NTNU – Trondheim
Norwegian University of
Science and Technology

Error Detection and Correction for Low-Cost Nano Satellites

Kjell Arne Ødegaard

Master of Science in Engineering Cybernetics
Submission date: July 2013
Supervisor: Amund Skavhaug, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Error Detection and Correction for Low-Cost Nano Satellites

Kjell Arne Ødegaard

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ENGINEERING CYBERNETICS

July 11, 2013

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.

— C.A.R. Hoare

Problem Description

The objective of the paper is to examine and evaluate the NTNU Test Satellite and propose reliability solutions for a low-cost system based on consumer grade components for the satellite. In order to find suitable solutions, a study of the satellite and its operational environment has been conducted. Based on this study a number of measures are proposed and a subset of the these solutions are implemented. The intention is to lay a solid foundation for future implementation of a dependable system on the NTNU Test Satellite.

Supervisor: Associate Professor Amund Skavhaug, Department of Engineering Cybernetics, NTNU.

Abstract

The objective of this paper is to suggest low-cost measures for dependability and robust Error Detection and Correction 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 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 are studied with third party tools. The Error Correction and Detection should prevent the expected hundreds of errors per day from accumulating in memory and affect the system. The resulting design is expected to be able to mask even frequent resets and errors from the system's operation. Parts of this thesis have been accepted for publication and the papers are included in the appendices.

Sammendrag

Målet med denne oppgaven er å foreslå økonomiske tiltak for pålitelighet og robust feildeteksjon og korrigering for bruk i applikasjoner som nanosatelliter hvor kostnad er svært viktig. Forskjellige metoder har blitt vurdert og hovedresultatet motvirker strålingseffekter som forårsaker feil i minnet ved hjelp av BCH koder. Den generelle implementasjonen til kodene er ineffektiv og den har derfor blitt tilpasset den integrerte datamaskinen. BCH kodene har blitt implementert på en kostnadseffektiv mikrokontroller med et sanntids operativsystem og feil har blitt injisert under systemets operasjon for å emulere strålingen i verdensrommet. Innvirkningen dette har på ytelsen til systemet har blitt studert ved hjelp av tredjeparts verktøy. Feildeteksjonen og korrigeringen er ventet å motvirke de forventede feilene fra å akkumulere i systemminnet og påvirke systemet. Det resulterende designet er ventet å maskere selv hyppige feil og omstarter fra systemets operasjon. Deler at denne rapporten har blitt akseptert for publikasjon og artiklene er inkludert i vedleggene.

Contents

List of Figures

List of Tables

List of abbreviations

ADCS	Attitude Determination and Control System
ASF	Atmel Software Framework
BCH	Bose-Chaudhuri-Hocquenghem
CRC	Cyclic Redundancy Check
CSP	Cubesat Space Protocol
EDAC	Error Detection And Correction
ELDR	Enhanced Low Dose Radiation
EPS	Electrical Power System
ESA	European Space Agency
GCR	Galactic Cosmic Ray
GPIO	General Purpouse Input Output
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit

JTAG	Joint Test Action Group
LEO	Low Earth Orbit
LET	Linear Energy Transfer
MRAM	Magnetoresistive Random-Access Memory
NUTS	NTNU Test Satellite
OBC	On Board Computer
RS	Reed-Solomon
RTC	Real Time Clock
SEB	Single Event Burnout
SEE	Single Event Effect
SEGR	Single Event Gate Rupture
SEL	Single Event Latchup
SEP	Single Event Phenomena
SET	Single Event Transient
SEU	Single Event Upset
SoPC	System on Programmable Chip
WDT	Watch Dog Timer

Chapter 1

Overview

1.1 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 goal of the CubeSat program is to provide practical, cost-effective and reliable launch opportunities for small satellites and their payloads through a standardized platform measuring from $10*10*10cm$ to $10*10*30cm$ [1] [2] [3]. In addition, the community 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 [2]. 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 [4] and have launched two satellites. The first exploded during launch and communication was never achieved with the second satellite. This history clearly illustrates 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 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 [2] [3] 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.

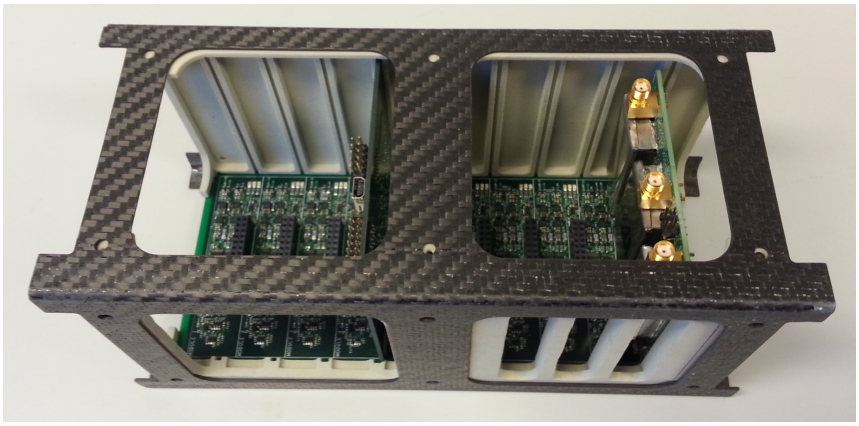


Figure 1.1: NUTS - NTNU Test Satellite

1.2 Problem

One of the main challenges for space applications is the hard radiation operating conditions [5] [6]. 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 com-

ponents traditionally lag behind their non-hardened equivalents in performance. This means that one gets a less capable system at a higher price point. At the same time it is usually not as important with a high availability design for CubeSats 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 recognize 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 NTNU Test Satellite (NUTS) has 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 EDAC. Power cycling implemented in the power supply and backplane logic clears Single Event Latchups (SEL) from 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 losing 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 [5]. In brief, radiation effects known as SEP can occur when cosmic radiation strikes certain parts of the semiconductor material, as outlined in Fig. 1.3. 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 $10^{-5}\ errors/bit - day$ [5]. For the NTNU Test Satellite (NUTS) [7] [8] 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 [5]. 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 [5] of radiation, both can experience failure of a permanent nature during the satellite's mission lifetime.

1.3 Scope and Disposition

1.3.1 Scope

The NUTS project was started in September 2010 with the goal to manufacture and launch a double CubeSat by the end of 2014. There have been a number of published articles and papers concerning the reliability of the system as the project has evolved. The most noticeable ones are the work done on the backplane by Dewald [9], the Electrical Power System (EPS) by Jacobsen [10] and the On Board Computer (OBC) by Holmstrøm [11].

The purpose of this work is to study useful techniques for implementing a reliable system by using the modules that have previously been developed. The approach has been to study reliability theory and find methods that can be used to mitigate the specific problems encountered by the use of COTS components in a space environment. The previous work has focused on reliability for certain aspects of the system, but has not provided software and methods to incorporate this into a dependable system.

The scope of this work is limited to soft and transient faults. If the components malfunction due to effects such as charge distribution, SELs or Single Event Gate Ruptures (SEGR) , 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 has 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.

1.3.2 Report Disposition

This paper will start with a basic introduction on the topics of space, radiation, electronics and redundancy. The reason for this is that the paper is meant to be an introduction for the multidisciplinary project team which might not all be equally versed in the previously mentioned subjects.

The paper continues with a presentation of the current design in NUTS. It touches on the limitations, challenges, requirements and the balancing of reliabil-

ity versus availability before continuing with the current design and its possibilities and limitations with regards to the use of COTS components in space.

The paper concludes with suggested solutions for the presented problems and presents the work done to implement some of these solutions and study the reliability of the system.

Parts of this paper have been accepted for presentation at the 2nd IAA Conference and the 5th European Cubesat Symposium. The paper and presentation is included inn App. B and C. In addition, parts of the paper have been accepted for publication at the SAFECOMP 2013 Dependable Embedded and Cyber-physical Systems (DESC) workshop. This paper is included in App. D.

1.4 Theory

Before we examine the satellite and the suggested dependability solutions, some relevant theory is presented. First some terms from reliability theory, followed by the problems encountered by electronic components in a Low Earth Orbit (LEO) environment and their expected lifetime.

1.4.1 Fault Detection and Manifestation

There are many kinds of problems that can manifest in a system. In reliability theory one differentiates between fault, failure and error. The definitions below are from [12, p. 22]. In figure 1.2, the cause and consequence of different faults are further detailed.

Failure occurs when the delivered service deviates from the specified service, failures are caused by errors.

Error is the manifestation of a fault within a program or data structure; errors can occur some distance from the fault sites.

Fault is an incorrect state of hardware or software resulting from failures of components, physical interference from the environment, operator error, or incorrect design.

Permanent describes a failure or fault that is continuous and stable; in hardware, permanent failures reflect an irreversible physical change. (The word hard is used interchangeably with permanent.)

Intermittent describes a fault that is only occasionally present due to unstable hardware or varying hardware or software states (for example, as a function of load or activity).

Transient describes a fault resulting from temporary environmental conditions. (The word soft is used interchangeably with transient.)

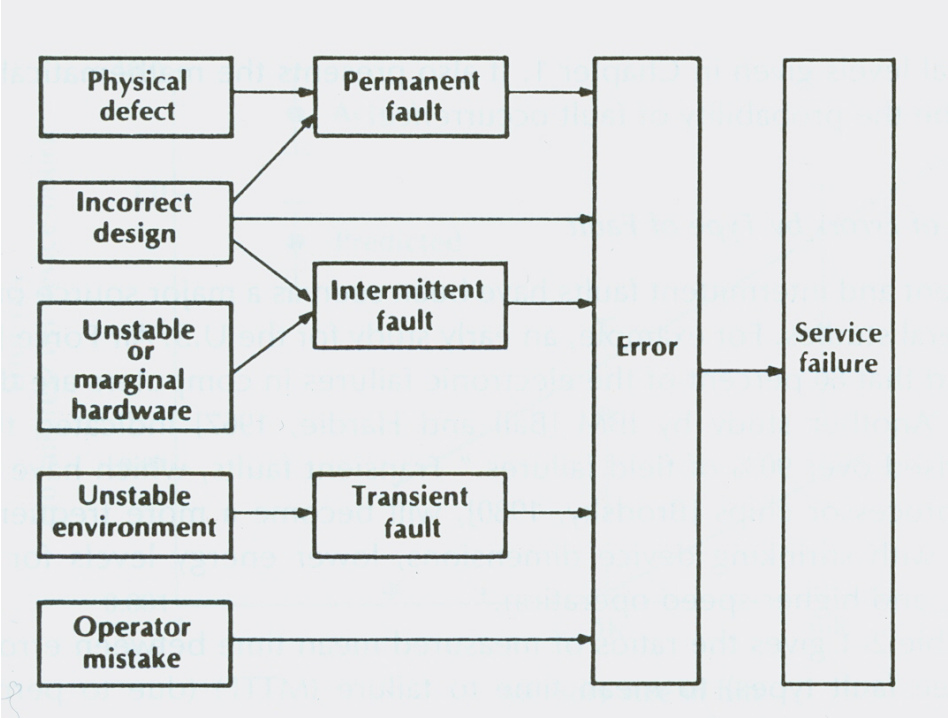


Figure 1.2: Sources of Errors and Service Faults [12]

1.4.2 Reliability and Availability

Reliability and availability have precise definitions in literature. The following definitions are from [12]:

The reliability of a system as a function of time, $R(t)$, is the conditional probability that the system has survived the interval $[0, t]$, given that the system was operational at time $t = 0$. Reliability is used to describe systems in which repair cannot take place (as in satellite computers), systems in which the computer

is serving a critical function and cannot be lost even for the duration of a repair (as in flight computers on aircraft), or systems in which the repair is prohibitively expensive. In general, it is more difficult to build a highly reliable computing system than a highly available system because of the more stringent requirements imposed by the reliability definition. An even more stringent definition than $R(t)$, sometimes used in aerospace applications, is the maximum number of failures anywhere in the system that the system can tolerate and still function correctly [12, p. 4].

The availability of a system as a function of time, $A(t)$, is the probability that the system is operational at the instant of time, t. If the limit of this function exists as t goes to infinity, it expresses the expected fraction of time that the system is available to perform useful computations. Activities such as preventive maintenance and repair reduce the time that the system is available to the user. Availability is typically used as a figure of merit in systems in which service can be delayed or denied for short periods without serious consequences [12, p. 4].

1.4.3 Single Event Phenomena, SEP

Electronic components are vulnerable to a number of effects when exposed to cosmic rays. The collective term for the different failure mode occurrences is Single Event Phenomena (SEP) or Single Event Effects (SEE). Cosmic radiation, or cosmic rays, applies to electrons, protons and the nuclei of all elements. The source of cosmic radiation is either galactic or solar. Galactic Cosmic Rays (GCRs) originate outside the solar system and permeate our galaxy. Solar particle events, in contrast, originate in the Sun and are produced in solar flares. They are lower in energy than GCRs and are mostly protons and alpha particles [13, p. 1-28]. Cosmic radiation is the cause of several types of SEP with different levels of severity, as detailed by Tab. 1.1, that have to be understood when working on a space

mission.

