Fault Tolerant Cryptographic Primitives for Space Applications

by

Marcio Juliato

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Spacecrafts are extensively used by public and private sectors to support a variety of services. Considering the cost and the strategic importance of these spacecrafts, there has been an increasing demand to utilize strong cryptographic primitives to assure their security. Moreover, it is of utmost importance to consider fault tolerance in their designs due to the harsh environment found in space, while keeping low area and power consumption. The problem of recovering spacecrafts from failures or attacks, and bringing them back to an operational and safe state is crucial for reliability. Despite the recent interest in incorporating on-board security, there is limited research in this area. This research proposes a trusted hardware module approach for recovering the spacecrafts subsystems and their cryptographic capabilities after an attack or a major failure has happened. The proposed fault tolerant trusted modules are capable of performing platform restoration as well as recovering the cryptographic capabilities of the spacecraft. This research also proposes efficient fault tolerant architectures for the secure hash (SHA-2) and message authentication code (HMAC) algorithms. The proposed architectures are the first in the literature to detect and correct errors by using Hamming codes to protect the main registers. Furthermore, a quantitative analysis of the probability of failure of the proposed fault tolerance mechanisms is introduced. Based upon an extensive set of experimental results along with probability of failure analysis, it was possible to show that the proposed fault tolerant scheme based on information redundancy leads to a better implementation and provides better SEU resistance than the traditional Triple Modular Redundancy (TMR). The fault tolerant cryptographic primitives introduced in this research are of crucial importance for the implementation of on-board security in spacecrafts.

Acknowledgements

My most sincere thanks for my wife Rita, for all her love, confidence and respect. My honest admiration for your strength while dealing with all the difficulties that have crossed our ways. Thanks for being such a generous and courageous person, as well as for facing head-on the next challenges that are approaching. Your collaboration and support were crucial in allowing me to pursue my PhD. I consider myself a lucky person to share my life with you. For you, my dear, my immense gratitude and love.

Special thanks to my parents Salete and Roberto for their infinite love, tireless dedication, insightful guidance, and immense comprehension. My profound admiration for the respectful and inspiring environment that I have been immersed since I know myself as a person. Thanks for instigating my curiosity and supporting me in all my endeavors (even in my craziest experiments). Thanks for teaching me how to have solid references and how to follow good principles. Without your support I would never have the chance to pursue my dreams and achieve my objectives. Thanks for my family, specially to my grandma Izabel and my sisters Vanessa and Denise, for their constant support even thousands of kilometers away.

Sincere thanks to my supervisor, Prof. Cathy Gebotys, for kindly showing me the horizon when I was looking at my own feet. Thanks for all confidence deposited in my person and in my capabilities. Also, thanks for your constant kindness and for receiving me with a smile in every single time that we met. Without your generous support, my PhD would never had become a reality. Kind thanks to my committee members, namely Professors Alfred Menezes, Andrew Kennings, Anwar Hasan, and Kris Gaj, for peer-reviewing and providing valuable comments on my thesis.

Thanks to the University of Waterloo and the Department of Electrical and Computer Engineering for the fabulous infrastructure and the many research and teaching opportunities that I have been offered while pursuing my PhD. Besides, my kind thanks to University of Campinas, Brazil, and to my previous supervisors, Professors Guido Araujo, Julio Lopez, and Paulo Centoducatte. I am really thankful to all institutions and educators that conducted me in the process of learning how to learn, specially for introducing me to critical thinking, skepticism, and scientific methodology.

Last but not least, thanks for my colleagues at the Embedded Security Lab, namely, Reouven Elbaz, Edgar Mateos, Dave Kenney, Solmaz Ghaznavi, Patrick Longa, Farhad Haghihgizadeh, Amir Khatibzadeh, and Brian White. I really appreciate your friendship, company, and discussions in the lab. Also, thanks to all my friends, some nearby, others far away, for all their support and for all journeys we have taken throughout all these years.

Dedication

This PhD thesis is dedicated for meritorious people that could not pursue their dreams and achieve their objectives. Not because of their intellectual capacity or motivation, but due to deficient health, disabilities, inadequate alimentation, peace deprivation, indecent quality of life, inappropriate conditions of study and work, socio-economic problems, natural catastrophes, wars and conflicts, despotism, prosecution, and discrimination in its various forms. This thesis is also dedicated to those people who, even against all odds and adversities, continue to work hard in the name of science, knowledge and free thinking.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
AsiaSat	Asia Satellite Telecommunications Co. Ltd
ASIC	Application Specific Integrated Circuit
BRAM	Block RAM
CCSDS	Consultative Committee for Space Data Systems
CCTV	China Central Television Station
CLA	Carry Look-Ahead Adder
CLB	Configurable Logic Block
CRC	Cyclic Redundancy Check
CSA	Carry Save Adder
DOD	United States Department of Defense
DMR	Dual Modular Redundancy
DSS	Digital Signature Standard
ECSS	European Cooperation for Space Standardization
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
ESA	European Space Agency
FAA	Full Adder Array
FPGA	Field Programmable Gate Array
GAO	United States General Accounting Office
GEO	Geostationary Orbit
GLONASS	Global Navigation Satellite System
GPL	GNU General Public License
GPS	Global Positioning System
GSO	Geosynchronous Orbit
HEO	High Earth Orbit
HMAC	Keyed-Hash Message Authentication Code
IKE	Internet Key Exchange
I/O	Input/Output
ISO	International Organization for Standardization
JPL	Jet Propulsion Laboratory
JTAG	Joint Test Action Group
KEK	Key Encryption Key
LE	Logical Element

LEO	Low Earth Orbit
LTTE	Liberation Tigers of Tamil Eelam
LUT	Look-Up Tables
MAC	Message Authentication Code
MBU	Multiple Bit Upset
MEO	Medium Earth Orbit
MoD	British Ministry of Defense
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standards and Technology
NRE	Non-Recurring Engineering
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
RNG	Random Number Generator
RF	Radio Frequency
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SEB	Single Event Burnout
SEE	Single Event Effect
SEFI	Single Event Functional Interrupt
SEGR	Single Event Gate Rupture
SEL	Single Event Latchup
SET	Single Transient Effect
SEU	Single Event Upset
SKI	Secret Key Infrastructure
SRAM	Static RAM
SDRAM	Synchronous Dynamic RAM
TCG	Trusted Computing Group
THCM	Trusted Hash and Configuration Module
TID	Total Ionizing Dose
TKM	Trusted Key Recovery Module
TM	Trusted Module
TMR	Triple Modular Redundancy
TPM	Trusted Platform Module
TRM	Trusted Reset Module
TT&C	Telemetry, Tracking and Command
UK	United Kingdom
U.S.	United States
USA	United States of America

Epigraph

I do not know what I may appear to the world,
but to myself I seem to have been only like a boy playing on the sea-shore,
and diverting myself in now and then finding a smoother pebble
or a prettier shell than ordinary,
whilst the great ocean of truth lay all undiscovered before me.

Isaac Newton (1643-1727)

Chapter 1

Introduction

This chapter presents the motivation that led to the development of this research, and lists the main contributions resulting from this thesis. It also provides the reader with a summary of each chapter that comprises the remainder of this document.

1.1 Motivation

Spacecrafts are extensively used by public and private sectors to support a wide variety of services. For instance, they provide communication services, support navigation and meteorological services, allow for scientific experiments to be conducted, and are even utilized to increase homeland security. Some countries also employ non-military satellites to increase their military communication capabilities. For instance, during the Desert Shield and Desert Storm operations the United States (U.S.) Department of Defense (DOD) used commercial satellites to perform 45% of all communications between the U.S. and the Persian Gulf [140]. With regard to the costs involved, the space sector is a multi-billion business. According to a report from the U.S. General Accounting Office (GAO) [140], the commercial satellite industry generated $85 billion in revenue in 2000. The Galileo program [54] from the European Space Agency (ESA) [53] had an initial cost of 3.4 billion Euros [52], with a recent request of additional 1.9 billion Euros [125] to complete the program. Galileo is estimated to have an annual market for services and equipment of 200 billion Euros by 2013 [52].

Considering the numbers presented above, the disruption of satellite services, whether intentional or not, can have a major economic impact. For example, in 1998 a failure

of the Galaxy IV satellite disrupted about 80 to 90 percent of 45 million pagers across the U.S. for 2 to 4 days [140]. This failure also blocked the authorization of credit card transactions at points of sale such as gasoline pumps. With the growing worldwide demand for satellite-based services, the dependence on these spacecrafts tends to increase and so do the financial losses in case of failure. As a result, spacecrafts must perform with high reliability and availability.

The construction, launch and operation of spacecrafts are usually carried out by space agencies and contractors. Details of such activities are traditionally not made available to the general public, leading to the false impression that third parties cannot tamper with or compromise spacecrafts after launch. However, as mentioned in [24], advances in technology allow for more complex attacks to be easily launched against space systems. As a result, it is definitely not a good strategy to rely on obscurity and uniqueness to assure the security of spacecrafts. Anti-jamming techniques are used to avoid interference with communications. However, strong security becomes mandatory if the spacecraft is to be protected against attackers. For example, if security is not addressed properly, an attacker could possibly perform replay attacks, intercept proprietary data, send invalid commands, or masquerade as a legitimate user.

Due to the current lack of appropriate countermeasures to protect spacecrafts, the probability of success may be high if some individual, group, or organization put effort in compromising space systems. Not only satellites orbiting Earth are at risk, but also deep space spacecrafts and probes. For instance, several TV transmissions based on satellites have already been attacked and hijacked [103]. Some of them were used to transmit adult content to inappropriate audiences, while others transmitted propaganda and protest material [138, 124, 49, 142, 153, 113, 141]. Although these incidents were related to the tampering of the satellites' data link, the incident can become even more serious if the Tracking, Telemetry and Control (TT&C) link is targeted. If the TT&C link is attacked, the spacecraft may be put out of its orbital path, dropped down to Earth, or compromised in such a way that it can become nothing else than space junk. For instance, reports indicate that attackers took control of a British satellite and demanded money to give the control back to the operators [56, 72, 126].

Since countries rely heavily on commercial satellites for communication, threats to those spacecrafts have the potential of putting a nation's critical infrastructures at risk [140, 145, 14]. For instance, at the time the United Kingdom (UK) Ministry of Defense (MoD) satellite was attacked, Margaret Beckett, leader of UK House of Commons, warned about the big impact that electronic attacks can have in Britain's communications infrastructures. She complemented: "Hijacking a satellite is one of the first activities in an infowar attack". Therefore, as pointed out by the U.S. GAO [140], it is evident that security of spacecrafts

should be more fully addressed.

