)$. If the rollback is successful the system continues, if not it resets.

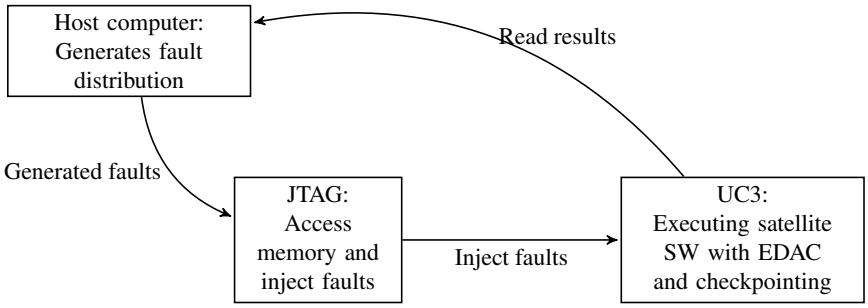

Fig. 3. Test Environment

7.2 Test Environment

The experimental systems consists of a host computer and an *Xplained* development board [15] from Atmel. The development board uses the AT32UC3-A3256 microcontroller [16], the same microcontroller as the NTNU satellite. The Xplained executes the EDAC and checkpointing system and two tasks that requests protected memory from the EDAC system. The software for the Xplained have been developed on Atmel Studio

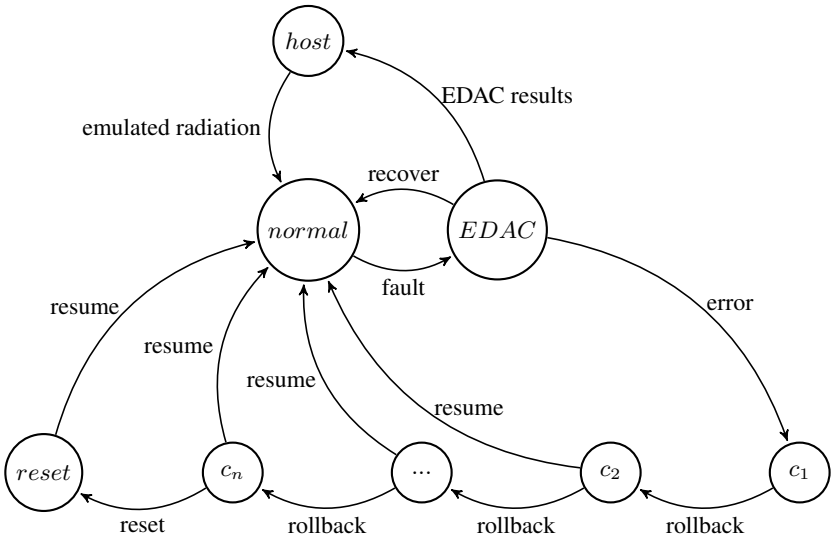

Fig. 4. Functionality Description

6.0.1996 with the AVRGCC 3.4.1.95 tool chain. We use the Atmel Software Framework (ASF) to provide drivers for external components and a protocol stack for communication between the host computer and the Xplained board. The operating system used is FreeRTOS 7.0.0. The host computer generates errors in a certain distribution to emulate radiation and injects these through a JTAG interface while monitoring the EDAC results.

In order to have more control of the results we have configured a representative test system. The representative code only includes the necessary components (FreeRTOS, ASF and BCH codes). This way we have the desired control of the execution environment. One reason for the necessity of this is that the code for the full satellite system is written by many individuals and due to its size it is difficult to maintain a comprehensive overview of all occurring events.

The main satellite repositories have 23405 lines of C and assembly code. The development environment for the representative test system have 18036 lines of code consisting mainly of operating system and drivers. The difference between the satellite repositories and the representative system is approximately 5400 lines of code. The implementation of EDAC and checkpointing adds approximately 2700 lines. It is significantly easier to control the representative system, since the omitted lines are continuously changing and perhaps not structured optimally having been written by students and not professional programmers.

The microcontroller has limited RAM to store the protected data. To compensate for this, and leave a bigger portion of system memory to tasks such as image compression, we store most of the protected data in flash memory. When the variables are requested they are loaded from flash to RAM. The protected data in the flash is corrected pe-

riodically. To communicate between tasks on the microcontroller we use the built in queues in FreeRTOS. In the representative test system we protect a smaller amount of data compared to the requirements of the finished satellite. To compensate for this we increase the intensity of the emulated radiation. The emulated error distribution of the protected data is generated and transferred to the microcontroller.