Table 1.1: Single Event Phenomena

Name	**Effect**
Single Event Transient, SET	Soft intermittent fault Propagating through circuit
Single Event Upset, SEU	Soft transient fault State change on latch or memory
Single Event Latchup, SEL	Apparent short circuit Can be mitigated with power cycling Can cause destructive thermal runaway
Single Event Gate Rupture, SEGR	Permanent failure
Single Event Burnout, SEB	Permanent failure

When a charged cosmic particle hits the component, the resulting collision deposits energy in the material. This is known as a Linear Energy Transfer (LET) and is defined as the linear density of energy deposited in material by a charged particle or ionizing radiation traveling through it. The SEU threshold LET is described at the energy level per amount of material of the radiation that will trigger SEU events. For COTS components this is typically $5\ MeV/mg/cm^2$ [5]. The expected SEU error rate for COTS in LEO is $10^{-5}\ error/bit-day$ [5]. This might appear to be a minuscule number, but with 128kB of RAM it amounts to over 10 errors accumulating per day in orbit.

The energy deposited when a particle hits a component can alter the electrical charge in the n-doped material sections of its internal transistors and capacitors. If the charge is altered sufficiently, the voltage level of the transistor or capacitor can change, and this results in change of the stored digital value. This is known as a SEU and is often referred to as a soft error or bit-flip.

The physical effects of cosmic rays in the form of heavy ions and protons on the electronic components are shown in Fig. 1.3. In spite of their small number, the heavy elements are very important due to their densely ionizing tracks. They are responsible for a large portion of the effects in detectors and microelectronics. Particle flux is also larger over the polar regions where ”open” geomagnetic field lines allow easier access [13, p. 1-28]. This means that heavy ions are more damaging to the components, as they can transfer more charge and often have higher energy than than the protons.

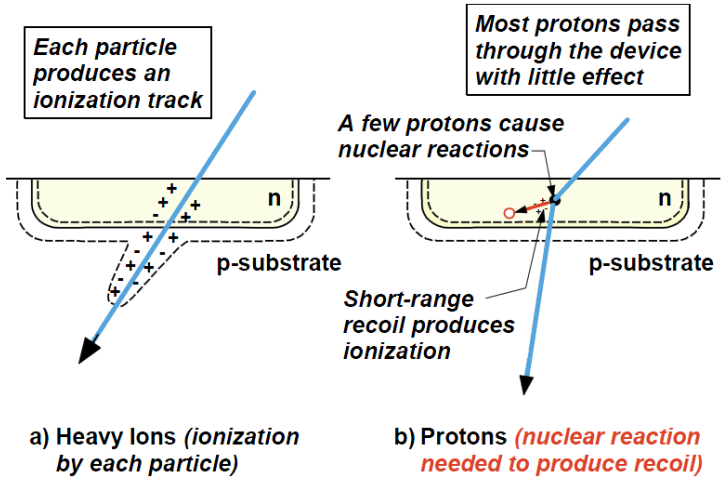


Figure 1.3: Mechanisms for Heavy Ion and Proton SEU effects [6]

If the cosmic ray hits specific parts of the electronic components with enough energy, more severe failures may occur. There are several types of failures that require physical intervention. The SEL manifests as an apparent short circuit that

may cause catastrophic thermal runaway [6], and is triggered when heavy ions, protons or neutrons hit a susceptible point in the component structure. It is only recoverable through power cycle and is strongly temperature dependent with the threshold for SEL decreasing at higher temperatures. Modern devices may have many different SEL paths and a proper characterization of a latchup is a difficult problem. It is also worth noting that modern devices may have both high and low current SELs, something that complicates the characterization further [6]. There are also destructive SEE such as the SEGR and the Single Event Burnout (SEB), but these are permanent and outside the scope of this work.

In this paper the main focus is on the most common, namely SET and SEU, both of which are considered soft errors [14], and the less likely SEL. These three failure modes are the only ones that can be fixed without a component replacement, something that is outside the scope of this work. More severe conditions such as strong electromagnetic pulse (EMP) (which could disable the entire system) or sun storms which would effectively overwhelm the COTS components in the student satellite are not considered.

1.4.4 Total Radiation Dose

The total ionizing dose is the combined damage of the semiconductor lattice that is caused in electronic components exposed to ionizing radiation over time.

The total radiation dose of the system in not a major concern as it will mostly affect mission lifetime. There is some concern, however, due to new effects such as Enhanced Low Dose Radiation (ELDR) sensitivity and subtle failure modes in complex parts. Another concern is that NUTS uses some sensitive technologies with internal charge pumps such as flash memories [6]. The effects of total radiation dose should therefore be kept in mind.

For satellites in inclinations between 20 and 85 degrees in LEO, both in the

northern and southern hemisphere, the typical dose rates are 10-100 Gy per gram of silicon per year [5]. As NUTS will have an even higher orbital inclination, a conservative assumption would be at least 100 Gy per year due to the higher radiation levels close to the Earth's magnetic poles. In combination with the information presented in Tab. 1.2 and [6] we see that some components may fail within the stipulated mission lifetime of 3 to 6 months. Linear Integrated Circuits (IC) , mixed signal IC's and flash memories are the most sensitive components and this should be kept in mind if there are any unexplained failures two to five months into the mission.

Table 1.2: Typical Total Dose Failures Levels for Various Technologies

Technology	**Failure level [Gy/gram Si /year]**	**Worst Case Lifetime [Month]**
Linear IC's	20-500	2.4
Mixed-signal IC's	20-300	2.4
Flash Memories	50-150	6
DRAMs	150-500	18
Microprocessors	150-700	18

Chapter 2

NUTS

NTNU has for a number of years had a student satellite. The main challenge for such small and low-budget systems is to accommodate a certain amount of features while working with a limited budget. In this chapter the limitations and current challenges in the NUTS project will be presented first, followed by the system requirements. The reliability and availability demands for the system, will be presented before the current design is described.

2.1 Limitations and Challenges in the NUTS project

As previously mentioned, some of the main challenges with NUTS are the design constraints in the form of a limited budget. In addition, there are standard satellite constraints such as weight, volume and power use [2]. In the scope of this work, the most relevant limitations are regarding the power usage and to what extent the dependability measures affect the performance of the satellite.

The available power is limited by the relatively small surface area of the solar panels. As there are subsystems that will have to be duty cycled because of their

high power use, it is important to limit the power use as much as possible in the data handling and control systems. The less power that is used continuously, the more the camera, ADCS and radio can be utilized.

The satellite will have various housekeeping routines and will need a certain amount of processing for the payload. When analyzing the satellite's systems and implementing dependability measures, it is important to keep the resource requirements as low as possible. Another factor is that the satellite should be responsive, even when executing error correcting routines.

In the scope of this work the main challenge is one of budgeting. We will have to budget the available resources in the most responsible manner and the more we can reuse available resources the better. The theory and methods presented to solve these problems are not new in themselves [12] [15] [16], but when applied to the relatively new CubeSat platform, the different focus and design challenges of the project makes for demanding work. How can we implement the required dependability and error handling while at the same time providing a responsive and robust satellite?

2.2 Requirements

This section presents the requirements that are relevant in a reliability context. There is no exhaustive system specification at this point, but a complete overview of the specified requirements to date can be reviewed at [11].

Table 2.1: General Requirements

Description
The satellite must execute a one-time initialization sequence on first boot up
The satellite must be able to create and store commands programmatically
It must be possible to initiate a full or partial satellite system reset from the ground station
It must be able to set the current time in the satellite, from the ground station

Table 2.2: Reliability Requirements

Description
Only uncorrupted commands shall be executed
A failing program must not affect the core functionality of the system
Execution of less-important tasks shall not affect the timeliness of higher-prioritized tasks
A frozen system program shall not render the satellite useless

It is desirable to have a highly reliable system. In order to operate correctly the satellite needs rules of conduct for when communication to the ground station malfunctions.

Table 2.3: Autonomous Requirements

Description
Self-repairing to the greatest possible degree
Absolute measurement of time in order to initiate self-repairing correctly
Correctness determination algorithms in order to initiate self-repairing correctly

In this context, the self repairing is primarily regarding the radio. If the radio malfunctions in orbit the satellite is unable to fulfill its mission requirements. There are many possible measures to increase the dependability of the satellite, as will be detailed in another chapter.

The satellite modules should have measure of real time in order to facilitate timeouts on critical systems. To understand the need for time awareness we can review a possible use case in the ADCS module. When the satellite ejects from the launch vehicle it can spin in such a way that the antennas are unable to communicate with the ground station. Therefore the satellite needs to be able to initiate detumbling autonomously after a predetermined time period, especially considering the likely antennae problems on NCUBE-2. The detumbling is the process of stabilizing the angular rate of the satellite after orbital insertion. Another relevant use case is that problems in the EPS causes the satellite to reset periodically or intermittently. This period could be shorter than the timeout to initialize automatic detumbling in the ADCS. Therefore the satellite needs to measure the total time since orbital deployment and initial boot.

Another important requirement is to be able to maintain proper operation during sporadic resets. As some of the dependability measures mitigate errors via power cycling a reset could occur at any time. Therefore, the satellite should stay

in a consistent state and protect its stored data.

The satellite should have measures to counteract radiation effects. As explained in Sec. 1.3.1, the faults considered are soft and recoverable. In case of SEU or SEL the satellite needs to be able to detect the faults reliably and initiate countermeasures.

In addition, the satellite should have some autonomous capacity. If communication to the ground station is lost or delayed because of some unforeseen event, the satellite should behave in a predetermined manner to reduce potential malfunction and inaccessibility. Some appropriate measures include maintaining a power budget, execution of self tests and to activate the beacon for status messages.

2.3 Reliability vs. Availability

In the NUTS' scope of operation it is not necessary with a highly available system. Availability, as described in Sec. 1.4.2, is the proportion of a time a system is in functioning condition. The payload, however, is not dependent on being available at all times. The satellite will have to duty cycle the payload to save power, and as long as it is able to complete and transmit a series of pictures to the ground station, the mission requirements are fulfilled. Therefore the availability requirement of the satellite can be relaxed to an adequate average availability.

The same argument can be used with regard to the radio communication. The satellite is normally in eclipse from the ground station and can therefore be unavailable for considerable fractions of time before it interferes with operations. Even if the satellite should malfunction during communication there are multiple passes each day. As long at the satellite's communication is not degraded to an inoperable degree, it is acceptable that it is unavailable in a window of minutes to hours.

Table 2.4: Design Requirements and Drawbacks

Problem	**Solution**	
	Reliability	**Availability**
Freeze/crash	Low power watchdog	Online backup module
Result check	Recalculation	Result checking logic circuits
Continuity of operation	Saving of current state	Seamless switchover
SEP	Checkpointing	Replication of logic circuits
Drawbacks	Must be very robust	Higher power consumption Extra circuitry

As can be seen from the assessment in Tab. 2.4, the design of a highly available system adds a number of strict requirements for the architecture of the system. The design effort could therefore favorably be shifted towards a system with high reliability and survivability, without the extra overhead of a highly available system.

2.4 Current Design

In the design of the satellite to date, a number of systems have already been specified and built. In order to integrate the solutions for dependability efficiently, we need to study the relevant satellite software and subsystems in the current design. Below follows a brief overview of the most relevant systems and their capabilities and limitations

2.4.1 Backplane

The backplane design includes control logic for submodules, power distribution and control and redundant power and communication buses. In addition, a lot of work has been done to ensure that the functionality of the backplane is realized exclusively with discrete logic. This allows for a more complete state space analysis and ensures that is it possible to account for all of the backplane's states. When all states are accounted for it is possible to guarantee that the backplane does not enter a deadlock, given that all the logic gates remain functional.

Each module slot in the backplane has redundant power supply and system bus connection. In addition, it is possible for one of the two master modules to power cycle or shut down individual modules in the system. In the event of a high current SEL, the chips in the power distribution network will limit the module's current without intervention from the control boards. Another automatic control feature is an integrated WDT for the entire backplane. This enables the satellite to reset in case of a malfunction of both control boards.

The two master modules have access to the programming pins of the MCUs. This means that they have the possibility to reprogram each other in the case of a critical malfunction. This measure should, however, only be used as a last resort due to the possibility of corruption while programming.

The master modules of the backplane have the possibility to communicate independent of the satellite's Inter-Integrated Circuit (I^2C) bus. This is useful because the bus and the overlying protocol are complex. As a result of this it could prove difficult to guarantee schedulability of the prioritized messages between the master modules in the event of a bus malfunction. The use of a separate bus with no contention for heartbeat and other status messages is a valuable feature.

The backplane contains a lot of useful functionality for configuring the satellite in a system reliability context, and will be central as the work progresses. For

further information on the capabilities of the backplane please refer to Dewald's master thesis [9].

2.4.2 On Board Computer, OBC

The MCU on the OBC is an AT32UC3-A3256 [17] with 16Mb additional SRAM and 16GB NAND flash for image processing and storage [11]. The MCU itself has a WDT and a Real-Time Clock (RTC) timer which will both be useful in a reliability context. In addition there is a lot of flash that can be used for storage when implementing checkpointing. The OBC will be placed in a master slot on the backplane with full access to the backplane logic and resources.