The implementation of strong security mechanisms in spacecrafts is not straightforward [23]. One of the difficulties encountered is the protection of security mechanisms from the harsh environment found in space. Radiation originating from both the Sun and deep space is the best documented cause of a class of errors known as Single Event Upsets (SEUs) [107]. SEUs are forms of soft errors, i.e., they are errors that occur in the circuitry of a system causing bit-flips, but are not damaging to the hardware. According to a study of on-orbit spacecraft failures [168], 45% of failures happen due to electrical reasons. Of all subsystems failures, 27% happen in the telemetry, tracking and control (TT&C) as well as in the command and data handling subsystems (CDH), mainly due failures in control processor 26%, in electric circuitry 17%, and computer resets 7%. Considering these two subsystems, the TT&C alone is responsible for more than 70% of the failures.

Although SEUs are of great concern in aerospace applications, some applications can comfortably tolerate bit-flips. On the one hand, a photo taken by an Earth observation satellite may not get corrupted if a bit-flip happens in a storage element holding one of its pixels. On the other hand, cryptographic computations are very sensitive with respect to SEUs, so that a bit-flip in cryptographic data is enough to cause the cryptographic computation to fail. For example, a bit-flip in early rounds of AES can cause 50% of the bits of the result to be erroneous [17]. As reported in [168], about 35% of the failures that happen on TT&C and CDH leads to the loss of the mission whereas 60% of them leads to the mission degradation. Therefore, the fault tolerant implementation of cryptographic primitives in the aforementioned spacecraft subsystems is crucial to ensure that mission reliability is not decreased by the utilization of security mechanisms.

Once launched, spacecrafts follow their orbital or deep space paths. The duration of their operational lifetimes, in turn, may vary from a couple of months to decades. Therefore, they must be capable of operating for long periods of time without physical maintenance. An exception to this case is the Hubble Space Telescope [120], which was serviced four times since it began its operation in 1990. In all those occasions, the Space Shuttle [123] was used. Since the average cost to launch the Space Shuttle is $450 million [123], post-launch maintenance is performed only in rare exceptions (not to mention the retirement of the Space Shuttle program in 2011). Post launch maintenance is not the case for the whole majority of spacecrafts. As a consequence, it is crucial to design fault tolerant systems, so that the effects of radiation on circuits do not cause critical failures. Such failures could disrupt the normal operation of the spacecraft or even cause its loss.

Furthermore, the fact of employing rockets to launch spacecrafts into space imposes very stringent restrictions on the hardware size and weight. Also, given that spacecrafts

are usually powered by solar panels and re-chargeable batteries, there are heavy restrictions on power consumption of their electrical subsystems. Additionally, heat transfer in space is solely carried out by conduction and radiation. The lack of convection imposes severe constraints on power dissipation of on-board circuitry. Finally, given the tight budget which is very often imposed on space agencies, the cost of embedded components must also be considered when building spacecrafts. As part of the computational subsystem of spacecrafts, cryptographic primitives used in space applications must take all these constraints into account while being designed and implemented.

1.2 Contributions

Given the range of threats against spacecrafts and the multiple constraints imposed by space systems, this research has the main goal of proposing fault tolerant security mechanisms for space applications. Specifically, main problems being addressed in this research include:

1. How to recover a spacecraft from an attacker who has gained control over its computational platform?
2. How to perform a secure restoration of the spacecraft's cryptographic capabilities in face of failures and attacks?
3. How to maximize security in the recovery mechanism and how to quantify it?
4. How to achieve efficiency in fault tolerance mechanisms for cryptographic primitives tailored to space applications so that they can cope with radiation-induced faults and demand minimum implementation requirements?
5. How to determine the resistance against of SEUs of fault tolerance mechanisms and how to compare different schemes?

In face of the aforementioned questions, the first goal is to propose schemes to recover the spacecraft and its cryptographic capabilities after major failures or attacks have occurred. A major failure can result from SEUs, a power outage, and/or a temperature variation. Since cryptographic data is quite sensitive to bit-flips, an SEU in cryptographic computation will certainly lead to failures. Furthermore, even when space systems implement security mechanisms, attackers can potentially exploit security holes to break into

the system. In that case, there should still exist some means to bring the spacecraft control back to the ground operators. More precisely, it is necessary to detect when attacks or failures have happened, bring the spacecraft to a safe state, and re-establish a secured communication channel with the control center.

The second goal relates to proposal and evaluation of fault tolerant schemes for integrity and authentication schemes. More specifically, the main focus is on the Secure Hash Algorithm (SHA-2) [133] and on the keyed-hash message authentication code (HMAC) [132] based on SHA-2. Hash functions can be applied to space systems in many different ways. It could be employed in invasion detection and recovery schemes [88] to determine, for example, whether an attacker, who may have broken into the spacecraft, has tampered with the system's program memories or the configuration of Field Programmable Gate Arrays (FPGAs). Integrity and authentication are also of utmost importance in secure communications. For instance, they are used to certify that the data received from a control center was not modified and came from a valid control center (not from an attacker). As a result, it becomes possible to assure, for example, that maneuvering commands are from a legitimate operator and that they were not accidentally or maliciously compromised. Authentication mechanisms are fundamental when updating the spacecraft hardware and software in case of attacks and failures. Moreover, the resistance against SEUS of each fault tolerant technique proposed for SHA-2 and HMAC was evaluated through a probability of failure analysis. Such an analysis allowed for the comparison of the proposed schemes against well established for space systems such as Triple Modular Redundancy (TMR).

1.3 Thesis Overview

The remainder of this thesis is organized as follows:

Chapter 2 provides a short introduction to fault tolerance and space systems. It includes fundamental concepts, commonly used definitions, and an introduction to radiation effects on electronic circuits. Also, technological, architectural and recovery techniques for mitigating SEUs are presented. In addition, several aspects of space systems are described, such as spacecraft subsystems, space missions types and security levels. Several threats against space systems are reviewed followed by a number of known attacks that have been reported during the last couple of decades. Moreover, a set of security requirements for space systems is presented, most of them based on standards and recommendations of the Consultative Committee for Space Data Systems (CCSDS).

Chapter 3 comprises the state-of-the-art in security for space systems. It includes some traditional SEU mitigation techniques that have been used in space applications. Next,

previous research on hardware implementation of cryptographic primitives is presented. Although most of them do not consider fault tolerance, they provide ideas and optimization techniques that can be adopted in conjunction with mitigation schemes. Some proposals of fault tolerant security mechanisms that could be utilized in space systems are also presented in this chapter.

Chapter 4 presents proposals for fault tolerant system recovery. Specifically, the proposed approach rely on a trusted platform along with a challenge-response protocol to recover spacecrafts in case of major failures and attacks followed by the re-establishment of their cryptographic capabilities. The system recovery process relies on the computation of integrity checks, which in turn employ fault tolerant hash functions.

Chapter 5 introduces hardware/software approaches tailored for SHA-2 and HMAC algorithms. Several levels of partitioning are explored, which includes simple operations implemented as custom instructions as well as entire SHA-2 and HMAC algorithms implemented as peripherals. Through experimental results based upon a NIOS2 processor it was possible to precisely determine the gains of performance in face of the increase in the system implementation area.

Chapter 6 proposes efficient fault tolerance mechanisms targeting SHA-2 hash functions. Several hardware design architectures are evaluated in hardware implementations based on FPGAs. Furthermore, implementation aspects of each approach are discussed, such as implementation area, memory requirements, power consumption, frequency of operation, and throughput. As a result, it was possible to show that the proposed scheme based on information redundancy provides fault tolerance with considerable savings in terms of implementation area, memory requirements and power consumption.

Chapter 7 explores the fault tolerant technique based on information redundancy to HMAC. Again, several hardware designs were proposed and analyzed using FPGAs. Experimental results have shown that the proposed technique provides even better implementation results for HMAC, which has a more complex architecture compared to SHA-2.

Chapter 8 introduces a probabilistic analysis in order to determine the robustness of each fault tolerance mechanisms in the presence of SEUs. A systematic analysis of the resistance against of SEUs is a crucial complementary step towards a more comprehensive evaluation the fault tolerant cryptographic modules. Through the determination of the probability of failure of each module, it was possible to directly compare the different fault tolerance approaches considered in this research, including the widely used TMR.

Chapter 9 discusses the contributions of this research and compares them with related work. Additionally, this chapter presents the conclusions of this thesis and future work.

Chapter 2

Background

This chapter comprises three main sections. The first one aims at providing background information on fault tolerance, which includes fundamental concepts and definitions, radiation effects on electronic circuits, as well as SEU mitigation techniques. The second section provides an overview on space systems, which comprises spacecraft subsystems, types of space missions and associated threats, security requirements, and hardware platforms in space. The last section introduces the SHA-2 and HMAC algorithms which are further utilized in the following chapters.

2.1 Fault Tolerance

Nowadays, computer systems are employed in a wide variety of tasks, varying from text editors and web browsers to embedded control of devices, machines and spacecrafts. On the one hand, when a text editor crashes, it may not cause much more than frustration and limited loss of data. On the other hand, a failure in a medical, military or aerospace system can pose threats to human life and national security, as well as cause environmental disasters and severe losses. Consequently, these critical tasks demand the employment of fault tolerant computer systems.

2.1.1 Fundamental Concepts

There are three fundamental concepts in the fault tolerant field: fault, error and failure. A fault refers to the occurrence of events in hardware and software, such as the alteration of

a physical property, a manufacturing imperfection, or a programming bug. Errors can be defined as consequences of faults, which cause the deviation of a given information from its correct value. A failure is a consequence of an error and is manifested as a corrupted execution of a given task [87].

Alternatively, faults, errors and failures can be explained by considering three universes, namely physical, informational and external [87]. The physical universe is constituted of physical entities, such as semiconductor devices. A fault is then defined as the physical alteration or defect of a component in the physical universe. On top of that comes the informational universe, which comprise pieces of information, such as a byte (or any other data unit). When a piece of information gets corrupted, it is said that an error has occurred. Finally, there is the external universe, where the consequences of errors show up. The incorrect result or processing, caused by errors, is called a failure.

To illustrate the differences between those three concepts, consider a high energy particle hitting a silicon chip. This collision could upset the charge balance on silicon causing a transistor to switch off. This alteration of physical properties can be denoted as a fault. If the affected transistor is used to implement a register, some stored information can be altered. The consequence is going to be an error in the data stored in the register. Next, by using this corrupted register, a cryptographic algorithm will certainly perform, for example, an erroneous signature check therefore causing a failure.

2.1.1.1 Design Goals

Systems are designed with multiple goals in mind, where functionality and performance are the more common ones. However, some other requirements are very often found in space applications, such as reliability, availability, and safety. The inclusion of fault tolerance into the system can help designers to meet the aforementioned requirements.