The result from use of the error correcting code is in Tab. 1. The protected memory is divided in blocks of 1008 bytes as this is the best fit between an even number of 9 byte BCH codes and the flash page size of 1024 bytes. Table 1 presents the results from the correction of three blocks of memory. The faults are generated as a normal distribution and injected. The faults that can not be corrected leads to errors. In Tab. 2 the number of faults per BCH code entry is bounded to the maximum correctional capability of the code. When the number of faults increase past 224 we cross this threshold. Table 2 is included to demonstrate the maximum effectiveness of the correctional codes under ideal circumstances.

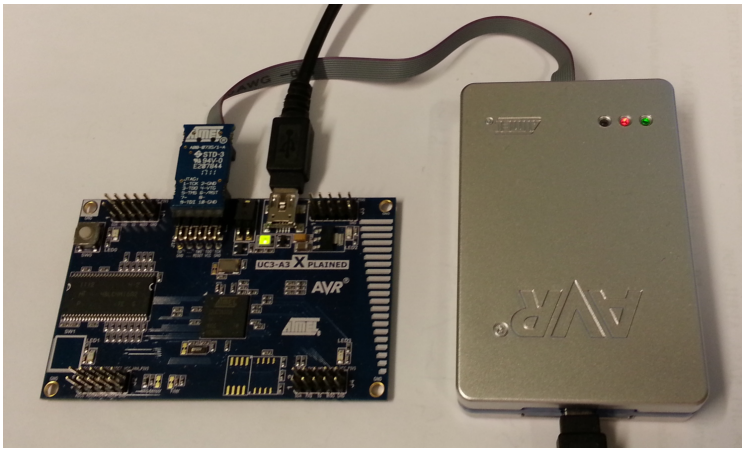

Fig. 5. UC3-A3 Xplained and JTAG ICE3

8 Results and Observations

The preliminary results are encouraging. The chosen parameters can detect and correct up to 2 randomly occurring errors per stored variable, and if the faults are located favorably, up to 224 errors per protected data block. Upon a closer examination of the injected errors we observe that the system runs to completion if the number of errors per message block is lower or equal to the number of errors the BCH codes can correct. Reed-Solomon (RS) error correction might have been a better choice since they perform better than BCH codes in burst error cases [2, p. 113].Nevertheless, with the expected fault intensity of $10^{-5} errors/bit - day$ it is very unlikely that the number of errors in a message block will exceed the codes' capacity. The decoding of RS and BCH has

Table 1. Error correction

Block 1		Block 2		Block 3		Error Comparison		Fault Generator		
Faults	Errors	Faults	Errors	Faults	Errors	Mean	Std.Dev	μ	σ	seed
32	0	28	0	36	0	0	0	0.3	0.5	13585
69	0	63	0	65	0	0	0	0.6	0.5	13585
132	0	128	0	128	0	0	0	1.2	0.5	13585
163	0	152	0	156	0	0	0	1.4	0.5	13585
169	0	161	1	165	0	0.33	0.47	1.5	0.5	13585
181	0	175	3	177	0	1	1.41	1.6	0.5	13585
191	4	195	5	188	3	4	0.82	1.7	0.5	13585
204	8	204	9	200	7	8	0.82	1.8	0.5	13585
210	9	215	11	213	11	10.33	0.94	1.9	0.5	13585
222	13	223	15	229	20	16	2.94	2.0	0.5	13585
235	18	230	30	235	24	24	4.90	2.1	0.5	13585
244	26	240	27	243	27	26.67	0.47	2.2	0.5	13585

Table 2. Error correction for ideal case

Injected faults	Errors
128	0
224	0
256	32

similar performance and as part of future work we could change the implementation to get better results in those cases.

While the general implementation of $BCH(N, K)$ codes is costly and very inefficient [2, p. 161], by taking advantage of specific aspects of the BCH codes and using look-up tables, we have optimized the implementation for our microcontroller. By choosing constant values for N and K we do the heavy computation of the generator polynomial coefficients in advance. With these techniques in place, the number of required cycles can be reduced by up to 51% [2, p. 164], but the precise computational cost may vary with the chosen embedded processor. The typical features that affect performance is word length (32, 16 or 8-bit) and if the processor uses soft float or has floating point processing implemented in hardware. Other factors such as the ability to use specific processor capabilities such as special instructions for digital signal processing can also increase performance.