2.4.3 Radio

The chosen MCU for the radio module is AT32UC3-A3256 [17]. In addition to the MCU, the radio module have two VHF radios, one for regular communication in full-duplex and one that will function as a beacon with simpler messages in half-duplex. The most interesting component concerning reliability is the MCU, as it is the same that is used in the OBC. This means that it will have the same capabilities with respect to reliability, such as WDTs etc., as the OBC. Like the OBC, the radio will be placed in a master slot on the backplane.

2.4.4 Payload

The primary payload is a camera for atmospheric studies. The Department of Physics wishes to study gravity waves in the upper atmosphere [18]. For further information on the payload please refer to Rønning's master thesis [19]. The payload's signal processing will require a great deal of computational power as detailed in Bakken's thesis [20]. This is arguably the most resource intensive system

on the satellite and it is important to have available computational capacity when necessary. The payload and its processing is the most limiting factor in terms of available processing power and availability in the satellite, as it is important for the mission's success.

A secondary payload in the form of a wireless bus is under construction. The design and implementation is detailed in Frances' thesis [21]. The bus adds redundant means of communication within the satellite without adding much in terms of weight or wiring, and it is therefore beneficial to a CubeSat. Even though the wireless bus may not have the high availability or reliability required for critical subsystems, its high bandwidth can be beneficial for the scientific instruments, and for NUTS the redundant bus is a valuable reliability feature.

2.4.5 Operating System

The satellite uses the FreeRTOS operating system on the OBC and Radio, and potentially on the ADCS and Payload as well. FreeRTOS is an open source real time OS that is module based and easy to customize to different configurations. It is very light weight, support threads and tasks and can be configured to have a POSIX simulator. The addition of an OS is positive in a reliability context because it facilitates increased focus on module based design. There are several advantages:

Scheduling ensures the proper execution of high priority system tasks. This is useful to counteract resource intensive tasks from obstructing the rest of the satellite's systems.

Memory protection is very useful for a system with many different tasks. It ensures that it is not damaging for the system if a task should run out of memory or if there is some faulty memory management in one of the pro-

cesses. In NUTS there are some very memory intensive applications such as video processing.

Communication stack allows for the lower level inter process and inter module communication to be abstracted away from the rest of the development.

File system stack allows for easy storage and retrieval of data. By using an OS it is possible to port the well tested and well performing YAFFS2 file system for flash storage.

The reasons detailed above allow the teams working on the specific applications to abstract away the system level programming and should supply a better developing environment.

2.4.6 Cubesat Space Protocol, CSP

Development on CSP started at Aalborg University in 2008. It is a small network-layer delivery protocol designed for CubeSat missions [22]. This is an important addition to the project as it allows the use of an advanced protocol for module and ground station communication, without incurring much of a development burden. The CSP protocol implements drivers (layer 1), MAC interfaces (layer 2), network router (layer 3) and a reliable datagram protocol (RDP) in the transport layer (layer 4) [22]. It is planned to be used at both the ground and space segment.

In the context of system wide reliability, CSP plays a significant role. The protocol is the fundament for communication between the satellite's different modules and subsystems, and will be used to determine the correct operation of said modules. The work that has been done regarding the reliability of CSP can be found in Sec. 4.2.2.

Chapter 3

Evaluated Solutions

As preparatory work to this paper a number of possible additions were evaluated with the intention of increasing the dependability of NUTS. Some of the solutions have been implemented while the rest are suggested as future work. The Error Detection And Correction (EDAC) module will make it possible to detect and correct the SEPs occurring in memory. Checkpointing is a proven technique to make the system resistant to losing data or operational context while recovering from failures. Master-Slave functionality allows for a spare control computer in case the main one malfunctions. The Watch Dog Timer (WDT) ensures that the system does not deadlock 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 safeguards against a malfunctioning module affecting the rest of the system. Finally, the ability to perform an integrity check on the program memory ensures that possible errors can be detected and restored.

3.1 Error Detection and Correction, EDAC

Due to the random nature of the expected faults 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 [23, p. 155] codes to correct the faults at run-time.

Since executing BCH codes in individual tasks would add a layer of complexity, a system task that manages the secure storage and recovery of protected data should be considered. A specialized EDAC system task with practical interface functions makes the system development more comprehensible by removing the sometimes complex algorithms from the scope of the developer. A module based design is 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 [12, p. 202].

Last, but not least, the number of detected radiation induced errors should be logged. It is useful both for this and future designs to have a numerical value on how well the satellite performs with respect to dependability and reliability.

3.2 Checkpointing

Checkpointing is a proven solution in software system redundancy. Checkpointing stores the system state that is necessary for continued execution and completion of the process at specific points during process execution [12, p. 214]. 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 [15]. It is important to ensure that the system is able to roll back multiple instances in case there is some unforeseen fault present that could compromise any single checkpointing instance.

Power cycling of faulty modules is implemented in the backplane. The mod-

ules 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.

3.3 Master-Slave

The design of the backplane allows for two master modules that can control the backplane logic and communicate independently of the system bus, allowing master-slave functionality in the control of the system. This enables one module to take over operations if the other one should fail. With the assumption that it is unlikely for both modules to fail at the same time, this increases the likelihood of the satellite being continuously operational.

An added complexity to the traditional master-slave setup is that the modules do not have the same capabilities. The radio module, for instance, is the only module that can communicate with the ground segment and the OBC is the only module with sufficient RAM and storage to perform image processing from the payload. The system would therefore operate with reduced functionality as long as the radio or OBC is offline. The backup module should therefore power cycle the main module in addition to managing the rest of the satellite.

When master-slave is implemented as tasks running on the subsequent modules there are a number of things to keep in mind.

Simple code is more analyzable since it has fewer states. A small amount of code is also less likely to experience SEE.

Independent software should be developed to interact as little as possible with

the rest of the system. By limiting the interaction with the rest of the system's software, the possibility of being trapped in a deadlock or waiting for an unavailable resource decreases.

Independent communication from the system bus should be possible. A high level reliable transmit protocol is complex to analyze and also has numerous points of failure. It can also be difficult to ensure its real time capabilities (see Sec. 4.2.2).

3.4 Watch Dog Timer, WDT

A very useful and often employed concept in reliable computing is the WDT. The basic functionality of the watchdog timer is fairly simple. The running program must periodically reset the WDT and if this fails the system will restart. The WDT should be activated every time the system performs and input, output or waits for an internal module. It is also possible that the system enters an unrecoverable state in other sections of the code, if a SEP occurs, as described in Sec. 1.4.3. In addition, a WDT protects against weaknesses in the system design and ensures that the system will not malfunction in the event of an untested software bug [12, p. 130].

All of the MCUs used in NUTS have internal WDT modules that should be used. In addition, it is possible to use an external IC with WDT functionality. This IC is located on the backplane and will provide a full system reset. Given the use of WDTs on each module and the possibility of losing data in the entire satellite, the WDT on the backplane should only be activated if both of the control computers fail to respond.

3.5 Periodic Reset

When designing the EDAC service it is difficult to guarantee full coverage of the error detection. Some errors may be left undetected and the EDAC service might be overwhelmed in periods with high radiation intensity (e.g. when passing through the South Atlantic Anomaly). Another consideration is that the EDAC service is not meant to be on a system-wide level due to the overhead of the implementation, especially for multiple-error-correcting codes [12, p. 147].

In modern components it may also be difficult to properly detect SEL events, since both high and low current SELs can occur (see Sec. 1.4.3). The high current SEL is caught by the backplane which triggers an automatic reset on excessive current consumption, but no such mechanisms exist for low current SELs is the design.

The possibility of undetected memory corruption and low current SELs is a real concern as they are both difficult to determine properly. To solve this problem a periodic power cycle of all the modules is suggested.

3.6 Disabling Faulty Modules

Due to the possibility of permanent failures in the satellite's modules it should be possible to execute a controlled shutdown of the different subsystems. The main difficulty with this is to construct algorithms to determine which modules are not operating correctly. Generally speaking the modules can fail in two ways, either a silent failure where the module becomes unresponsive, or what is known as a babbling-idiot failure [24]. A babbling-idiot fault typically occurs when a node occupies the bus and transmits high-priority messages at erroneous time instances so frequently to cause additional delay in the communication of properly operating nodes. The worst-case scenario happens when a node keeps the bus continuously

busy, thus inhibiting every communication between the other nodes [24].

Fault determination measures can also fail themselves. This is mitigated with the presence of two control modules and the assumption that only one will critically fail at a time. This is the same number of control computers that NASA originally had on the space shuttle engine control [25]. It is considered safe because of the low probability of radiation hitting the same part of the logic in two separate modules. Once the faulty module is determined it can be disabled via the backplane logic.

3.7 Program Integrity Check

In addition to secure storage there should be a periodic subroutine that performs a Cyclic Redundancy Check (CRC) on system flash in order to determine if there has been corruption of the program flash. This can be done alternately by two or more different but identical functions placed in different parts of the flash to mitigate the risk of an error in the CRC function itself. The CRC is possible to implement with a small amount of code and it should therefore be a feasible solution.

It should be possible to store copies of the main program in the large flash data-bank available. If this data is maintained by an error-checking subroutine and checked before flashing other modules it should be reasonably safe. In case of failure of part of the flash-bank, the programs could be stored multiple times on different parts of the flash.

In addition, one could implement a testing routing to be performed by another microcontroller. This routine would invoke a critical subset of software and hardware modules and check their result against answers calculated in advance.

3.8 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 module that 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 [26] 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.

Chapter 4

Implemented Solutions and Test Environment

This part presents an overview of the solutions that have been implemented for NUTS as well as a detailed description of the test environment, setup and tools used. First we will present a summary of other projects' solutions for EDAC and increased fault tolerance.

4.1 EDAC on Other Satellites

A number of other other CubeSat and NanoSatellite projects focuses on deploying robust EDAC to counteract SEP during satellite missions. At the 5th European Cubesat Symposium in Brussels a number of different solutions were presented.

In his presentation, X. Yu [27] solved these problems by using a System on Programmable Chip (SoPC) approach on a commercially available FPGA. This chip comes with anti-SEU technology internally and a system-level backup hot redundancy.

The presentation by B. Osbourne [28] deals with the inclusion of an FPGA in the payload board called RUSH. This computer board will demonstrate and validate new approaches for rapidly recovering from radiation induced errors in reconfigurable hardware.

V. Broun [29] addresses the effectiveness of shielding analog and digital components and measures radiation received by the components. Different areas of the satellite have different types and thicknesses of shielding and the result of the identical components are compared to determine the effectiveness of the different shielding configurations.

In his presentation, T. Rajkowski [30] deploys an FPGA in combination with a low-power MCU. They use CRC and modular redundancy of functional blocks in addition to NOR flash working in triple redundancy to store FPGA configuration. In order to provide latch-up protection they monitor current consumption and intervenes if it exceeds safe levels. Watchdogs and coupled watchdogs are applied to recover from functional interrupts. In most sensitive points of design an analogue voting mechanism is used.

4.2 Design Justification

4.2.1 Software Based approach

What are the reasons for choosing software based approach in this thesis? This section presents the positive properties of a software approach to dependability and EDAC and in which cases it is applicable.

A software approach can have very low overhead in the system design phase. The requirement for including software in a project is very low, essentially only a compiler and some processing time is required. Compared to developing hardware features, especially if we consider prototyping and production lead time, a

software approach could decrease development time. In addition, this approach can increase dependability (as we will show) while having a good cost to benefit ratio, making it ideal for low-cost CubeSats.

Microprocessor control is becoming more common, and with this increase, the number of critical or important applications that are controlled by microcontrollers increases as well. As process technology continues to decrease in size, all computer systems face an increasing risk of experiencing faults caused by cosmic rays.This infers that resistance to and consideration of these effects is becoming more important.

We want our designs to be useful for other projects than NUTS. The software approach is useful for many small designs, for instance RF sensor networks. They are often battery powered and constricted to a small form factor. A redundant hardware design would use valuable space and power on these systems, resulting in a bulkier, shorter lived and ultimately less useful sensor network. These networks have other features that make them particularly suitable for a software dependability approach. The protocols are already resistant to temporary loss of connection and varying response times, both of which are a requirement for using the soft dependability approach proposed in this thesis.

The software approach adds dependability with low extra cost in applications were it would not otherwise be practical. It is important to remember that an increased software load results in more executable code which reduces sleep time and thus increases power use as well. Nevertheless this impact should be smaller than duplication of functionality for most systems. It is also worth noting that while a software based dependability approach is useful in some applications they are still limited. In critical applications with a high reliability requirement, such as safety features in modern cars, the temporary downtime while the system performs error correction could be catastrophic.

4.2.2 Critical Communication Between Modules

The communication between the modules in NUTS is based on I^2C and CSP. In order to understand the ramifications of this decision for the reliability of the satellite we refer to the previous work [31]. The text is adopted for this paper and the entirety can be found in Appendix E or the electronic attachments.

CSP is a message based communication protocol that allows the creation of sockets for inter process and network communication. In order to determine if CSP is fit for communication between critical modules a formal verification of the protocol was attempted. A process algebraic model of CSP was constructed and analyzed as a Finite State Process (FSP) in the Labeled Transition System Analyzer (LTSA) tool. The chosen error model was a standard implementation of a unreliable medium as the one in [32].