Reliability is defined as the probability that a system will perform its designed tasks for a certain period of time, say $[t_0, t]$, assuming that it was perfectly functional at t_0. During $[t_0, t]$ no incorrect performance or maintenance is acceptable. In space applications, t can be in the order of decades. Fault tolerance can improve system reliability by keeping it functional when hardware and software failures occur. This can be achieved, for example, through the use of redundant units, so that the system can operate even when one or more units are not functional.

Availability is the probability that a system is operational at the instant of time t. That is to say that the system has a very narrow time frame to recover from failures. Hence, it is possible to have frequent, but short, periods of inoperability and still achieve high availability. The use of spare units, for example, favors high availability, if tasks can be transfered from one unit to the other in a short time frame.

Safety refers to the capability of the system to fail in a safe manner. It is defined as the probability that the subsystem will function correctly or will stop working in such a way that it does not disturb the rest of the system. Detecting subsystem failures and disrupting its abnormal functioning is fundamental for achieving safety.

2.1.2 Radiation Effects

Radiation effects on electronic chips can vary from temporary malfunction to permanent damage [121]. They can suffer from Total Ionizing Dose (TID) Effects as a result of a long exposure to radiation. Also, they can be caused by a single energetic particle and then are denoted as SEEs. TID effects are consequences of accumulating ionizing damage, which causes devices to suffer with, for example, threshold shifts, increased current leakage and altered functional timing. They are mainly caused by protons and neutrons, usually found in Solar Wind, in the Van Allen Belt, and in Cosmic Rays [121]. TID effects can be reduced through device shielding, but since that involves the device manufacturing, it is out of the scope of this research. SEEs are caused by high-energy subatomic particles and electromagnetic waves, which can come from many different sources. High-energy subatomic particles, such as electrons, protons, neutrons, alpha particles (Helium nuclei), heavy ions (heavier than Helium), commonly found in Solar Wind, in the Van Allen Belt, and in Cosmic Rays. Electromagnetic waves, such as X-rays and gamma rays can be found in cosmic rays and in radioactive decay. The interaction of cosmic rays with Oxygen and Nitrogen present in the Earth's atmosphere also generate neutrons as well as other secondary particles such as pions and muons.

SEEs can happen at the ground level, aircraft altitudes, and in space [102]. At the ground level, they are mainly caused primarily by neutrons and secondarily by protons. At the sea level, for example, an average flux of 20 neutrons/cm^2/hour can be found. However, not only radiation coming from space interferes with applications at the ground level. The packaging material of chips may contain radioactive impurities, whose radioactive decay emits particles with the potential of causing SEEs. Airborne applications face higher levels of neutrons flux, more specifically, 7,200 neutrons/cm^2/hour. Therefore, they are

more prone to suffer SEEs than applications on the ground. Nevertheless, ground level and airborne applications are naturally protected by the Earth's atmosphere. Radiation coming from space, such as gama rays, X-rays, protons, alpha particles and heavy ions, are mainly absorbed by the Earth's atmosphere. That implies that spaceborne applications suffer more severely from its direct exposure to radiation coming from space. Although the Earth's magnetosphere deflects most of the Solar Wind, some radiation are still able to reach Earth's atmosphere and surface in polar regions. Besides, the magnetosphere traps protons, electrons, and other nuclei such as alpha particles. This region of trapped particles is called Van Allen Belt. As a consequence, spacecrafts operating past the Earth's magnetosphere must deal with higher levels of radiation, either trapped in the Van Allen belt or coming from space.

When subatomic particles and electromagnetic waves carry enough energy, they can displace electrons, atoms and molecules. This process is referred to as ionization. As pictured in Figure 2.1, an ionizing radiation transversing a silicon chip creates a track of electron hole pairs. The consequence is a charge accumulation, which might become a transient current pulse (often called a glitch). This phenomenon is denoted as Single Transient Effect (SET).

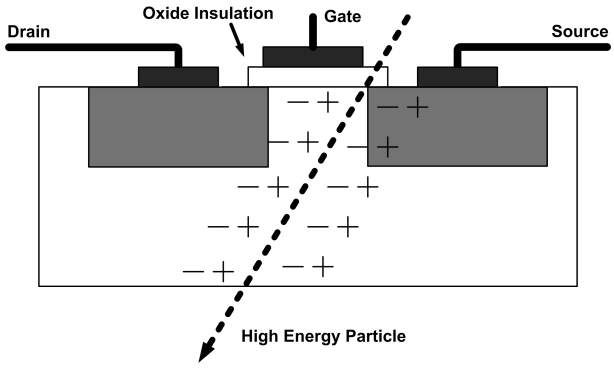


Figure 2.1: Ionization of a Silicon Surface

Advances in technology have made it possible to build transistors with reduced dimensions, thinner gate oxides, and lower operation voltages. As a consequence, information can be represented or stored using electric signals with reduced intensity. As a result, particles that once used to be innocuous, can now have higher potentials to cause upsets. In other words, as transistor sizes scales down, they become more sensitive to SEEs.

Several types of SEEs can happen within digital circuits [5]. If a transistor has a parasitic PNPN structure, radiation effects can cause the parasitic transistor to turn on

the primary one indefinitely. More specifically, this transistor pair acts like a thyristor. This phenomenon is called Single Event Latchup (SEL). The only way to turn off the induced thyristor is to power cycle the device. Another phenomenon happens when a transistor is turned on by radiation and its drain-source voltage is higher than the device's breakdown limit. The consequences are excessive drain-voltage currents and overheating, therefore causing the transistor burn out. This event is denoted as Single Event Burnout (SEB). Yet another effect caused by radiation is called Single Event Gate Rupture (SEGR). It happens when ionizing radiation hits the transistor at the same time a high voltage is applied its gate. The result is the rupture of the gate's dioxide insulation, followed by overheating and destruction of the gate region. Likewise SEBs, SEGRs causes permanent damage to the transistor. Nevertheless, the more common consequences of SETs are SEUs.

2.1.2.1 Single Event Upsets

An SEU can be defined as a change of state in storage elements as a consequence of ionizing radiation. Since digital computers manipulate information in its binary representation, a change of state is basically a bit-flip in registers or memory elements. SEUs are considered soft errors in the sense that they do not cause permanent damage to the circuit. Actually, the effects of SEUs over registers and memory can be reversed by rewriting the correct information to the storage element. Given that an SEU is a more common consequence of an SET, the former is of main interest in this research. In order to adopt countermeasures to mitigate SEUs in digital circuits, it is necessary to understand how SEUs are generated. SETs can happen in combinational and sequential logic of digital circuits.

A typical digital circuit configuration very often found in sequential circuits is illustrated in Figure 2.2. It consists of a register feeding a combinational logic, which in turn writes its results back to a register. If an SET happens in a register, as depicted in Figure 2.2 (a), it can cause its transistors to change state therefore flipping the stored bit. In this case, it is said that an SEU happened directly in the register. As a consequence, the combinational logic will receive an incorrect input value coming from the registers. SETs can also occur in a transistor of the combinational part of the circuit, as shown in Figure 2.2 (b). The inversion of a transistor state will cause a flipped signal to propagate through the combinational logic. Eventually, this corrupted output reaches the input of a register. Next, if setup and hold times of the register are met, the input will be sampled at the rising (or falling) edge of the clock. The result is an incorrect computation being stored within a register, i.e. an indirect register SEU occurring through the combinational logic.

Similarly, SETs can cause SEUs in SRAM memory cells. Figure 2.3 (a) shows a simplified SRAM memory cell, consisting of four transistors, $T1$, $T2$, $T3$, and $T4$, as preseted

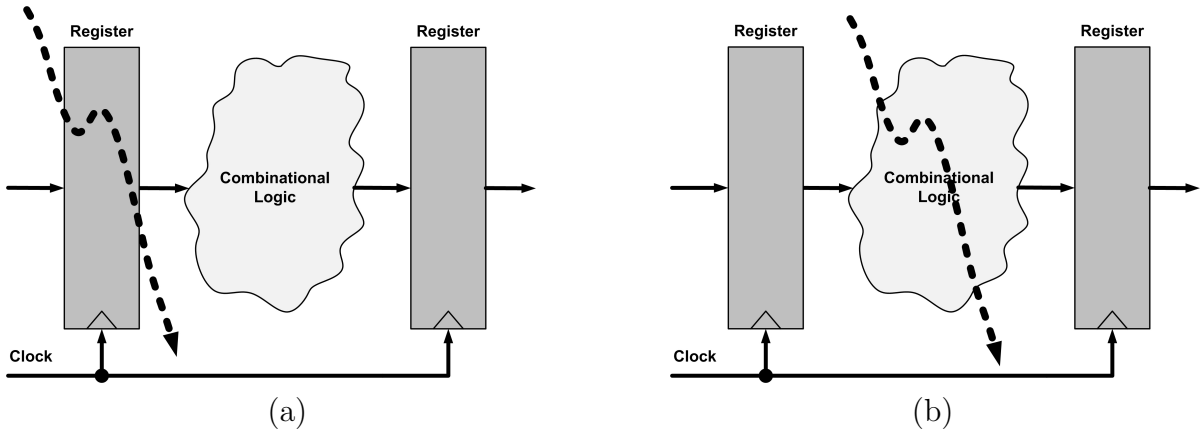


Figure 2.2: SET in Sequential and Combinational Logic

in [102]. Depending on the states of the transistors, the cell can store a '0' or a '1'. Assuming that the transistors are in such a configuration that the cell holds the value '1' as the information bit. Also, let a SET happen in the gate of transistor $T4$, as illustrated in Figure 2.3 (a), causing it to turn on. Consequently, the upper-left and bottom-left transistors will be, respectively, turned on and off. Also, the up-right transistor will be turned off. The result is an inversion on the value of the stored bit, which became '0', as shown in Figure 2.3 (b). In sum, an SEU happened in the SRAM cell.