For testing purposes, optimized $BCH(67, 53)$ codes have been implemented. This code length is used in the European Digital Video Broadcasting standard [17] and it is therefore easier to find hardware implementations if increased performance is required. However, the final parameters should be adjusted based on how much computational power that is available after the payload and radio systems have been fully integrated and tested. This is due to the energy budget. The codes should not run a significant amount of time since the available battery power is limited.

9 Discussion and Conclusion

The main focus of this work has been to study the NTNU satellite to date and present possible solutions to some of the problems. The work aims to implement a more reliable overall system. The bulk of the work has been to understand the satellite's systems and reason which solutions that are most fitting to solve the expected problems.

The different problems are presented together with suggested solutions. Further, it details how these problems can be solved with the constraint of using the already developed satellite systems.

Some of the strategies for low-cost components can be questioned. Why use a low-cost component when the launch cost is very high? But then again these components have the low complexity required to be included in student designs. Even with these low-cost solutions one should remember that a processor that costs $1 today can be more powerful and uses much less power than the processors in the $100-$1000 from 25 years ago. When this is combined with the wide availability of inexpensive sensors, the result is that it is possible to collect much more data at a lower cost than before. For the same reasons it is also possible to deploy redundant sensors, and as with the processors, the inexpensive cameras of today can have far greater capability than those used by NASA in the 70s.

The future work will focus on implementation of the solutions discussed in this paper. As more of the subsystems reach completion they have to be integrated in the scheduling and fault recovery schemes of the satellite. The available processing power will be determined by the system's operating parameters and the load of other tasks such as the compression algorithms. Because of this it is not advantageous to provide a finely tuned system at this point, but rather to focus on a useful module for the satellite being built now. An exhaustive fault injection test to determine how the full system performs under stress is planned as the system reaches completion. With the chosen strategy for protecting code and data in the presence of cosmic rays, using simple methods in software have the possibility of enhancing the dependability significantly.

References

1. Daniel P. Siewiorek and Robert S. Swarz, *Reliable Computer Systems, Design and Evaluation*. Digital Press, Burlington, 2nd Edition, 1992.
2. Hazarathaiah Malepati, *Digital Media Processing, DSP Algorithms Using C*. Newnes, Burlington, 2010.
3. *Space Radiation Effects on Electronic Components in Low-Earth Orbit*, PRACTICE NO. PD-ED-1258, JPL NASA, APRIL 1996.
4. Edward M. Silverman, *Space Environmental Effects on Spacecraft: LOE Materials Selection guide*, NASA Contractor Report 4661 Part 1, 1995.
5. Sammy Kayali, *Space Radiation Effects on Microelectronics*, JPL NASA
6. Emma Litzier, *System Overview - Space Segment*, http://nuts.cubesat.no/the-satellite, 2013.
7. Amund Skavhaug and Odd Pettersen *microFaultTolerant (μFT) - A system for achieving cost effective fault tolerance in microcontroller based equipment*, Real-Time Systems, 1995. Proceedings., Seventh Euromicro Workshop on, Conference Publication, 1995

8. Olof Hannius and Johan Karlsson, *Impact of Soft Errors in a Jet Engine Controller*, Computer Safety, Reliability, and Security Lecture Notes in Computer Science, Volume 7612, Springer, 2012.
9. Jan Tommy Gravdahl, Egil Eide, Amund Skavhaug, K Svartveit, KM Fauske, Fredrik Mietle Indergaard, *Three axis Attitude Determination and Control System for a picosatellite: Design and implementation*, Proceedings of the 54th International Astronautical Congress, 2003.
10. Kjell Arne Ødegaard and Amund Skavhaug *Survey of correction methods for faults and errors induced by cosmic radiation on operating system level in CubeSats*, IAA-CU-13-09-09, 2013, http://nuts.cubesat.no/publications-and-reports.
11. Roger Birkeland and Odd Gutteberg, *Overview of the NUTS CubeSat Project*, IAA-CU-13-09-09, 2013, http://nuts.cubesat.no/publications-and-reports.
12. *NUTS - Publications and reports*, http://nuts.cubesat.no/publications-and-reports
13. Cubesat Specification, http://www.cubesat.org/images/developers/cds_rev12.pdf
14. CubeSat mission statement, http://cubesat.org/index.php/about-us/mission-statement
15. UC3-A3 Xplained, http://www.atmel.com/tools/UC3-A3XPLAINED.aspx
16. AT32UC3A3, http://www.atmel.com/Images/doc32072.pdf
17. Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television, European Broadcasting Union, 2004. http://www.etsi.org/deliver/etsi_en/300700_300799/300744/01.05.01_40/en_300744v010501o.pdf