The Cubesat Space Protocol implements a Reliable Datagram Protocol (RDP) in accordance with RFC-908 [22]. A large portion of the work was to translate the protocol from documentation and source code to an accurate FSP model. The implemented model of CSP was deadlock and livelock free with the provided error model. However, due to the possibility of implementation of changes to the protocol, it is difficult to ensure that the implemented model is an accurate representation of the source code. In order to ensure an updated model one would need to devise and employ an automatic translation tool, otherwise it would be too time consuming.

Given the difficulty of verifying the correctness of the transfer protocol and the possibility of compromised operation due to radiation effects, a simple communication form was chosen for the critical systems. A heartbeat and a boolean test is easily implemented as a very low level protocol by toggling the General Purpose Input Output (GPIO) lines of the microcontrollers. This form of communication has a low information content but should be considerably more reliable

than a bus protocol.

4.3 Functional Overview of Implemented Solutions

In addition to the literary study and review of possible solutions for the satellite, there have been conducted more detailed studies on a couple of subjects. A presentation of this work is given in this chapter.

4.3.1 Master-Slave

The Master-Slave setup is presented in Fig. 4.1 and 4.2. The setup is realized with two Xplained boards [33] representing the OBC and the radio modules in the satellite. The master and the slave each have access to a GPIO line (used as a heart beat) and the reset line of the other. In this setup both the master and the slave emits a periodic heartbeat in state S_{HB}. In S_M the slave monitors the master's heartbeat and vice versa. If no heartbeat is received a timeout occurs switching the state to S_{TO} and then to S_{Re}, resetting the other processor, as can be seen in Fig. A.1 and Fig. A.2. If the master sends the reset signal it switches to S_M and resumes normal operation. On the other hand, if the slave sends the reset signal it assumes the role of master in S_{AM}. If a heartbeat is received, the slave goes back to S_M. This variation to the traditional setup is necessary since the master and slave do not posses the same capabilities. The intention is that a reset and/or power cycle will clear any SEU or SEL present in the other processor and restore normal functionality. The timeout duration is increased on consecutive resets to combat a reset cycle where the master does not have sufficient time to assume normal operation before the next reset. A logic analyzer was was used to tap the communication lines and monitor the operation of the master and the slave.

We use a GPIO connection because it is one of the simplest information trans-

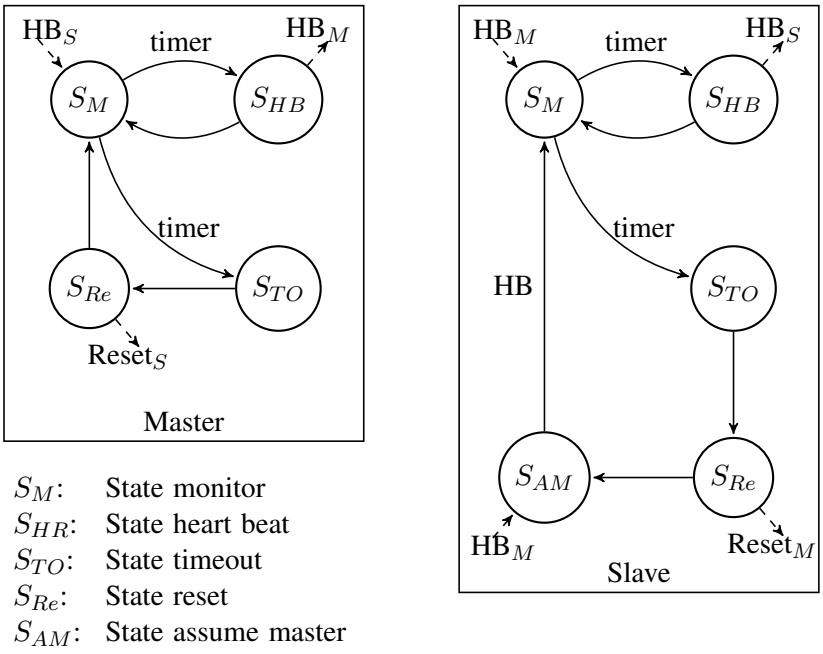


Figure 4.1: Master-Slave Description

fers that is possible between two MCUs. The only required action of the program code is to write to an I/O register. Compared to a bus interface such as I^{2}C it has far less likelihood to fail, especially when a network protocol such as CSP runs on top of said interface [31]. In addition, guaranteeing prioritized communication on a multi-master I^{2}C bus is difficult. It is possible that the bus will be polluted with messages from other malfunctioning modules [24]. If an external interrupt pin can be spared on the master and slave it is requires less resources to wake up from sleep to register the heartbeat. Considering all this we advocate for this form of communication to be used in the final revision of the satellite.

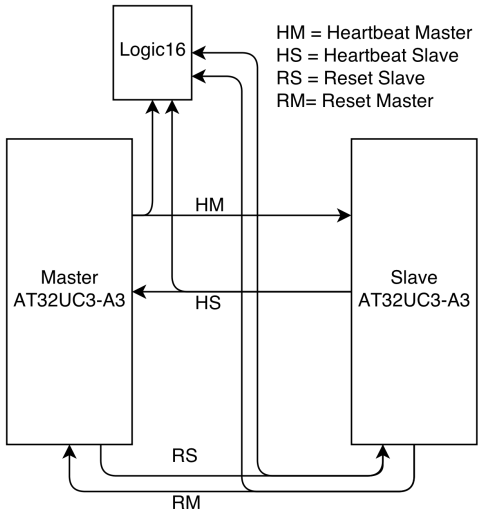


Figure 4.2: Master-Slave Connection Diagram

4.3.2 CRC

The CRC is implemented with code borrowed from Atmel Software Framework (ASF) and is the first routine that executes after boot. The same development environment with two Xplained boards were used and the functionality is described in Fig 4.3. GPIO lines were used for direct communication between the master and the slave. At this point there is no action taken if a CRC error is detected. In order to correct faults in the program memory of the MCU one would need to reprogram the whole or part of the system flash, and this is a risky operation, especially if it is to be done automatically. At this time the NUTS project has not included reprogramming because of these risks, but the CRC routine would still provide valuable information regarding the health of the system.

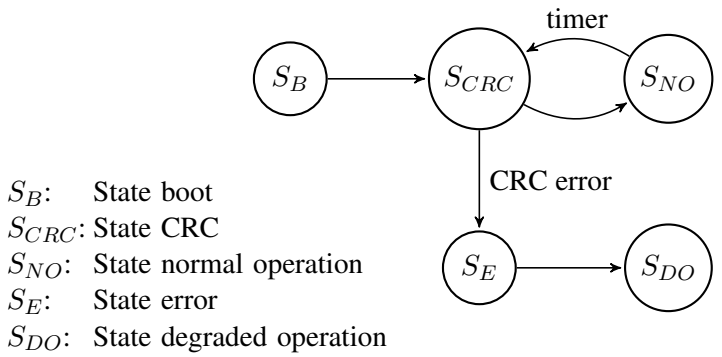


Figure 4.3: CRC Functinal Overview

4.3.3 Checkpointing and EDAC with BCH

The EDAC and checkpointing is implemented as an integrated system. Figure 4.4 shows the components of the EDAC system and a brief presentation of functionality is provided in Fig. 4.5. 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 cannot be recovered the system enters the checkpoint stages ($c_1, c_2, \ \dots \ , c_n$). If the rollback is successful the system continues, if not it resets.

4.4 Tools

4.4.1 Atmel Studio

Atmel studio is a free to use IDE for AVR ICs developed by Atmel. It is the chosen development environment in the satellite project and its repositories. The software

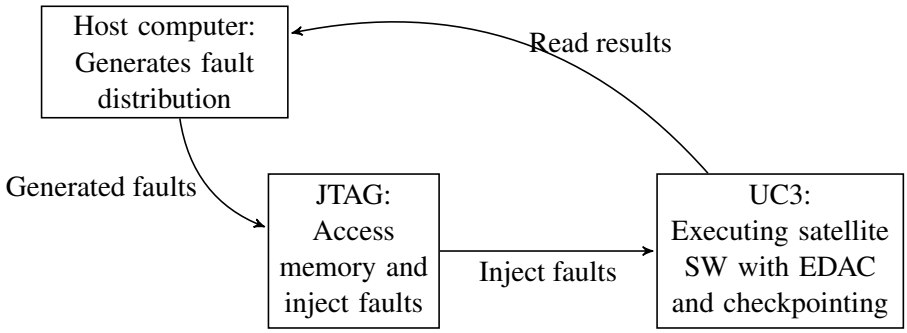


Figure 4.4: Test Environment

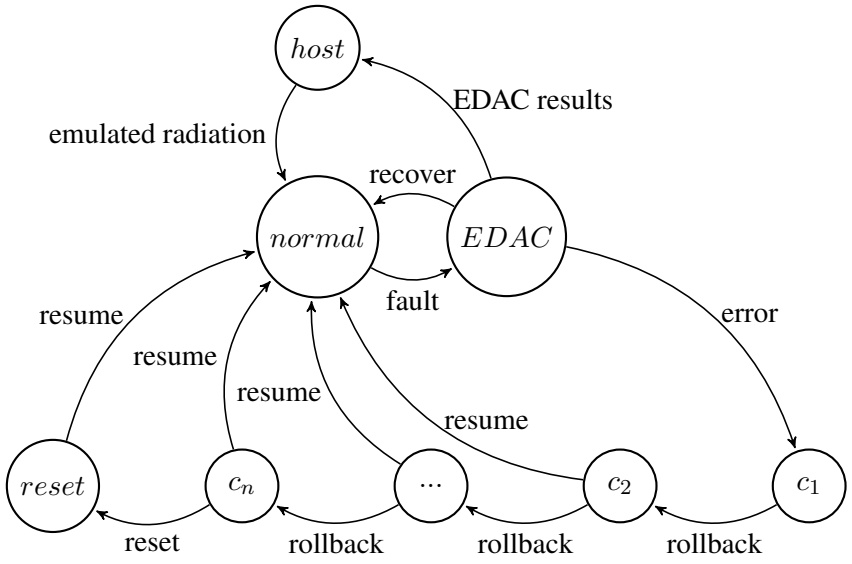


Figure 4.5: Functionality Description

for the Xplained and EVK1104 have been developed on Atmel Studio 6.0.1996 with the AVRGCC 3.4.1.95 tool chain. We use the ASF to provide drivers for external components and a protocol stack for communication between the host computer and the Xplained board.

4.4.2 Atmel Xplained UC3-A3

The Xplained is a small development board from Atmel with the same microcontroller (AT32UC3-A3256 [17]) that NUTS uses. It is conveniently powered by USB and have most of the necessary pinouts for development. It has a number of common functionalities and drivers provided by ASF which eases development.

4.4.3 Atmel EVK1104

The EVK 1104 is a larger development board from Atmel with the same microcontroller (AT32UC3-A3256 [17]) that NUTS uses. It has a larger package microcontroller and therefore more pinouts. It is used mainly because of its included flash memory which is similar to the flash memory of the satellite. A number of drivers and common functionalities is provided by ASF which eases the development.

4.4.4 Logic Analyzer

When developing solutions that rely on multiple microprocessors and the communication between them a logic analyzer is an invaluable tool. Besides the advantageous overview of the bus communication the logic analyzer has several advantageous features. Compared to the USB communication stack available in Atmel Studio it has lower overhead in memory when used to communicate with a computer, as no additional code needs to execute. By using the logic analyzer both

for communication with the computer and debugging the bus it is easy to have the correct context between messages, bus communication and state changes. Unlike the USB stack and the accompanying drivers, it is robust with respect to restarts. We could achieve this with RS232 serial communication but we would lose the context to the bus communication. In addition, the development boards does not have a RS232 IC. The particular logic analyzer used was the Logic16 from Saleae that is provided by the project.

4.5 Test Environment

The experimental systems consists of a host computer and an *Xplained* development board [33] from Atmel. The development board uses the AT32UC3-A3256 microcontroller [17], 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 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 addition, the developed code was tested on the actual satellite hardware, as can be seen in Fig. 4.7.

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

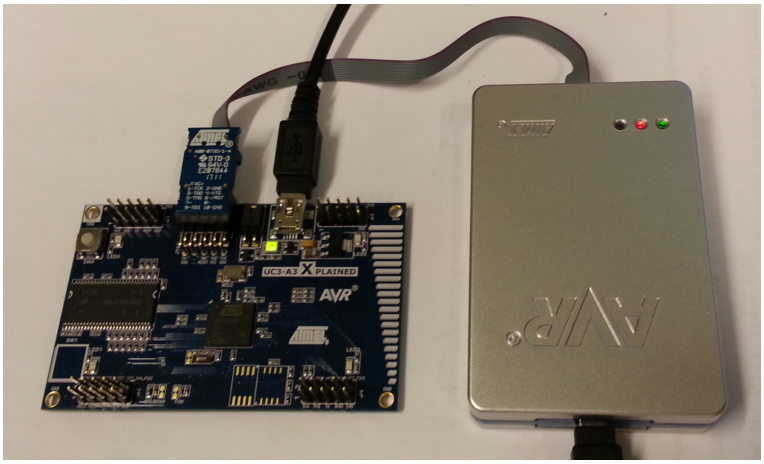


Figure 4.6: UC3-A3 Xplained and JTAG ICE3

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 periodically. 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.