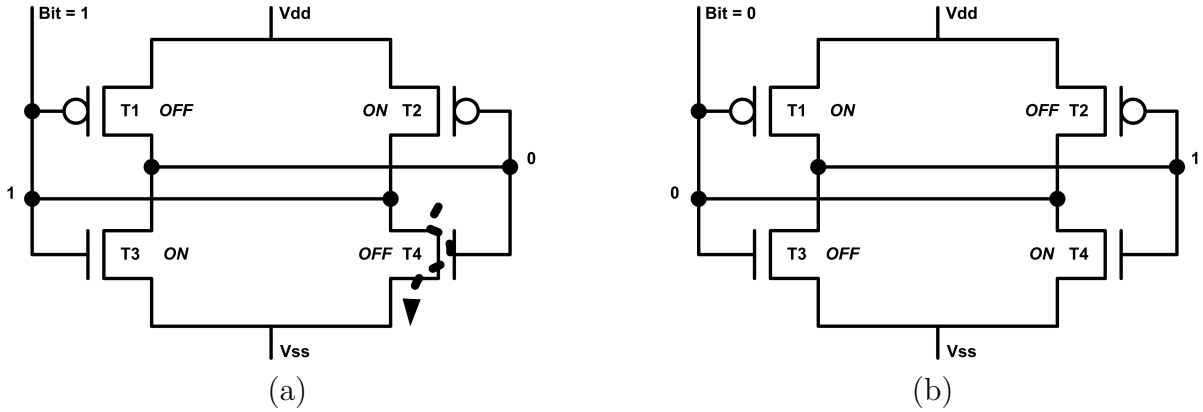


Figure 2.3: SET in an SRAM Memory Cell

Although SEUs in a single bit has been discussed so far, multiple bits can also be affected by a single high-energy particle. Depending on the angle of incidence of the particle, it may travel through several transistors therefore causing the so called Multiple Bit Upset

(MBU). Due to the two-dimension disposition of transistors on a chip die, MBUs are less likely to happen than SEUs. However, due to transistor scaling, a single particle can affect neighboring transistors, thus causing MBUs to occur more often. Moreover, depending on a combinational logic fan-out, a single SET can provoke multiple transient pulses at the outputs. Again, if these SETs meet setup and hold times, they can upset multiple registers therefore causing MBUs.

SEUs can also interfere with controlling portions of a digital circuit, for example, its state machine. As a consequence, the circuit can enter invalid states and even come to a complete halt therefore requiring a power cycle to come back to normal operation. This functional disruption due to SEUs are denoted Single Event Functional Interrupt (SEFI).

2.1.3 SEUs in ASICs and FPGAs

ASICs are customized integrated circuits designed to perform a dedicated function. They can be designed using three main techniques: Standard Cells, Gate Arrays, and Full Custom. Standard Cell employs third party tools and utilizes a library of pre-fabricated building blocks (the so called standard cells) to design the integrated circuit. Once the physical placement of all standard cells is done, the designer must perform the circuit routing therefore electrically connecting all the standard cells. At the end, lithographic layers are produced, which are then used to fabricate the chip die. In contrast, Gate Array design does not employ individual cells. Instead of that, several pre-designed lithographic layers are employed, each of them consisting of transistors, gates and other devices. In this technique, all the appropriate elements must be connected together to obtain the desired circuit functionality. Alternatively, the Full Custom approach requires all elements of the entire lithographic layers to be designed. In other words, all the transistors of the chip are individually conceived and electrically connected.

Regardless of the technique used, chip dies are produced following the specifications of lithographic layers. After that, the chip's functionality is fixed and no further modification is possible without a chip re-spin. Although ASICs' combinational logic can suffer from SEUs, the implemented circuitry cannot be altered by such upsets. Consequently, ASICs provide high immunity against SEUs regarding its circuit functionality. However, radiation effects on the transistors of an ASIC can still cause SEUs in registers and memory elements, as previously described.

FPGAs are devices based on an architecture that can be configured in many different ways, following the designer's specifications. A traditional (simplified) FPGA internal architecture, as illustrated in Figure 2.4, is constituted by Logic Blocks, Input/Output

(I/O) Blocks and Block Random Access Memories (Block RAMs) which are electrically connected by Routing Matrices and Interconnection Buses. More precisely, the term *programming* refers to the process of configuring and interconnecting the FPGA's internal elements. This is achieved by 1) configuring the Logic Blocks and Block RAMs with the appropriate binary values; 2) configuring the Routing Matrices to turn on and off the appropriate connections; and 3) configuring I/O Blocks to function as input, output, or tri-state pins.

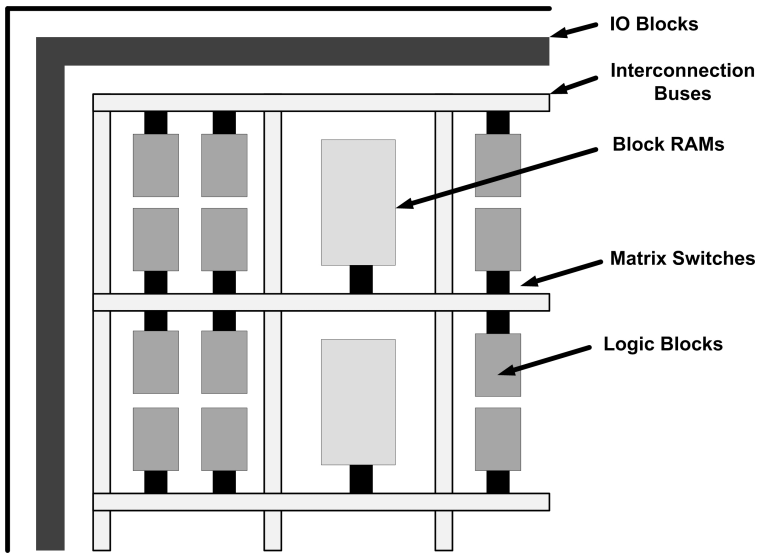


Figure 2.4: Traditional FPGA Architecture

Logic Blocks are configurable elements which implement combinational logic by using Look-Up Tables (LUTs). Specifically, a traditional Logic Block is composed of a LUT, an adder (or carry logic), a multiplexer and a register, as shown in Figure 2.5. I/O Blocks perform the signal interfacing between the FPGA's internal elements and its external circuitry. Block RAMs are on-chip memory based on Static RAM (SRAM) cells. Interconnection buses are sets of wires responsible for electrically connecting the Logic Blocks, Block RAMs and I/O Blocks. Actually, the fine-grained connection between those elements and the Interconnection Bus is performed by a set of multiplexers, namely Matrix Switches. By using this architecture, for instance, it is possible for the FPGA to receive input signals, route it to the Logic Block, perform a computation, and output the results. Obviously this is a trivial example in face of the very complex designs currently supported by FPGAs.

FPGAs are produced by several manufacturers, such as Altera [41], Xilinx [84], Actel [36], QuickLogic [48], Atmel [44], and Lattice [47]. Most of them implement an archi-

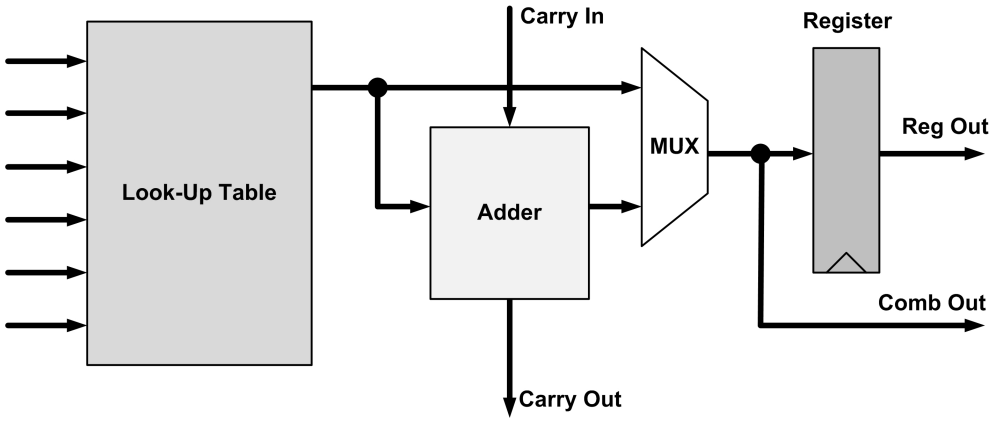


Figure 2.5: Traditional Logic Block Architecture

tecture similar to the one aforementioned.

FPGAs are available in different technologies: SRAM, Flash and Anti-fuse. One-time configurable devices named structured ASICs are also currently available. The type of FPGA is defined by the technology used to implement its configuration memory, which also determines whether the device can be reconfigured. Independently of implementation technology, most devices include Block RAMs traditionally constructed from SRAM cells. Moreover, all devices implement registers within their Logic Blocks.

Advantages and disadvantages exist for each of those devices. Some may provide high resistance against radiation and no reconfigurability. Others may provide high density and reconfigurability, but no protection against radiation. For the whole majority of devices, both SRAMs cells and registers are sensitive to SEUs and must be protected through the use of mitigation techniques. According to the FPGA features, a given type of device becomes more suitable to specific subsystems of the spacecraft.

SRAM-based FPGAs can provide reconfigurability, high density and high performance. These features make these FPGAs attractive for space applications. Current devices support the implementation of very complex systems while keeping satisfactory performance. However, SRAM FPGAs consumes a considerable amount of power when compared to ASICs. Besides, their configuration elements are based on SRAM memory and can be affected by SEUs [71]. If an SEU occurs in a bit of a LUT, it can modify the implemented combinational logic. Also, an SEU in a memory cell of a Matrix Switch can connect (or disconnect) a wire, therefore compromising the correct signal routing. In addition, an SEU in an I/O Block can turn an output pin into an input pin and vice-versa.

In sum, an SEU in a configuration element can easily disrupt the functioning of an entire design implemented in the device. Due to the vulnerability of their configuration elements to SEUs, fault tolerant techniques must be carefully chosen to guarantee the functionality of the design implemented within the device. Also, SRAM-based FPGAs tend to be used in applications where some failures might be tolerated. Good examples are payload systems which are usually complex in terms of functionality, but not crucial to the normal operation of the spacecraft. Applications requiring high reliability such as control of the spacecraft may be implemented using ASICs, flash and anti-fuse FPGAs.

Flash-based FPGAs employs non-volatile memory cells based on flash memory technology to store the device configuration. Since flash memory can be electrically erased and reconfigured, the device can be reprogrammed. Additionally, radiation cannot easily modify the state of flash cells, which makes the device configuration resistant to SEUs. These FPGAs can provide both low power consumption than SRAM-based FPGAs, although still higher than anti-fuse devices. However, the frequency of operation of the latter is usually lower than the former. Flash-based FPGAs do not currently provide such high density as their SRAM counterparts do [62]. However, an advantage is that the configuration elements of flash-based FPGAs offer very high immunity to SEUs [32, 64]. As a result these devices can be used to implement important subsystems, which require both reliably and reconfigurability, as is the case of TT&C and attitude control.

Anti-fuse FPGAs use metal-to-metal connections to define their configuration. Given that these connections are permanent, an anti-fuse FPGA is programmed only once and cannot be reconfigured. Furthermore, metal-to-metal connections cannot be altered by radiation. Thus, the configuration of anti-fuse FPGAs are immune to SEUs and more reliable than reconfigurable FPGAs. Further, anti-fuse FPGAs consume very low power compared to SRAM FPGAs. If no reconfigurability is needed, anti-fuse FPGAs is a very good option to achieve high reliability and low NRE costs. This is normally the case of TT&C and attitude control subsystems of spacecrafts.

Structured ASICs represent another promising technology for space application. Examples include HardCopy [37] and EasyPath [80] from Altera and Xilinx, respectively. Structured ASICs combine both the high flexibility and low NRE costs of FPGA design with ASICs features such as low power and high speed. The methodology consists of first designing and verifying the project using FPGAs. Once all the design requirements are

achieved, the project is migrated to structured ASIC device which becomes the final version of the chip to be used by the target application. After that, its functionality cannot be altered. Although the configuration of structured ASICs is immune to SEUs, registers and memories must employ fault tolerance techniques to cope with bit-flips. Considering the stringent requirements of space applications, structured ASICs have become an attractive alternative to the traditional ASICs adopted in crucial spacecraft subsystems.

ASICs provide both higher performance and lower power consumption. A major drawback of ASICs is their high NRE costs. Furthermore, once the chip is manufactured, its functionality cannot be modified. Any further modification or addition of features imply in a costly chip re-spin, followed by testing and validation. The whole process may take weeks to finish which can potentially impact the mission schedule. Since ASICs do not utilize reconfigured elements such as SRAM-based FPGAs, the former provides a higher immunity against SEUs than the latter in terms of circuit functionality. However, ASICs memories and registers are still vulnerable to bit-blips caused by SEUs. ASICs have traditionally been used for crucial spacecraft subsystems requiring high levels of reliability such as TT&C and attitude control.

In order to increase reliability, space systems computational platform employ hardware devices that utilizes technological techniques such as radiation hardening. Examples of radiation hardened FPGAs commonly found in space are the flash-based Actel RT ProASIC3 [33] and its anti-fuse counterparts RTAX [34] and RTSX [35]. Besides, Xilinx provides SRAM-based radiation-hardened FPGAs such as the Virtex-4QV [83] and the recently released Virtex-5QV [82].

It is important to notice that, although the Jet Propulsion Laboratory (JPL) Electronic Parts Engineering Office [139] has performed radiation testing and qualification of some devices produced by some manufacturers, it does not mean that devices produced by other manufacturers cannot operate in space. For instance, Altera CycloneII [38] FPGAs utilized in this research have not been broadly utilized in space applications. However, according to Altera [42], they meet the requirements of military and aerospace customers. This FPGA relies upon an automated mechanism to perform CRC checks over its own configuration and demand the chip re-configuration if some error is found.

In summary, depending on the device used its functionality may or may not be immune to SEUs. In spite of that, whatever the device used, fault tolerant techniques must always be present to ensure protection of registers and memory elements against SEUs. Thus, if fault tolerant design is to be met in space applications, SEUs must be properly addressed by mitigation techniques.

2.1.4 Mitigation Techniques

Fault tolerance is an attribute of a system capable of performing its functions correctly in the presence of faults within its constituting components [87]. In order to achieve fault tolerance, fault mitigation techniques must be considered. They can be classified into Technological techniques, Architectural techniques, and Recovery techniques.

Technological techniques consist of adopting different fabrication methodologies to achieve higher protection against SEUs. Fabrication processes, such as Epitaxial CMOS, Silicon on Sapphire and Silicon on Insulator [102], can help minimizing SELs and TID effects. Even though, they are expensive processes to be adopted in the fabrication of chips with low production volumes. Besides, those processes are unable to completely eliminate the occurrence of SETs and SEUs. An example of FPGA using Epitaxial CMOS is the Aeroflex's RadTol Eclipse [77]. Xilinx and Actel also provide radiation hardened/tolerant FPGAs such as the Virtex [83, 82], RTAX [34] and ProASIC [33]. Technological techniques are out of the scope of this project given that they are applied during the chip fabrication. This research considers the utilization of architectural and recovery techniques, as presented next.

Architectural Techniques do not impose any changes to the device fabrication process since they rely on design strategies to achieve fault tolerance. They are usually based on some sort of redundancy, such as Hardware redundancy, Time redundancy and Information redundancy, as further detailed in Sections 2.1.4.1, 2.1.4.2 and 2.1.4.3. Software redundancy is another technique used, but it is out of scope of this research. Further, architectural techniques can be applied to different levels of the design. Depending on how the redundancy is implemented, only error detection is possible. However, according to the level of redundancy employed, it is possible to detect and correct one or more errors.

Recovery Techniques have the major goal of either keeping or bringing the system back the normal regime of operation. It can be preventive or corrective. Preventive strategies avoid the accumulation of upsets, whereas corrective ones tends to fix uncorrectable effects of upsets. Further details on recovery techniques are shown in Sections 2.1.4.4 and 2.1.4.5.

2.1.4.1 Hardware Redundancy

Hardware Redundancy consists of the replication of hardware modules along with a voter. Actually, the voter has an important role in this technique, since it is responsible for comparing different results and detect whether an error has happened. According to the number of modules employed, errors can be masked out by the voter so that the right computation is output. Depending upon the implementation particularity, designers may decide to replicate only small structures such as gates and registers, or entire functional units.

A widely used scheme to perform error detection is DMR, which is illustrated in Figure 2.6. In this case, two hardware modules send their computational results to a comparator. If the partial results match, the comparator knows that the computation has been performed correctly. Hence it asserts the DMR output. However, if the partial results diverge, the comparator is not able to tell which computation is erroneous. As a result, it is only able to tell that an error happened, but no result is output.

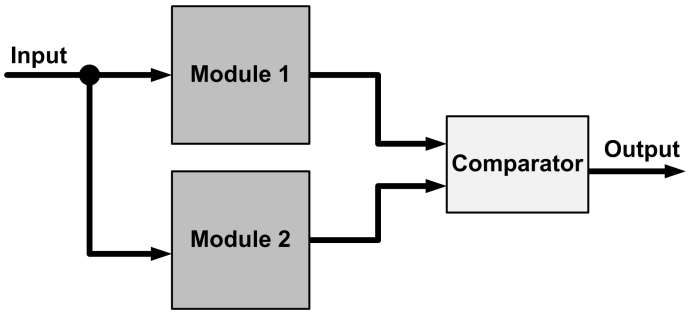


Figure 2.6: Dual Modular Redundancy

Another traditional scheme used for error detection and correction is TMR. As shown in Figure 2.7, this approach employs three hardware modules and a voter. Differently than DMR, the voter must assert the output by evaluating three results. The result is determined by majority voting, i.e. the result repeated twice is elected as correct by the voter. Notice that the voter does not check the validity of the repeating result. However, it is much less likely to have two similar wrong results than two correct ones. As a consequence, it is probable that the repeating result is the correct one. It is also possible to have three different results. In that case, the voter only knows that errors have happened, but cannot determine the output. TMR can be generalized to an N-modular redundancy scheme. If N is odd and $N \geq 3$, it is possible to employ majority voting to determine the output.

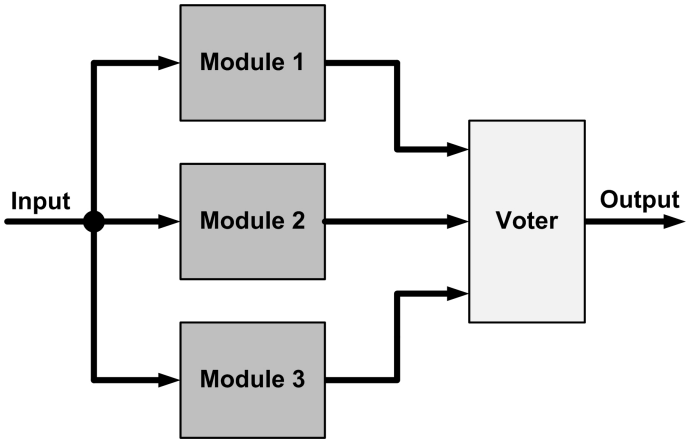


Figure 2.7: Triple Modular Redundancy

2.1.4.2 Time Redundancy

Instead of utilizing extra hardware, it is possible to detect errors using time redundancy. The basic idea is to employ the same hardware to repeat the computation two or more times at different points in time, as illustrated in Figure 2.8. The result of each computation is stored in registers. In the end, the results are analyzed by a voter to resolve discrepancies and determine the output. If majority voting is not possible, the computation can be repeated until a consensus is achieved. Although this approach permits the detection of errors caused by SETs, it cannot be used to detect permanent hardware failures. That can be accomplished, however, by encoding the operands before the computation and decoding them afterwards [87]. Given that this research addresses only soft-errors, recomputation with encoded operands is not further considered.

2.1.4.3 Information Redundancy

Information redundancy utilizes the strategy of adding extra information to the data in order to achieve fault tolerance. That is usually achieved by the use of error detecting and correcting codes. A code consists of a set of rules to represent data into a code word. A code word is basically a set of symbols representing the data. For instance, a binary code has its symbols formed by digits 0 and 1. Furthermore, a code word is said to be valid if it obeys a set of rules defining the code. Otherwise, it is said to be invalid. The encoding process takes the data in its original form and, by following the rules of a code, transforms it into a code word. The decoding process, in turn, takes a code word and transform it

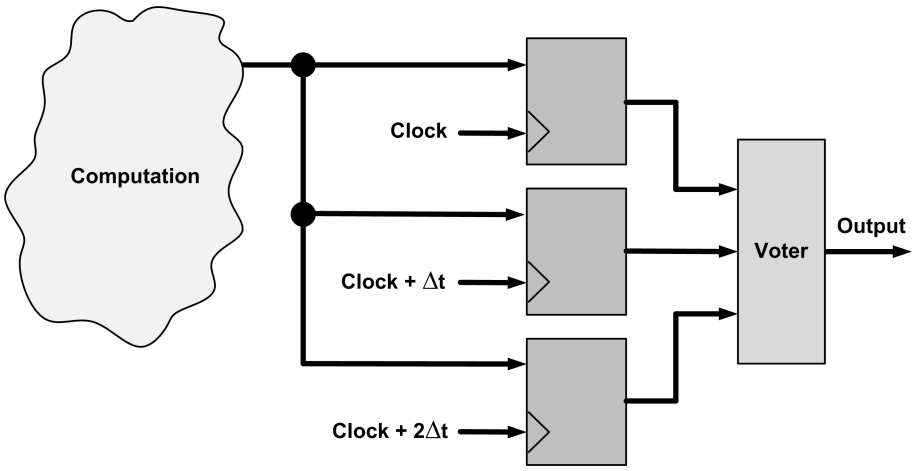


Figure 2.8: Time Redundancy

back to the original data. It is possible to create binary codes which have the valid codes as a subset of the total number of combinations of 0s and 1s. Then, if a valid code word is modified by some means, an error for example, it becomes an invalid code word. Therefore, the decoding process will detect the error given that the code word is not valid. Actually, this is the concept behind error detecting codes. Furthermore, error correcting codes are made possible through the use of extra information into the code word. As a consequence, it is possible to determine the correct code word from the corrupted one.