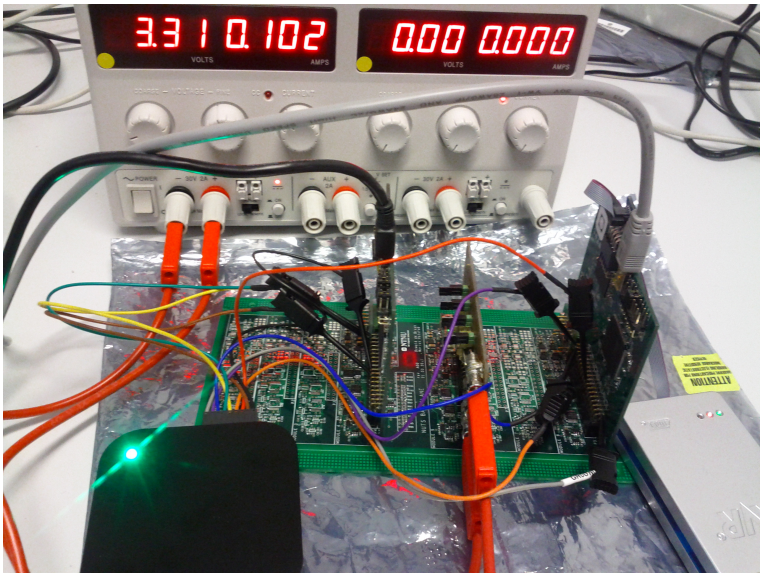


Figure 4.7: Test Setup with Satellite Hardware

The result from use of the error correcting code is in Tab. 4.1. 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 4.1 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. 4.2 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 4.2 is included to demonstrate the maximum effectiveness of the correctional codes under ideal circumstances.

Table 4.1: 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
98	0	103	0	90	0	0	0	0.9	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 4.2: Error Correction for Ideal Case

Injected faults	Errors
128	0
224	0
256	32

4.6 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 [23, 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 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 [23, 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% [23, 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 [34] 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.

4.7 Encountered Problems

In this section a number of encountered problems are collected to be helpful for future development. The collection of these in a section is meant to be used as a reference for further development in the project.

How do we test for correctness of the error correction when we do not have a reference that is guaranteed to be correct? When we develop in a laboratory environment we can always keep a reference of the data that is used in the EDAC system. Once the satellite is deployed we do not have the possibility to do this as the storage is compromised. This is a difficult problem and we have some possible solutions. We can store the data multiple times and do a majority vote, but this increases computational time threefold. Checkpointing can be used to keep multiple temporal copies and would not add the same continuous overhead. It is also possible to add an extra CRC after the correction to determine if the recovered data is equal to the stored data. All of these schemes can mitigate the possibility of unrecoverable faults. However, we still have the problem of correctly determining

the present of an unrecoverable error, as they are all vulnerable to an extra error in the added protection. At this time it is left as future work as no satisfactory solution could be devised.

The implemented solutions for dependability and EDAC can also fail themselves. The presence of errors in the CRC code can result in constant reporting of failure even though the rest of the system code is functional. This can be mitigated with the inclusion of two or more CRC routines checking and storing individual results. If this is done it must be ensured that compiler optimization does not combine this code in the same place. The EDAC code can also fail to correct data or store erroneous data in the flash. Some protection against this is provided by checkpointing and an algorithm to determine correct functionality is left as future work.

The simulated serial over USB have caused some problems. The CDC driver for Windows does not function properly when the device is reset and the serial connection malfunctions. Another problem is that the serial implementation buffers the output and halts the microcontroller when the buffer becomes full. For larger amounts of data and proper formatting the serial connection is useful but otherwise we used a logic analyser for lower level access. RS232 would be useful but the development board does not have a RS232 IC.

While it is easy to create a random distribution of faults it is harder to determine if it is realistic for the particular IC. How does the cosmic ray affect the area of the chip that is hit and how are the neighboring transistors affected? A fault pattern from a cosmic ray in RAM could affect several memory bits depending on the angle of attack and the energy of the cosmic ray but it is difficult to know exactly how without testing or using components with flight heritage. Because of this we have to make assumptions that may or may not be correct for the components we are using. Considering this it seems prudent to make a conservative assumption of how many faults that will occur and adjust the EDAC accordingly.

4.8 Future Work

There are many possible directions for the future work on reliability and dependability that could be interesting for the NUTS project.

If a continued low overhead approach is desirable one could focus on the software methods proposed in this paper. For the EDAC codes RS could be used in stead of BCH. RS codes have better performance in burst error cases and use simliar computational power when decoding [23]. In addition, further development on a safe periodic reset and proper categorization and shutdown of faulty modules could be worked on. The checkpointing scheme implemented in this thesis needs more work. In its present form it stores some steps with data from system tasks, but not lower level system information such as memory content and status registers. Additionally, in order for the mechanisms to function reliably, a robust error determination algorithm needs to be implemented.

Reprogramming between the radio and the OBC from a secure storage could add more advanced error recovery to NUTS. This is a hazardous operation in a remote environment with cosmic rays, but could be implemented as a last resort alternative.

If the satellite project requires stricter guarantees on reliability and dependability one could integrate the protection in hardware. Memory protection with dual memory blocks and FPGA is an approach used by other CubeSat projects (see Sec. 4.1). The FPGA could the be reconfigured to mitigate SEU [35] and one could use Magnetoresistive Random-Access Memory (MRAM) which stores data as magnetic values rather than as electric charge, and is therefore more resistant to SEUs [36]. Accompanying this with a robust boot loader would be an interesting addition to this or the next satellite.

Alternatively, by using the ideas of reconfiguration of a FPGA from [35], one could implement a soft microprocessor as the system computer in the satellite.

By using this approach it should be possible to mitigate effects such as SEU and SEL without the use of spare components. Notable issues include being able to reconfigure the soft core in safe way (relative to the rest of the satellite), power consumption and performance.

Chapter 5

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 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.

One important consideration is that the correctness of the error correcting codes are hard to determine once in orbit. In the test setup the initial and corrected data are compared to determine the number of uncorrected errors. However, once the satellite is operational, this is no longer possible. It is therefore important to make sure that the codes have high enough correctional capabilities. The same ting can be achieved by having a sufficiently high frequency on the correctional subroutine.

Some of the strategy 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 1970s.

The future work will focus on implementation of the solutions discussed in this paper. As more of the subsystems reaches 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.

Bibliography

[1] K. A. Ødegaard and A. Skavhaug, "IAA-CU-13-13-01 Survey of correction methods for faults and errors induced by cosmic radiation on operating system level in CubeSats," *Proceedings, 2nd IAA conference*, 2013.

[2] Specification, CubeSat Design, "Rev. 12," *Cal Poly, August*, 2009.

[3] J. Puig-Suari, C. Turner, and W. Ahlgren, "Development of the standard cubesat deployer and a cubesat class picosatellite," in *Aerospace Conference, 2001, IEEE Proceedings.*, vol. 1, pp. 1–347, IEEE, 2001.

[4] J. T. Gravdahl, E. Eide, A. Skavhaug, K. Svartveit, K. Fauske, and F. M. Indergaard, "Three axis attitude determination and control system for a picosatellite: Design and implementation," in *Proceedings of the 54th International Astronautical Congress*, 2003.

[5] JPL NASA, "Space radiation effects on electronic components in low-earth orbit," *Practice no. PD-ED-1258*, vol. 1, 1996.

[6] A. Johnston, G. Swift, L. Scheick, and J. Conley Jr, "Space radiation effects on microelectronics," *Jet Propulsion Laboratory, Electronic Parts Engineering Office, Section*, vol. 514, 2002.

[7] R. Birkeland and O. Gutteberg, "Overview of nuts," tech. rep., Technical report, NTNU, 2013.

[8] R. Birkeland, "Nuts-1 mission statement," tech. rep., Technical report, NTNU, 2011.

[9] D. De Bruyn, "Power distribution and conditioning for a small student satellite," *Master's thesis, Norwegian University of Science and Technology*, 2011.

[10] L. E. Jacobsen, "Power system of the ntnu test satellite," *Master's thesis, Norwegian University of Science and Technology*, 2011.

[11] D. E. Holmstrøm, "Software and software architecture for a student satellite," *Master's thesis, Norwegian University of Science and Technology*, 2012.

[12] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems: design and evaluation*. Digital Press, 1992.

[13] E. M. Silverman, "Space environmental effects on spacecraft: Leo materials selection guide, part 2," *Progress Report, Apr. 1993-Mar. 1995 TRW, Inc., Redondo Beach, CA. Space and Electronics Group.*, vol. 1, 1995.

[14] R. W. James and J. L. Wiley, *Space mission analysis and design*. Kluwer Academic Publishers, London, USA, 3rd ed., 1999.

[15] S. Skavhaug and O. Pettersen, "Microfaulttolerant (μft)-a system for achieving cost effective fault tolerance in microcontroller based equipment," in *Real-Time Systems, 1995. Proceedings., Seventh Euromicro Workshop on*, pp. 344–351, IEEE, 1995.

[16] A. Skavhaug, *A holistic approach to development of dependable industrial SCADA systems*. PhD thesis, Norwegian University of Science and Technology, 1997.

[17] Atmel, "At32uc3a3." `http://www.atmel.com/Images/doc32072.pdf`, 2013.

[18] N. McFarlane, "The effect of orographically excited gravity wave drag on the general circulation of the lower stratosphere and troposphere," *Journal of the atmospheric sciences*, vol. 44, no. 14, pp. 1775–1800, 1987.

[19] S. S. Rønning, "Optimizing an Infrared Camera for Observing Atmospheric Gravity Waves from a CubeSat Platform," *Master's thesis, Norwegian University of Science and Technology*, 2012.

[20] M. Bakken, "Signal processing for communicating gravity wave images from the NTNU Test Satellite," *Master's thesis, Norwegian University of Science and Technology*, 2012.

[21] J. Frances, "Internal Wireless Bus for a CubeSat," *Master's thesis to be published, Norwegian University of Science and Technology*, 2013.

[22] GOMSpace, "Cubesat space protocol (csp) network-layer delivery protocol for cubesats and embedded systems.," *GS-CSP-1.1*, 2011.

[23] H. Malepati, *Digital media processing: DSP algorithms using C*. Newnes, 2010.

[24] G. Buja, J. R. Pimentel, and A. Zuccollo, "Overcoming babbling-idiot failures in can networks: A simple and effective bus guardian solution for the flexcan architecture," *Industrial Informatics, IEEE Transactions on*, vol. 3, no. 3, pp. 225–233, 2007.

[25] J. E. Tomayko, "Computers in spaceflight the nasa experience," *Computers in Spaceflight The NASA Experience*, vol. 1, 1988.

[26] O. Hannius and J. Karlsson, "Impact of soft errors in a jet engine controller," in *Computer Safety, Reliability, and Security*, pp. 223–234, Springer, 2012.

[27] X. Yu *et al.*, "Research on the on-board computer subsystem of cubesats based on sopc technology," in *5th European Cubesat Symposium, 2013, Book of Abstracts.*, vol. 1, pp. 1–135, Von Karman Institute for Fluid Dynamics, 2013.

[28] B. Osborne *et al.*, "UNSW EC0 CubeSat Design: Experiments in Radiation Tolerance Critical Systems, GNSS Remote Observation and 3-D Printed Satellite Structures," in *5th European Cubesat Symposium, 2013, Book of Abstracts.*, vol. 1, pp. 1–135, Von Karman Institute for Fluid Dynamics, 2013.

[29] P. V. V. Broun, P. Camus and J.-M. Gillis, "Comparison of the effectiveness of different kinds of radiation shields on a cubesat," in *5th European Cubesat Symposium, 2013, Book of Abstracts.*, vol. 1, pp. 1–135, Von Karman Institute for Fluid Dynamics, 2013.

[30] T. Rajkowski and R. Graczyk, "Picard, an on-board computer for future cubesat missions," in *5th European Cubesat Symposium, 2013, Book of Abstracts.*, vol. 1, pp. 1–135, Von Karman Institute for Fluid Dynamics, 2013.

[31] K. A. Ødegaard, "Correction of faults and errors induced by cosmic radiation on operating system level in the nuts studsat project," *Project report, Norwegian University of Science and Technology*, 2012.

[32] J. Magee and J. Kramer, *Concurrency: State Models & Java Progamming*. Wiley, 2006.

[33] Atmel, "Uc3-a3 xplained." http://www.atmel.com/Images/doc32159.pdf, 2013.

[34] ETSI, EN300744, "300 744 digital video broadcasting (dvb); framing structure, channel coding and modulation for digital terrestrial television," 2004.

[35] J. Alme, *Firmware Development and Integration for ALICE TPC and PHOS Front-end Electroncis*. PhD thesis, PhD thesis, Universitetet i Bergen, Bergen, Norway, 2008.