A term used for both error detection and error correction codes is the Hamming distance. It refers to the number of bits in which two words differ. If the Hamming distance of two code words is one, a single bit-flip transforms the first code word into the second one. If the Hamming distance between the two code words were two, it would have been necessary to have two bit-flips to transform one code word into the other. The minimum distance between two valid code words is defined as the code distance. For instance, a code distance of three would make it possible to correct one bit-flip (distance one from the correct code word) and detect two bit-flips (distance two from two valid code words). Generally speaking, up to c bit-flips can be corrected and up to d bit-flips can be detected if and only if $2c + d + 1 \leq H$, where H is the Hamming distance of the code.

Parity code is one of the simplest forms of an error detecting code. It consists of adding an additional bit of information in the data word, so that the code word contains either an even or an odd number of 1s. In the case of having an even number of 1s, it is called even parity. Otherwise, it is called odd parity. Assuming an even parity is used and suppose that a code word suffer a bit-flip. Then, the final number of ones in the code

word is going to be odd. Hence, a single error can be detected. Notice, however, that parity code will take a double bit-flip as a valid code word. Parity code can be employed in digital circuits in many different ways. One possibility is to encode registers so that single bit-flips can be detected. Also, it is possible to use parity prediction to check the result of computations against single bit-flips. In this scheme, as shown in Figure 2.9, the operands are analyzed by a parity prediction module, which outputs the expected parity of the computation. Then, if the parity of the result does not match the expected parity, an error has happened. Further, if both parities match the computation was either performed correctly or two bit-flips happened in the result.

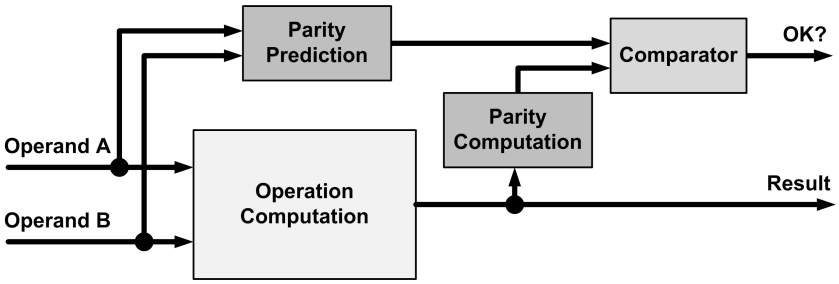


Figure 2.9: Parity Prediction

Hamming code [75] is an example of an error detecting and correcting code based on overlapping parity codes. Hamming code is a very interesting scheme for single bit correction. The encoding and decoding processes are relative simple and inexpensive, respectively, in terms of circuit complexity and computational time. Also, the code does not require much redundancy to provide single error correction. For single error correction over a d-bit data, a Hamming code utilizes c parity bits, which results in a code word of size $d + c$. Precisely, the relation between d and c is $2^c \geq c + d + 1$.

The overlapping parity is organized in such a way that the c parity bits have one combination for each erroneous data bit and for each erroneous parity bit, as well as one combination for the error free case. To illustrate that, consider a 4-bit data word, represented by (d_3, d_2, d_1, d_0). In order to be able to provide single bit correction, it becomes necessary to use 3 parity bits (c_1, c_2, c_3). The parity and data bits are then partitioned in three groups: (d_3, d_1, d_0, c_1), (d_3, d_2, d_0, c_2), and (d_3, d_2, d_1, c_3). Each parity bit is responsible to keep the parity of some of the data bits. The encoding process takes the original data, and according to the bit groupings, computes the respective parity bits (c_1, c_2, c_3). During the decoding process, the parity bits will tell whether the data suffered a bit-flip. For example, if d_0 suffer a bit-flip, both c'_1 and c'_2 computed in the decoding process will not

match with c_1 and c_2 in the code word. Table 2.1 lists all parity bits affected by single bit-flips.

Table 2.1: Parity Bits Affected by Bit-Flips in a Hamming Code

Erroneous bit	Parity bits affected	Syndromes
d_0	c_1, c_2	110
d_1	c_1, c_3	101
d_2	c_2, c_3	011
d_3	c_1, c_2, c_3	111
c_1	c_1	100
c_2	c_2	010
c_3	c_3	001

Since each erroneous bit causes a unique parity combination, this information is called a syndrome and can be used to locate and correct erroneous bit. If the data and parity bits are organized as in Figure 2.10, the syndrome will refer to the position of the erroneous bit within the code word. After that, it is just a matter of flipping the erroneous bit back to its correct value.

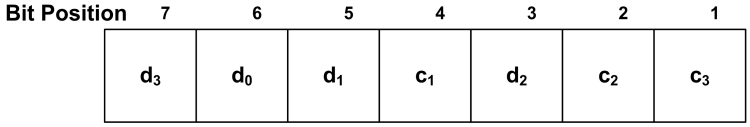


Figure 2.10: Hamming Code-Word Structure

In this research the following terminology for Hamming codes is used: (w,v), where v is the number of data bits (d), and w is the number of data bits along with parity bits ($d + c$).

2.1.4.4 Scrubbing

Scrubbing is an example of preventive strategy to avoid the disruption of the FPGA operation due to accumulation of upsets. The strategy consists in reprogramming the FPGA, from time to time, with a fresh configuration file. On the one hand, this will restore the integrity of the configuration file if some upset has corrupted it. On the other hand, since no integrity check is involved, the FPGA is reconfigured even when it holds an intact configuration file.

2.1.4.5 Configuration Check and Reconfiguration

In order to reduce unnecessary FPGA reconfiguration, some strategies try to first determining whether the FPGA needs to be reconfigured. This can be accomplished by Read-Back operations as supported by Xilinx [21] FPGAs. A read-back operation consists of reading the configuration file in operation within the FPGA, subsequently checking its integrity against a default configuration file. The FPGA is reconfigured only if some discrepancy appears. Moreover, the reconfiguration can be partial or total. If only part of the design is affected by upsets, partial reconfiguration can be performed without disrupting the entire operation of the FPGA. In some other cases, depending on how compromised the configuration file is, it may be necessary to perform a total FPGA reconfiguration.

Instead of utilizing read-back, some FPGAs produced by Altera and Atmel utilizes an approach based on automated integrity checks [13, 39]. In order to keep the integrity of the configuration of the FPGA, cyclic redundancy check (CRC) is commonly used. In this scheme, the CRC value of the current configuration of the FPGA is continuously computed and compared to an expected stored value. If they do not match, a signal is issued ordering the reconfiguration of the FPGA.

2.2 Space Systems

The history of artificial satellites begun in October 4, 1957 when the (former) Soviet Union launched Sputnik I [117]. It was a small satellite with 58cm of diameter, weighting 83.6Kg, whose elliptical orbit was performed in 98 minutes. Sputnik I continuously transmitted frequencies of 20.005 and 40.002MHz, so that it could be confirmed that it was successfully launched and that it was in Earth's orbit. The first communications satellite was the SCORE (Signal Communications by Orbiting Relay Equipment) satellite [70, 69], launched in December 18, 1958 by NASA. SCORE carried a message of the U.S. president Eisenhower stored on two tape recorders, which was intended to be transmitted to the North-American population. In its first pass over California, the primary tape recorder did not respond properly. Then, on December 19, 1958, the second tape recorder finally responded to the control center commands and broadcasted the president's message. SCORE's also transmitted in real time and store-and-forward voice and teletype messages between ground stations in the U.S. during 12 days.

Since late 50's, satellites revolutionized the way communications are performed. Nowadays these spacecrafts are heavily employed in communications infrastructures, as well as for many other applications. Currently, spacecrafts can be classified into five classes [115,

24]: scientific research, meteorological, communications, navigation, and military. Space systems comprise of two segments: space segment and the ground segment. The space segment consists of the spacecraft itself and the means used for launching the spacecraft. The ground segment, consists of the ground stations, communication networks, control centers, and the infrastructure to manage and operate spacecrafts and associated mission data. Usually, a command originates in a control center, travels through communications networks until it reaches a ground station. The antennas pertaining to the ground station beams up the command to the spacecraft. The inverse path is utilized when the spacecraft is sending data to the control center.

The path on which a satellite revolves around Earth (or around any other celestial body) is called orbit. Orbits shapes can be of two types: (nearly) circular and elliptical. Further, according to their altitude, orbits can also be categorized as Low Earth Orbit (LEO), Medium Earth Orbit (MEO), Geosynchronous Orbit (GSO), Geostationary Orbit (GEO), and High Earth Orbit (HEO) [137]. Low Earth orbits may achieve up to 1700 Km of altitude. Medium Earth orbits vary between 1700 and 35700 Km. Geosynchronous and Geostationary orbits refer to orbits of altitudes of 35700 Km. High Earth orbits are considered orbits above 35700 Km. Deep space is another term very often used for distances beyond the gravitational influence of Earth.

Spacecraft links can be classified according to the direction of the communication: uplinks carry data from Earth to space, downlinks carry data from space to Earth, and cross-links carry communication between spacecrafts. Regarding to the type of data of each link, they can be divided in two groups: 1) Data links, and 2) Telemetry, Tracking and Control links (TT&C). Data links carry communications between communications stations and the spacecraft. The TT&C link is used by control stations to send commands and receive telemetry and tracking data to/from the spacecraft.

2.2.1 Spacecraft Subsystems

A spacecraft is consisted of various subsystems. The mechanical structure is considered a subsystem and is designed to provide mechanical support and assist in the thermal control of the constituent parts of the spacecraft. The communications subsystem is responsible for receiving, amplifying and retransmitting radio frequency (RF) signals. The following sections provide further information on additional subsystems such as the Telemetry, Tracking and Control subsystem (Section 2.2.1.1), Electrical Power subsystem (Section 2.2.1.3), Thermal Control subsystem (Section 2.2.1.2), and Attitude Control subsystem (Section 2.2.1.4).