[36] D. Nguyen and F. Irom, "Radiation effects on mram," in *Radiation and Its Effects on Components and Systems, 2007. RADECS 2007. 9th European Conference on*, pp. 1–4, IEEE, 2007.

Appendix A

Plots and Data

A.1 Code Analysis

In order to measure the relative complexity of various branches of source code the number of lines have been counted.

Satellite repositories

http://cloc.sourceforge.net v 1.58 T=0.5 s

Language	files	blank	comment	code
C	87	4300	5625	14876
C/C++ Header	102	3103	12819	8287
Assembly	4	117	293	242
SUM:	193	7520	18737	23405

Minimal representative test system with OS and drivers

http://cloc.sourceforge.net v 1.58 T=0.5 s

Language	files	blank	comment	code
C	49	3809	6995	11645
C/C++ Header	89	2767	13890	6149
Assembly	4	117	293	242
SUM:	142	6693	21178	18036

EDAC and checkpointing

http://cloc.sourceforge.net v 1.58 T=0.5 s

Language	files	blank	comment	code
C	55	4373	8538	13488
C/C++ Header	102	3115	15757	6994
Assembly	4	117	293	242
SUM:	161	7605	24588	20724

A.2 Plots

These plots show the behavior of the master and slave in relation to each other.

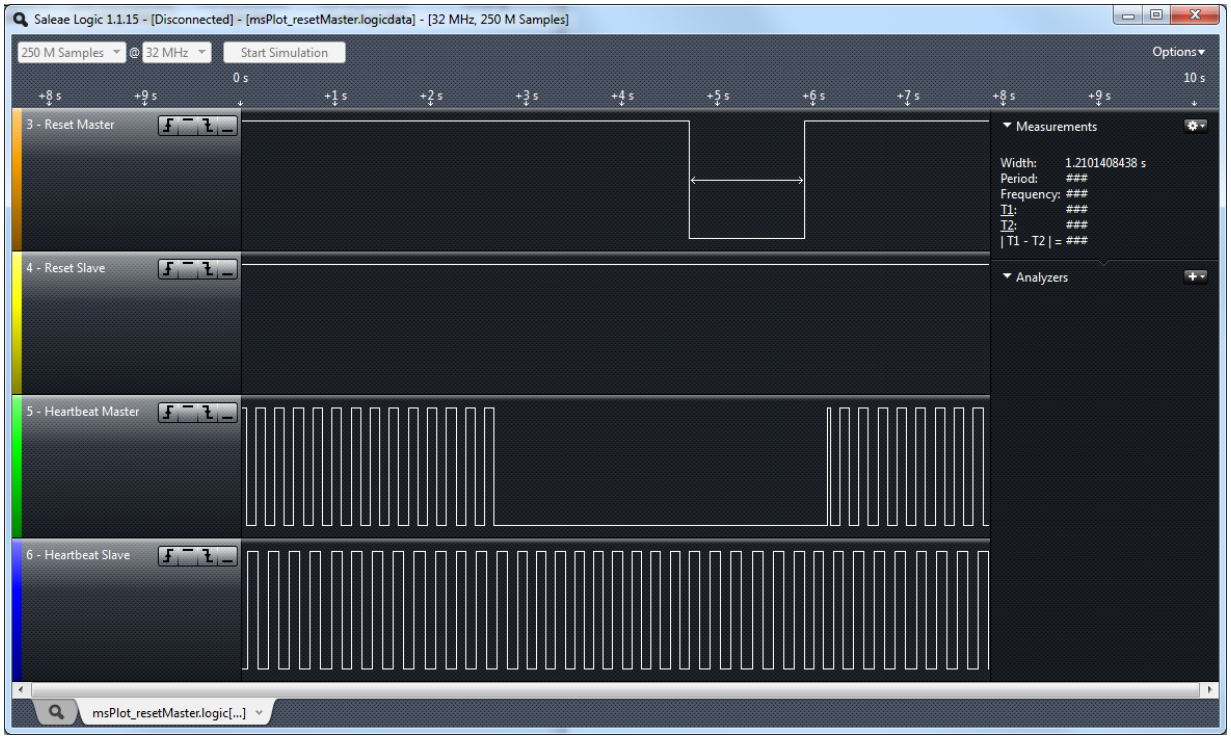


Figure A.1: Timeout and Reset of Master

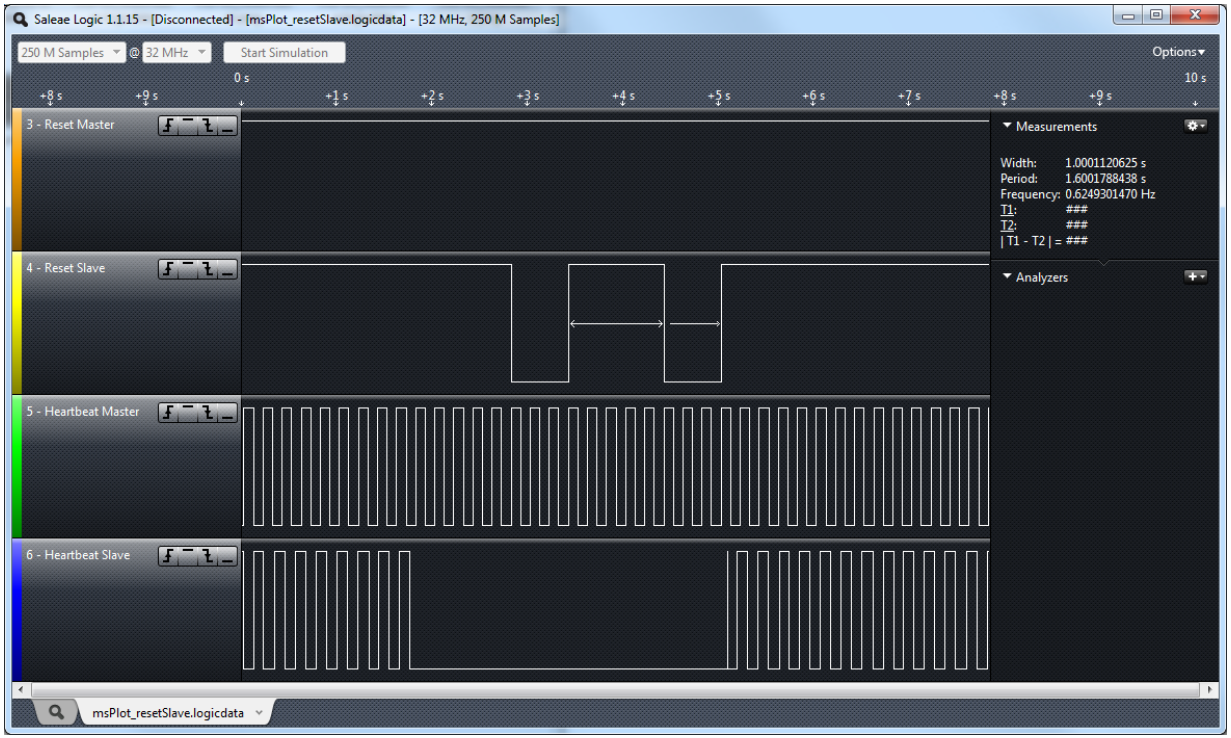


Figure A.2: Timeout and Reset of Slave

Appendix B

The 2nd IAA Conference

Part of the work in this thesis have been presented on two conferences thanks to NUTS and our sponsor NAROM. The contacts and new ideas obtained from the CubeSat community proved invaluable both as inspiration and feedback on the chosen solutions. I would like to thank my co-supervisor Roger Birkeland, head of the NUTS project for suggesting the conference, assistance and opportunity. The paper is included in its entirety the electronic attachment.

The conference was in Rome in February of 2013 and had attendances from academia, industry and and ESA . The conference's web page can be found at `http://www.gaussteam.com/2nd-iaa-conference/`.

IAA-CU-13-13-01
Survey of correction methods for faults and errors induced by cosmic radiation on operating system level in CubeSats

Kjell Arne Ødegaard*, Amund Skavhaug†

Abstract

The NTNU Test Satellite (NUTS) project aims to build and launch a double CubeSat within 2014. The Cubesat is increasingly used as a low-cost platform for research and even commercial use. By investigating what is possible to accomplished with low cost components in terms of reliability, two important things can be achieved. The cost of using space platforms for research can be reduced while maintaining the required level of reliability and at the same time the number of non-functional satellites (i.e. problematic space junk) can be reduced.

The focus of this work has been to explore the use of consumer components in space and the impact of the harsh vacuum and radiation environment on these components. Although some approximations regarding long term radiation effects are investigated, the main focus have been intermediate radiation faults in processors and memory and how to best mitigate them.

This paper provides an overview of significant problems with consumer electronic components in a space environment and investigates these problems. It also presents a brief summery of the effects that causes these problems and how they are most likely to affect the finished system. Data from NASA is used in order to find an approximation of the expected fault intensity in the satellite's components.

A number of possible low-cost solutions to the presented problems are evaluated according to the assumptions in the error model. Checkpointing provides the system with a restorable safe state in the event of a restart. Error detection and correction in storage in storage systems provides safe storage of critical runtime variables. In addition, two levels of watch dog timers, program memory integrity check and a periodic full reset with power cycle to clear remaining errors.

Our design shall be able to mask the system's operation from even frequent resets while the Error Detection And Correction system prevent the expected hundreds of errors per day from accumulating in memory, at some point resulting in failures. To emulate the space environment for repeatable test runs on specific parts of the system a fault injection rig using JTAG is being constructed.

*Master Student, Department of Engineering Cybernetics, Norwegian University of Technology and Science, Norway, kjellaod@stud.ntnu.no

†Associate Professor, Department of Engineering Cybernetics, Norway, Norwegian University of Technology and Science, amund.skavhaug@itk.ntnu.no

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. The small standardized form factor where many satellites can be packed together an piggybacked to other launches keeps the launch costs low, and the satellites themselves often use Consumer Of The Shelf (COTS) electronic components.

The use of consumer 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 design without running big financial risks. This is interesting as it allows for development and advancement in an otherwise conservative industry.

Due to both cost concerns, availability and simplicity, CubesSats commonly use Consumer Of The Shelf, or COTS, components. A number of different factors that will be detailed further on in this paper make these components vulnerable to the environment in space and in this paper we explore measures to alleviate the impact of these factors to the reliability, availability and survivability of the design.

Problem

One of the main challenges for space applications is the hard radiation operating conditions [3] [4]. To solve these challenges, radiation hardened electronic components and fault tolerant hardware has been used is space systems for a number of years to either ensure error free operation or to mask the errors from the system. In the context of a CubeSat, however, the challenges of high reliability system design shifts. It is still desirable with a high reliability system but the budgetary constraints are much stricter than for commercial or government designs.

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 the same time it is not very important with a high availability design since the system does not control critical applications, but rather performs data collection tasks. This means that the on line redundant backup components can be omitted as long as we ensure that the system does not malfunction critically (i.e. fail). By using software methods, combined with 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.

This paper aims to investigate low-cost methods to increase mission lifetime of small COTS based satellites. When considering the different reliability measurements it is important not to impact the performance of the rest of the system to an unacceptable degree. By mitigating the effects of Single Event Phenomena (SEP) occurrences in non-hardened components, it is possible to ensure higher up-time and increased mission lifespan. 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 smart solutions and COTS hardware to have a usable system in extreme conditions.

The problems with COTS components in space is numerous, as detailed by NASA [3]. In brief, radiation effects known as SEP can occurs when cosmic radiation strikes certain parts of the semiconductor material as outlined by figure 1. If the cosmic ray has enough energy this 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 [3] is $10^{-5}\ errors/bit-day$. For NUTS 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 1000-10000 rad(Si)/year with an orbital inclination between 20 and 85 degrees [3]. NUTS will have an even higher inclination and thus even higher radiation levels can be expected. With the total dose failure level of flash memories from 5-15 krad(Si) and microprocessors for 15-70 krad(Si) [3] of radiation, both can during the satellite's mission lifetime.

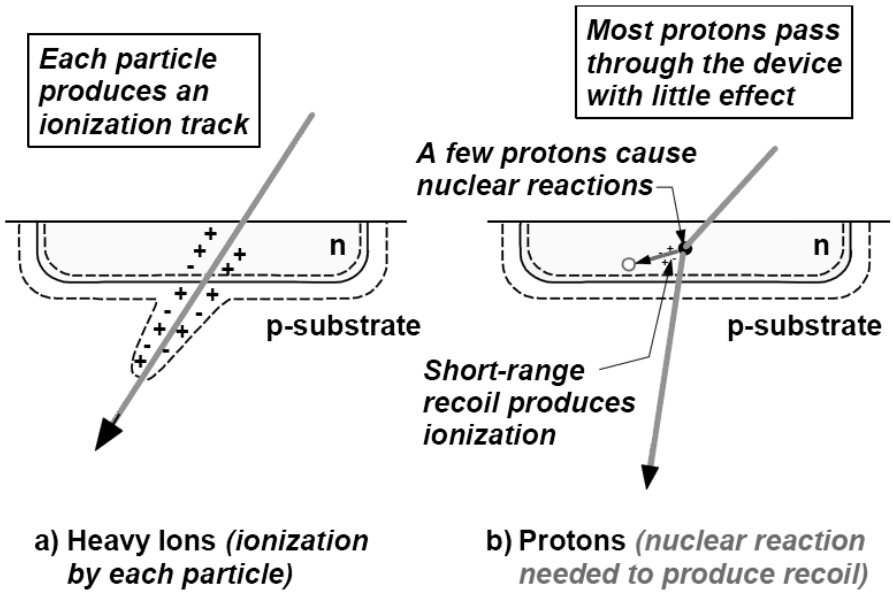


Figure 1: Mechanisms for Heavy Ion and Proton SEU effects [4]

Methods

The majority of the work in this paper has been to study the satellite and its systems and suggest solutions to the problems that are likely to be encountered.