2.2.1.1 Telemetry, Tracking and Command

The telemetry, tracking and command subsystem is responsible for monitoring the state and the health of the spacecraft, as well as for commanding its operations. Telemetry is used to collect information from other subsystems of the spacecraft, such as voltage, current, pressure, temperature, attitude sensors, accelerometers, etc. Tracking is used to collect data related to the spacecraft positioning in space. Based on that, operators can compute future orbital positioning and perform corrections in the spacecraft's trajectory. Both telemetry and tracking may perform on-board processing on the collected data, modulate it in RF, and send it to the ground station. Command include functions such as controlling on-board equipment, setting operational modes, and correcting the spacecraft orbit. It can also be used to reprogram on-board components such as processors, FPGAs, and reconfigurable devices, as well as to communicate and perform computations related to the spacecraft payloads.

In some cases, TT&C along with a Command and Data Handling (CDH) unit is treated as the same subsystem, as in [70]. In other cases, as in [168], they are considered separate subsystems with interconnected functions.

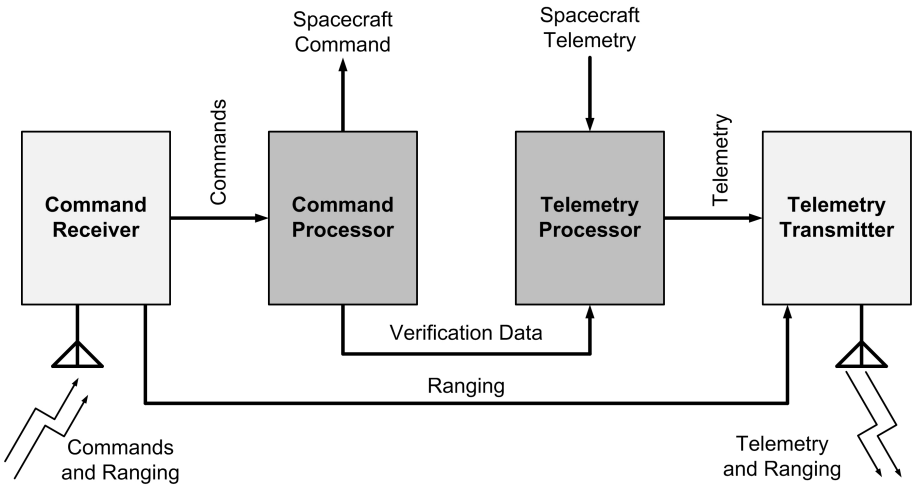


Figure 2.11: TT&C Subsystem

While the telemetry link may tolerate small errors, a bit-flip in the control of a spacecraft may lead to undesirable consequences. As a result, a command has a long way to go until it is in fact executed in the spacecraft. Traditionally, a command is created, modulated in RF, and sent to the spacecraft. The spacecraft's command subsystem receives the signal, demodulates and stores it. Then, the stored command is sent back to the control center

via the telemetry link, where it is verified. Only after a final verification, the control center orders the command execution. After that, the command subsystem distributes the appropriate signals to different parts and other subsystems of the spacecraft. A block diagram of the TT&C subsystem is shown in Figure 2.11.

In order to achieve a secure operation of the spacecraft, it becomes fundamental to implement strong security mechanisms to achieve confidentiality, data integrity and authentication in the TT&C link.

2.2.1.2 Thermal Control

Given the absence of air and other fluids surrounding the spacecraft, it is impossible to rely on convection to dissipate heat during the spacecraft operation in space. Heat transfer must be exclusively carried out by conduction and radiation, making the thermal balance of the spacecraft quite challenging. Spacecrafts receive heat radiated by the Sun, and depending on its orbital position, they may also receive a small portion of heat reflected by the Earth. In addition, internal subsystems dissipate power during their normal operations, which is then carried to the exterior of the spacecraft through conduction by its mechanical structure. The thermal balance is achieved by radiating heat into space. Radiation depends upon the shape and surface properties of the spacecraft.

The spacecraft's temperature is determined by the amount of incident energy, internal dissipation, and thermal radiation into space, as illustrated in Figure 2.12. Given the stringent constraints to achieve the thermal balance, low power dissipation is an important issue to be addressed when implementing security mechanisms for space applications. Actually, since the power dissipation of a circuit is closely tied to power consumption, both issues should be addressed carefully.

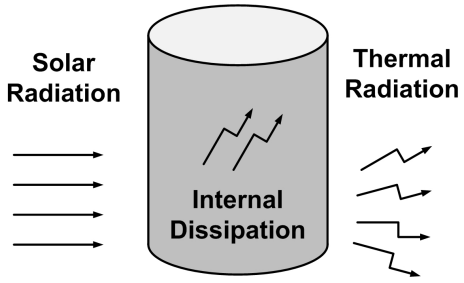


Figure 2.12: Thermal Balance

2.2.1.3 Power

The power subsystem is basically consisted of solar panels, re-chargeable batteries, and control electronics. When the spacecraft is facing some source of light, the silicon photovoltaic cells on the solar panels supply electrical power to the batteries and the rest of the spacecraft. When the spacecraft is in such orbital position where light is blocked by any celestial body (Earth, Moon, etc.), re-chargeable batteries provide the required electrical power.

Actually, things are not that simple. Solar panels, batteries and loads may operate at different voltages. Solar panels generates no power during an eclipse and can produces higher voltages when facing the Sun. Batteries operate at different voltages while charging and discharging. Additionally, battery's charge/discharge cycles must be controlled in order to avoid either an over-charge or a deep discharge. Spacecraft's on-board subsystems may also demand different loads, varying over time. In order to orchestrate this complex system, a control electronics is required. A block diagram of the power subsystem is shown in Figure 2.13.

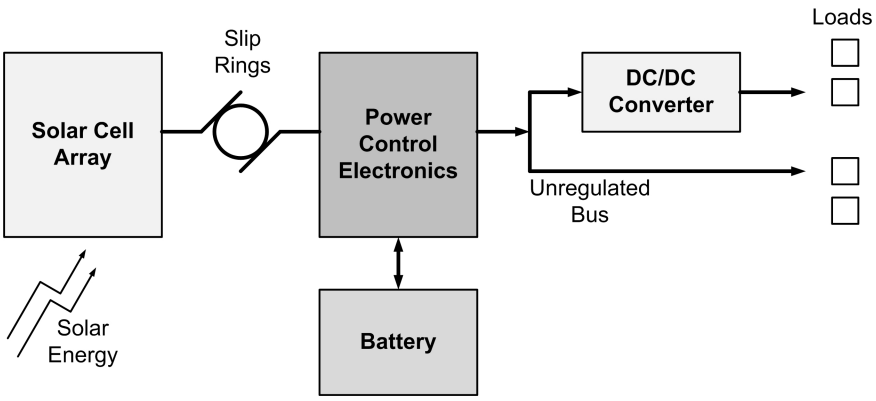


Figure 2.13: Power Subsystem

The amount of power generated by the solar panels depends on the intensity and the angle of incidence of light, as well as on the characteristics of the photovoltaic cells. Due to the rotational movement of the spacecraft while in orbit, the angle of incidence of light varies, and so does the amount of power generated. In order to maximize the power generation, rotational movement of the solar panels are mandatory to keep them oriented to the Sun. This requires a motor to rotate the panels and slip rings to transfer the power from the solar cells to the spacecraft electronic systems.

Moreover, given that the Sun radiates a fixed amount of light, power generation is limited. For example, a GEO satellite may receive in the average 1370 W/m^2 of power radiated from the Sun [70]. On the one hand, the higher the power requirements of a spacecraft, the bigger the solar panels must be. On the other hand, the bigger the solar panels, the heavier they are, thus resulting in additional launch costs.

Furthermore, the more power is consumed, the more heat is generated, and the latter impacts on the spacecraft's thermal balance. Even though processors and FPGAs may demand less power than RF transmitters, for example, the heat created by the former elements has to be carried to the exterior of the spacecraft by conduction. This is mainly done through a thermally conductive medium. Thus, the more heat is generated, the higher is the demand on the mechanical elements utilized for heat transfer. Therefore, it is crucial to design security mechanisms so that their implementation (either in software or in hardware) minimize power consumption, and consequently power dissipation.

2.2.1.4 Attitude Control

In order to spacecrafts accomplish their intended mission goals, they must keep their correct orbit and orientation. For example, communications satellites must keep their planned orbits not only to avoid collisions with other spacecrafts and space debris, but also to keep their antennas always pointed towards Earth. Other examples include ESA's Rosetta [55] and NASA's Cassini [119] spacecrafts, which are intended to perform fly-by of asteroids and planets. That requires maneuvering operations to keep the spacecraft pointed to the target object. Moreover, spacecrafts are subject to disturbances caused by several factors. Common factors are pressures due to solar radiation, misalignment of thrusters, gravitational and magnetic disturbances, etc. Consequently, spacecrafts must be capable of detecting orientation errors and re-positioning themselves.

The correction of orbital and spatial trajectories is performed by the attitude control of the spacecraft. Pointing errors can be detected by instruments such as Earth/Sun sensors, rotation wheels, RF sensors, gyroscopes, and star trackers. The attitude of the spacecraft can be corrected by actuators such as reaction wheels (which spin both ways) and momentum wheels (which spin in only one way). If the reaction/momentum wheels are not capable of performing the attitude control, thrusters must be fired. Although the utilization of thrusters is very efficient, it requires fuel consumption. In order to minimize fuel consumption, additional schemes can also be employed. For instance, magnetic torquing coils and solar flaps produces a weak, but steady torque, which are useful in the attitude control.

The coordinate system used for attitude control is shown in Figure 2.14. It has three axes (x, y, z) and its origin is located at the spacecraft's center of mass. A spacecraft can rotate around three different axes while in orbit. Similarly to airplanes, spacecrafts' attitude control uses the names roll, pitch and yaw, to represent the movement around the axes x, y, z, respectively.

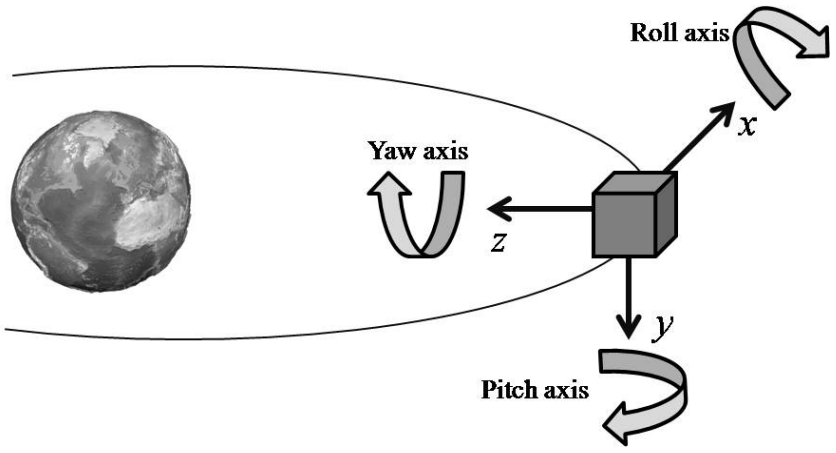


Figure 2.14: Attitude Control: Pitch, Roll and Yaw

By using the orientation detection mechanisms, spacecrafts can measure their deviation from a reference point (Earch, Sun, stars, etc). Some spacecraft carry some on-board capabilities to automatically perform roll, pitch, and yaw movements, through rotation wheels and thrusters. Some others, rely these tasks on their control center. When a control center is involved, the TT&C link is used to read telemetry and send commands to the spacecraft. It is important to emphasize the importance of attitude control, given that they perform crucial and very sensitive movements of the spacecraft.