Error Detection and Correction, EDAC

It would be advisable to have a system task that is in charge of secure storage of variables. Due to the random nature of the expected faults it would be difficult to determine if the data variables are safe to operate on. To guard against this it would be advisable to store the variables multiple times in order to be able to do a majority voting on the correctness or have an error correcting algorithm such as Bose-Chaudhuri-Hocquenghem (BHC) [2, p. 155] codes to correct the faults in run-time. This would add a large amount of complexity to the individual tasks running on the satellite. At the same time it is something that would be advisable for most of the tasks, due to the expected number of SEUs that will affect the system memory and accumulate over time.

A specialized secure storage system task could ease the programming burden for the rest of the designers by removing the sometimes complex algorithms from the smaller programs. 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].

Checkpointing

Checkpointing is a proven solution in software system redundancy. It works by storing the system state that is necessary for continued execution and completion of the process, at specific points during process execution [1, p. 214]. 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. An important feature to ensure is the ability to roll back multiple instances in the case of some unforeseen fault being present in the restored system.

Power cycling of faulty modules is already implemented in the backplane. The modules of the satellite must therefore tolerate a sudden reset without losing any significant amount of work. There are also some events such as antennae deployment and detumbling that should only be executed once and including these events in the saved system state will provide a simple measure of ensuring operational progress for the satellite.

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 module 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 JTAG in the software running on the board. This is somewhat better because the efficiency of the error correcting code can be determined directly since the number of inserted faults is known. Arguments against this testing regime are the lack of 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 while at the same time allowing for repeatable test runs and focus on specific parts of the system.

Other methods

In addition to EDAC and Checkpoint, 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 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 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.

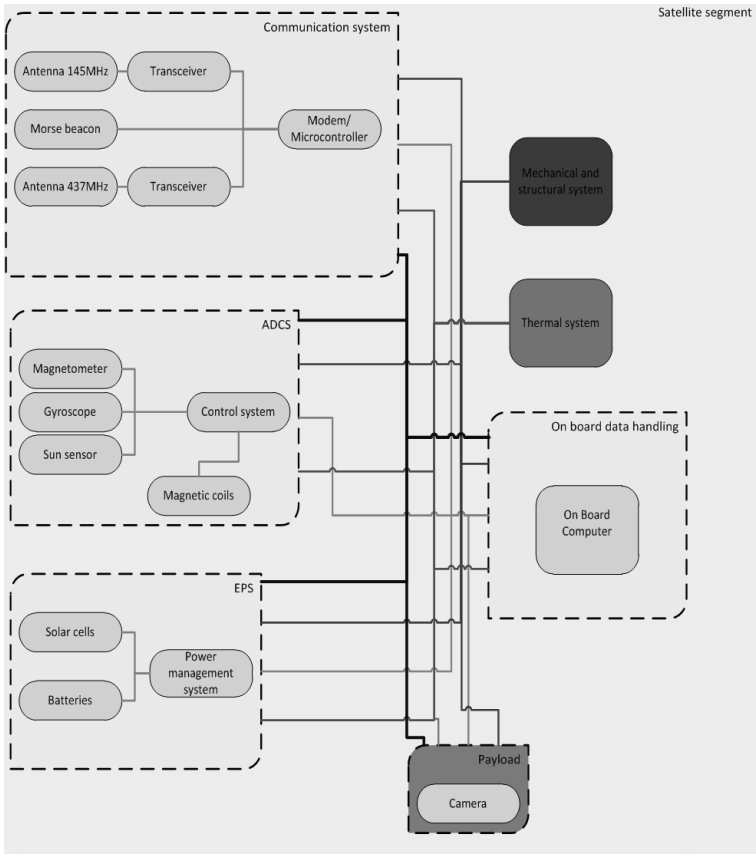


Figure 2: System overview [5]

Results and Observations

The preliminary results are encouraging. The chosen parameters for the $BCH(N, K)$ codes can detect and correct up to two randomly occurring errors per message block.

While the general implementations 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, the BCH codes can be optimized for embedded hardware. By choosing constant values for N and K the heavy computation of the generator polynomial coefficients can be done in advance. With the optimization 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.

For testing purposes, optimized $BCH(67, 53)$ codes have been implemented. However, the final parameters have to be adjusted based on how much computational power that is available in the finished system.

Discussion and Conclusion

The main focus of the work has been to study the satellite to date and present possible solutions in order to implement a 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 expected to affect the satellite is 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 strategy 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 is more powerful and uses much less power than the expensive processors 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.

The future work will focus on implementation of the solutions discussed in this paper. As more of the subsystems reaches 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 compressing algorithms. Because of this it is not advantageous to provide a finely tuned system at this point, but rather focus on a useful module for the future satellite. An exhaustive fault injection test to determine how the full system performs under stress is planned as the system reaches completion.

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] Sammy Kayali, *Space Radiation Effects on Microelectronics*, JPL NASA

[5] Emma Litzier, *System Overview - Space Segment*, http://nuts.cubesat.no/the-satellite, 2013.

Appendix C

The 5th European CubeSat Symposium

Part of the work in this thesis have been presented on two conferences thanks to NUTS and our sponsor NAROM. The contacts and new ideas obtained from the CubeSat community proved invaluable both as inspiration and feedback on the chosen solutions. I would like to thank my co-supervisor Roger Birkeland, head of the NUTS project for suggesting the conference, assistance and opportunity. The paper is included in its entirety the electronic attachment.

The conference took place in Brussels and had over 80 presentations, a poster session and multiple attendances from the industry as well as ESA. The conference's web page can be found at `https://www.cubesatsymposium.eu/`.

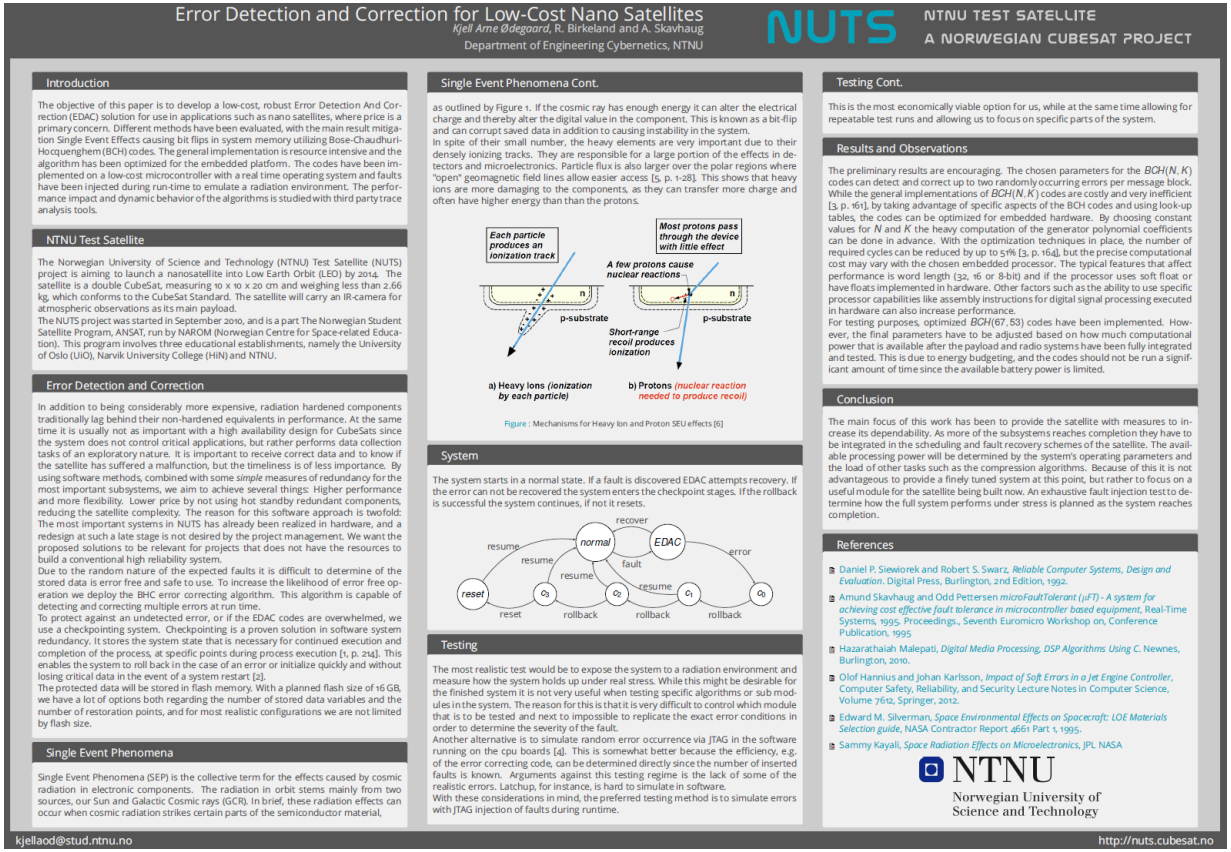


Figure C.1: Poster Overview

Introduction

The objective of this paper is to develop 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 optimized for 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.

NTNU Test Satellite

The Norwegian University of Science and Technology (NTNU) Test Satellite (NUTS) project is aiming to launch a nanosatellite into Low Earth Orbit (LEO) by 2014. The satellite is a double CubeSat, measuring 10 x 10 x 20 cm and weighing less than 2.66 kg, which conforms to the CubeSat Standard. The satellite will carry an IR-camera for atmospheric observations as its main payload.
The NUTS project was started in September 2010, and is a part The Norwegian Student Satellite Program, ANSAT, run by NAROM (Norwegian Centre for Space-related Education). This program involves three educational establishments, namely the University of Oslo (UiO), Narvik University College (HiN) and NTNU.

Error Detection and Correction

In addition to being considerably more expensive, radiation hardened components traditionally lag behind their non-hardened equivalents in performance. At the same time it is usually not as important with a high availability design for CubeSats 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. By using software methods, combined with some *simple* measures of redundancy for the most important subsystems, we aim to achieve several things: Higher performance and more flexibility. Lower price by not using hot standby redundant components, reducing the satellite complexity. The reason for this software approach is twofold: The most important systems in NUTS has already been realized in hardware, and a redesign at such a late stage is not desired by the project management. We want the proposed solutions to be relevant for projects that does not have the resources to build a conventional high reliability system.
Due to the random nature of the expected faults it is difficult to determine of the stored data is error free and safe to use. To increase the likelihood of error free operation we deploy the BHC error correcting algorithm. This algorithm is capable of detecting and correcting multiple errors at run time.
To protect against an undetected error, or if the EDAC codes are overwhelmed, we use a checkpointing system. Checkpointing is a proven solution in software system redundancy. It stores the system state that is necessary for continued execution and completion of the process, at specific points during process execution [1, p. 214]. 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 [2].
The protected data will be stored in flash memory. With a planned flash size of 16 GB, we have a lot of options both regarding the number of stored data variables and the number of restoration points, and for most realistic configurations we are not limited by flash size.

Single Event Phenomena

Single Event Phenomena (SEP) is the collective term for the effects caused by cosmic radiation in electronic components. The radiation in orbit stems mainly from two sources, our Sun and Galactic Cosmic rays (GCR). In brief, these radiation effects can occur when cosmic radiation strikes certain parts of the semiconductor material,

Figure C.2: Poster Column 1

Single Event Phenomena Cont.

as outlined by Figure 1. 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.
In spite of their small number, the heavy elements are very important due to their densely ionizing tracks. They are responsible for a large portion of the effects in detectors and microelectronics. Particle flux is also larger over the polar regions where "open" geomagnetic field lines allow easier access [5, p. 1-28]. This shows that heavy ions are more damaging to the components, as they can transfer more charge and often have higher energy than than the protons.

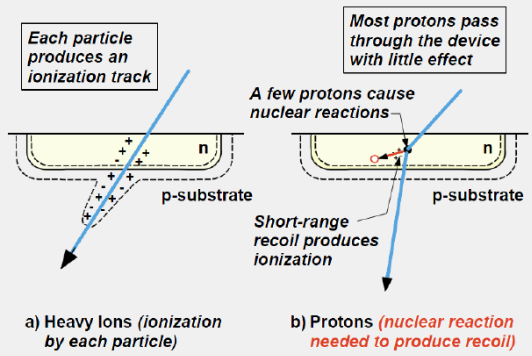
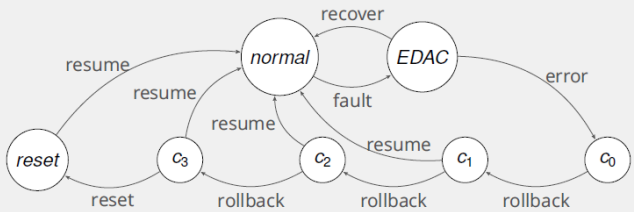


Figure : Mechanisms for Heavy Ion and Proton SEU effects [6]

System

The system starts in a normal state. If a fault is discovered EDAC attempts recovery. If the error can not be recovered the system enters the checkpoint stages. If the rollback is successful the system continues, if not it resets.



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 module that 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 JTAG in the software running on the cpu boards [4]. 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 of the 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.

Figure C.3: Poster Column 2

Testing Cont.

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.

Results and Observations

The preliminary results are encouraging. The chosen parameters for the $BCH(N, K)$ codes can detect and correct up to two randomly occurring errors per message block. While the general implementations of $BCH(N, K)$ codes are costly and very inefficient [3, p. 161], by taking advantage of specific aspects of the BCH codes and using look-up tables, the codes can be optimized for embedded hardware. By choosing constant values for N and K the heavy computation of the generator polynomial coefficients can be done in advance. With the optimization techniques in place, the number of required cycles can be reduced by up to 51% [3, 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 have floats implemented in hardware. Other factors such as the ability to use specific processor capabilities like assembly instructions for digital signal processing executed in hardware can also increase performance.
For testing purposes, optimized $BCH(67, 53)$ codes have been implemented. However, the final parameters have to 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 energy budgeting, and the codes should not be run a significant amount of time since the available battery power is limited.

Conclusion

The main focus of this work has been to provide the satellite with measures to increase its dependability. As more of the subsystems reaches 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.

References

- Daniel P. Siewiorek and Robert S. Swarz, *Reliable Computer Systems, Design and Evaluation*. Digital Press, Burlington, 2nd Edition, 1992.
- 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
- Hazarathaiah Malepati, *Digital Media Processing, DSP Algorithms Using C*. Newnes, Burlington, 2010.
- 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.
- Edward M. Silverman, *Space Environmental Effects on Spacecraft: LOE Materials Selection guide*, NASA Contractor Report 4661 Part 1, 1995.
- Sammy Kayali, *Space Radiation Effects on Microelectronics*, JPL NASA

NTNU
Norwegian University of Science and Technology

Figure C.4: Poster Column 3

Appendix D

SAFECOMP 2013

Part of the work in this thesis have been accepted for publication at the SAFECOMP 2013 Dependable Embedded and Cyber-physical Systems (DESC) workshop. The work done for and the reviews received during the process of writing the paper have been very beneficial. I wold like to thank my supervisor Amund Skavhaug for suggesting the conference and the advice and assistance that was provided during the process. The paper is included in its entirety in the electronic attachment.

The conference is held in Toulouse, the center for European aerospace industry, in September of 2013 and have attendances from academia and a special focus on industry and industrial applications. The conference's web page can be found at `http://conf.laas.fr/SAFECOMP2013/` and `http://safecomp.org/`.

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.

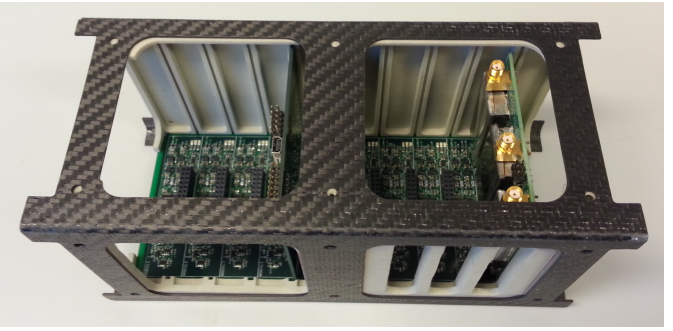


Fig. 1. NUTS - NTNU Test Satellite

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.

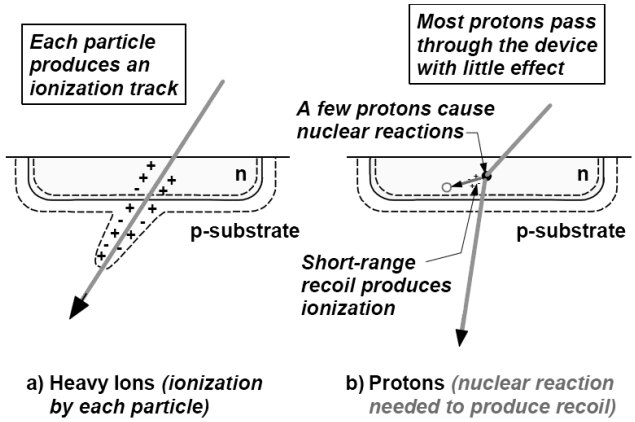


Fig. 2. Mechanisms for Heavy Ion and Proton SEU effects [5]

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.

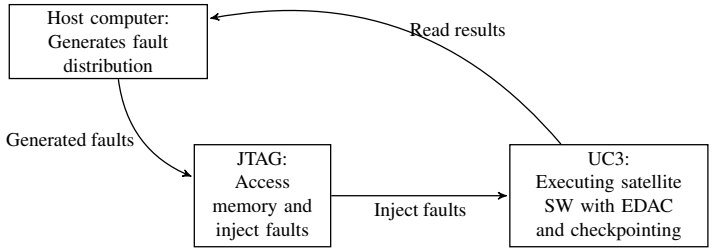


Fig. 3. Test Environment

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

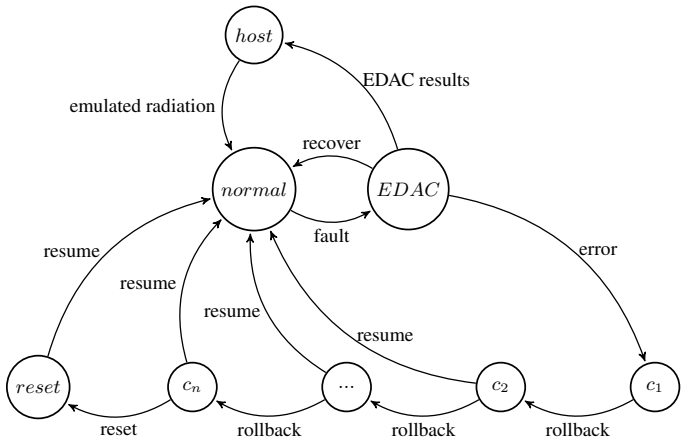


Fig. 4. Functionality Description

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.

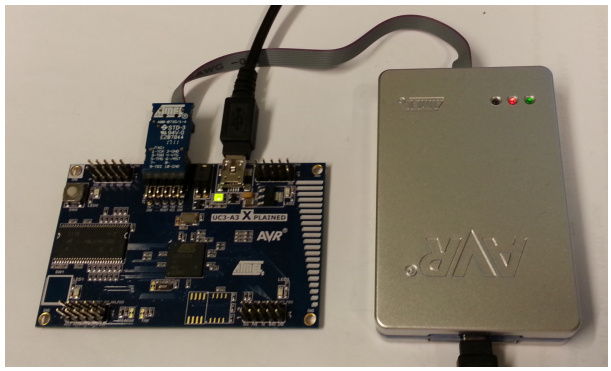


Fig. 5. UC3-A3 Xplained and JTAG ICE3

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

Appendix E

System Analysis of CSP with FSP and LTSA

This section is included from the previous work of the author regarding mathematical analysis of protocols. The work is also included in the electronic attachments.

4.2 System analysis of CSP with FSP and LTSA

4.2.1 Introduction

In order to determine to what degree it is possible to trust the Cubesat Space Protocol (CSP) network it was planned to analyze the protocol formally. CSP will be the main method of communication between the modules and also to some extent for internal communication between tasks running on the same module. CSP allows creating sockets for inter process and network communication similar to the ones used on Linux. This message based communication is preferable in a real time and reliability context due to the reduction of complexity in the program.

The use of a message based communication between tasks would require a reliable medium of communication. CSP is a relatively new protocol and although it has been deployed on a satellite it cannot yet be considered mature. In order to determine if this protocol is useful it is planned to device a process algebraic model of CSP and test this representation. A suitable approach for this is a Finite State Process (FSP) representation in the Labelled Transition System Analyser (LTSA) Java tool.

4.2.2 Implications

If CSP turns out to be reliable this would be useful for the further reliability design in the system. Barring failure in hardware the protocol itself could be said to be reliable and could be used safely for inter task communication.

The bus could still malfunction, but since there are two physically separate buses with their own bus repeaters on the backplane, these failures can be assumed to be quite rare. The expected number of SEUs in LEO are $10^{-5} error/bit-day$ which translates to roughly 10 errors per day in internal SRAM and 160 errors per day in external SRAM under normal conditions.

4.2.3 Documentation on CSP

The Cubesat Space Protocol implements a Reliable Datagram Protocol (RDP) in accordance with RFC-908. Further documentation can be obtained from [16].

4.2.4 Method

A process algebraic model of CSP was made with the LTSA tool and with help from supporting literature [2] [5]. The bulk of this work was to accurately translate the protocol from documentation and source code into a viable FSP model.

In order to create a system model there has to be some assumptions on which components will fail and how these failures will manifest themselves. This is called the system's error model. For the transmission line the error model was fairly simple.

There can be an error at the most every third transmission, and if an error occurs the two next transmissions will be error free. In Communicating Sequential Processes (a superset of FSP) notation this translates to:

$$E_0 = left?x \rightarrow (right!x \rightarrow E_0 \sqcap right!(1-x) \rightarrow E_2)$$
$$E_{n+1} = left?x \rightarrow right!x \rightarrow E_n \quad \text{for} \quad n = 0, 1$$

The theory and rationale behind this choice is available in [5, Sec. 5.1].

This might seem a bit simplistic, especially since errors often comes in bursts. A more complex error model could be constructed, but the solution would still be to retransmit lost transmissions until contact in achieved. The increased number of retransmits would require more power, something which is a limited resource on the satellite. With this in mind is seems more reasonable to limit the number of retransmits and wait until the ground station initiates another transmission.

4.2.5 Model Code

```
//Bounded buffer for CSP, limited to 5 packages
//error is bitwise error in the message

BUFFER_SEND(N=5) = COUNTS[0],
COUNTS[i:0..N] =
        (when (i<N) put_send->COUNTS[i+1]
        |when (i>0) get_send->COUNTS[i-1]
        ).
```

```
APPLICATION1 =
        (put_send -> send_complete -> APPLICATION1
        |put_send -> send_fail -> APPLICATION1).

SEND_CALL = (get_send -> send -> ack -> send -> ack
        ->send -> ack -> send_complete -> SEND_CALL).

BUS (N=2) = BUS[0],
BUS[i:0..N] =
        (when (i==0) send -> recv -> BUS[0]
                | error -> BUS[2]
        |when (i>0) send -> recv -> BUS[i-1]
        ).

BUFFER_RECV(N=5) = COUNTR[0],
COUNTR[i:0..N] =
        (when (i<N) put_recv->COUNTR[i+1]
        |when (i>0) get_recv->COUNTR[i-1]
        ).

RECV_CALL =
        (recv -> ack -> put_recv -> recv_complete ->
                RECV_CALL
        |error -> recv -> ack -> put_recv ->
                recv_complete -> RECV_CALL).

APPLICATION2 =
        (recv_complete -> get_recv -> APPLICATION2).

||CSP = (APPLICATION1 || BUFFER_SEND(5) || SEND_CALL ||
        BUS ||
        RECV_CALL || BUFFER_RECV(5) || APPLICATION2).
```

4.2.6 Results

The result of this study of CSP shows that the implemented model of the protocol is deadlock and livelock free. Beyond that, the LTSA tool does not

provide any insights or analysis of the system.

4.2.7 Discussion

The model was based on the system described in the documentation and on the provided source code. In the process of building a model based on an already implemented system it is difficult to make any guarantee that the process algebra that have been created from the source code is an accurate description of the system.

The protocol is also open source and this is why NUTS is able to use it in the first place. In order to be able to make a definitive statement of the reliability of the implemented code one would need to deploy some sort of automatic translation tool. Otherwise it would prove very time consuming to maintain a correct model of CSP.

The two other teams working with the implementation of CSP in the project ran into some problems. They fixed these problems by making changes to the code for the radio and the satellite bus in order to have a functional system. The result of these changes is that the implemented process model is no longer valid, and the drawn conclusions regarding the validity and reliability of the protocol has to be re-evaluated.

4.2.8 Conclusion

The implemented model of CSP was deadlock and livelock free with the provided error model, and the specification of the protocol seems to be robust from this standpoint.

However, due to changes in the implemented protocol from various groups in the project, the model used in the verification is no longer valid. It is difficult to get an accurate representation of the implemented system in the process model and the extra work needed is not deemed a reasonable endeavor.

Formal verification methods are most useful in the specification phase and the techniques used on CSP might prove useful in the specification of the planned reliability measures for the satellite.

The final word on the conclusion is that the model that was used when designing the specification for the protocol is valid.