2.2.2 Space Missions

Space missions can divided into several types, based upon the mission goals. Also, they can be categorized according to their security requirements, which are basically driven by the cost and importance of the mission.

2.2.2.1 Missions Types

Spacecrafts can be classified into five main types: communications, scientific research, meteorological, navigation, and military [24]. Military spacecrafts are out of the scope of this research. Most mission have traditionally been deployed with limited, if any, security [23, 24, 26, 29, 25].

Communication satellites are fundamental building blocks of the international communication infrastructure. They are involved, for example, with the transmission of voice, video, and data over long distances. The absence of this kind of satellite may disrupt an important world wide communication channel, thus causing severe losses.

Scientific spacecrafts are not considered part of a national asset infrastructure. Although scientific missions suffer from the same threats than any other space mission, they involve much less risks compared to communications satellites and manned space flight. However, loss of important data and large investments may incur if this kind of spacecraft is either attacked or compromised.

Meteorological satellites have traditionally been involved with Earth observation missions. These spacecrafts, sometimes called Earth Observation satellites, are employed on climate study and prediction, as well as weather forecast. Outages in this type of service may cause loss of scientific data and also result in life threatening conditions.

Navigation satellites were originally intended to provide positioning and navigation for enterprises such airlines, maritime, trucking and the military. In the last few years, these systems became more and more popular among civilian applications such as automobile navigation systems, emergency location through cell-phones, and hand-held devices for hunting, fishing, exploring and hiking. Examples of navigation satellites are the North-American GPS, the European Gallileo [54], and the Russian GLONASS [7]. Given the current importance and widespread use of these systems, any threat against navigation satellites may be translated as a risk to the navigation infrastructure and potential threat to human life.

2.2.2.2 Mission Security Levels

Space missions can be categorized in three main levels of security [23]: high, moderate and minimal security. Actually, an additional "insecure" category could be considered for spacecrafts without any kind of protection.

High Security missions are generally involved with government and military programs, and are mandatory for manned space flight. Navigation and high cost communication satellites might also fall into this category. In high security missions, the secure access and control of the spacecraft is mandatory under any operational or environmental conditions. More precisely, all data involving the command and telemetry of the spacecraft, as well as the ground data system must employ access control, confidentiality, data integrity and authentication, and also have high availability.

Moderate Security missions are usually considered for communication, meteorological and non-critical navigation missions. In this category, data systems are protected against unauthorized access, e.g. sensitive or critical data related to commercial and operational purposes. In this category, data integrity and authentication of spacecraft commands are mandatory, and confidentiality might be required in some cases. Telemetry data require confidentiality and data integrity. All systems in the ground segment must utilize data integrity and authentication, and possibly confidentiality and access control.

Minimal Security includes the remaining missions, not covered by high and moderate security levels. Minimal security missions require data integrity and authentication for commands, whereas confidentiality might be required in some cases. Confidentiality and data integrity may be used to protect some telemetry data and parts of the ground segment. The ground segment may also include access control for data protection.

2.2.3 Threats Against Space Missions

As any other computer system, there are many threats against the computational platform of space missions [24], which have the potential of compromising the entire mission. In order to achieve higher levels of protection in space data systems, a set of cryptographic primitives has been proposed to be adopted as standard by CCSDS [30]. In addition, it is important to adequately identify threats, so that the appropriate countermeasures can be taken.

2.2.3.1 Common Threats

According to the capabilities of attackers, threats can be classified as passive and active threats. Passive threats refers to threats where the attacker can only monitor and listen to the communication channel to acquire some information. Active threats requires a more sophisticated attacker, who must be able to insert, delete and alter messages in the channel to compromise the communication. These threats are present in both space segment and ground segment.

Data Corruption: Data can get corrupted in several, if not all, constituting elements of space systems. For example, it can happen in the control center, during the transmission to and from the spacecraft, or while on-board of the spacecraft's subsystems. Data corruption may be caused by multiple factors, such as software and hardware failures, unintentional and intentional data modifications, or due to SEUs. The corruption of data can lead to serious consequences to a space mission, including its loss. For instance, a corrupted command has the potential of accidentally firing thrusters which could put the spacecraft spinning out of control. Data corruption can be avoided by using integrity check schemes. Further, due to the harsh space environment the spacecraft immersed, fault tolerance techniques are also of extreme importance to prevent data corruption from occurring.

Data Interception: Data sent by either the ground station and the spacecraft can be intercepted even by a modest, passive attacker. Since communications utilizes specific radio frequencies, it would be possible to use an appropriate antenna and find the right frequency to be able to listen the communication. By listening to communications, an attacker can gain knowledge of protocols and data structures, which can be useful to mount an attack. Moreover, attackers can determine when entities are communicating by performing a traffic analysis of the communication channel. Interception of data can be avoided by using spread spectrum and frequency hopping techniques, and also by applying cryptographic techniques such as data encryption.

Replay: A common product of data interception is the so called replay attack. In this kind of attack, old messages are replayed to mislead a given target, which can be a ground station or a spacecraft. An example of this kind of attack would be an attacker intercepting maneuvering commands, storing them for some time, and at his/her own discretion re-transmitting them to the spacecraft. As a result of this unauthentic maneuvering command, the spacecraft could repeat a thruster burn resulting in an incorrect orbital position.

Replay attacks can be avoided by employing authenticated commanding and using counters and time stamps in messages.

Jamming: An active attacker can interfere with wireless communications by using an RF signal of frequency similar to the original one. This is a serious threat to communication link between the spacecraft and ground station. Moreover, the advent of jamming during critical and emergency commanding may have disastrous consequences. Since the control center cannot perform any better than wait for the interference to terminate, it might be too late to recover the commanding action, leading to the loss of the spacecraft. Jamming can be reduced by using frequency hopping and spread spectrum techniques. Also, multiple uplinks/downlinks and multiple access points allow for additional communication channels in case one of them is jammed.

Unauthorized Access: Due to the complexity of current space missions, several tasks are divided among different teams, not necessarily at the same physical location. Each team has its own privileges and is able to operate upon its designated part of the system. Unauthorized access happens when authentication is weak enough to allow someone to perform actions that he/she was not supposed to. For example, a group of control operators may be responsible for analyzing telemetry data, and based on that, perform commanding actions. A sub-group of this team may be eligible to exclusively access telemetry data, but not to send commands to the spacecraft. In this case, an unauthorized person, who is only able to read telemetry data, can may put the mission at risk by sending unauthorized commands. Unauthorized access can be addressed by implementing access control in control centers. Another important countermeasure against unauthorized access is the implementation of encryption and authentication over the TT&C link.

Masquerade: An attacker can lie about his/her identity, and impersonate a legitimate user to the system. In this case, the system would not realize that the communication is taking place with an illegitimate entity. As a consequence, an attacker would be assumed as a valid user, and therefore provided all the capabilities of the impersonated entity. An attacker may also try to masquerade as a legitimate entity by performing replay attacks. Masquerade can be avoided by using strong access control in the spacecraft control center. Also, data origin authentication in the TT&C link should be enforced.

Ground Facility Physical Attack: A brutal way of compromising space systems is the physical attack against a ground facility. A terrorist attack has the potentiality of disabling

the entire ground station or control center headquarters for a long period of time. This may also happen due to natural disasters. If no backup control station exist to take place of the original one, permanent damage to the mission goals is almost certain. Other attacks may target antennas to disrupt communications, or even taking control over the facility to control the spacecraft. Reinforced access control, guards and gates can help with the security of ground facilities. However, it is also important to count with backup sites in case the main facility is disabled.

Software and Hardware Threats: Space systems are composed of multiple subsystems, which in turn are constituted of software and hardware parts. For instance, ground segment networks may be vulnerable to system invasion and infection by viruses and worms. Another example is the TT&C subsystem of spacecrafts, which may be composed by several special purpose hardware and general purpose processors. As any kind of computer system, spacecraft software and hardware might contain bugs and security holes. An attacker can exploit those issues to break into the system and take complete control over the spacecraft. Intensive testing, simulations and evaluations is a way of reducing the number of bugs and security holes in on-board software and hardware.

2.2.3.2 Reported Attacks Against Satellites

Companies involved with the multi-billion dollar space business may suffer financial losses if the reliability of their systems is questioned. Although one cannot be certain about the existence of "cover ups", a number of attacks against satellites have been reported by the media. This clearly reflects the consequences of deploying spacecrafts without appropriate security mechanisms. It would not be surprising, though, if we had an astonishing number of satellites being compromised every month. Fortunately, this is currently not the case. However, given to advances in communications technology and its associated cheap access, communications security is becoming a more common requirement in the concept of new space missions.

One of the first known cases dates back to November, 1985, when a Disney Channel was superimposed for 90 seconds by an adult TV broadcast. Disney kept it secret until the "Captain Midnight" incident on April 27, 1986 [103]. In fact, until January 15, 1986, dish owners used to receive the HBO channel for free. After that date, HBO started encrypting its transmission, so that users had to buy a $385 decoder and pay a fee of $12.95 per month to be able to continue watching the channel. It was then on April that Captain Midnight interrupted the transmission of TV signal to 1.7 million satellite dish owners

in the eastern USA. The broadcasted the message "$12.95/month? No way!", and also threatened interrupting other paid programs starting on May. Captain Midnight had not even bothered breaking the encrypted transmission; He sent his protest by simply beaming up a signal to the HBO satellite strong enough to override the original channel's signal. Given the strength of that signal, it was estimated that it did not come from a backyard reception dish. The attacker would have to have utilized a 30 feet satellite dish. Thus, investigators believed that Captain Midnight was an employee/engineer at a commercial satellite uplink site. After this incident, HBO increased the strength of its signal to avoid other pirates doing the same. Even though, other cases like Captain Midnight are expected to happen again. At that time, HBO used to employ geosynchronous satellites like Hughes Galaxy 1 and RCA Satcom 3R to transmit their signals. The problem was that anyone that knew the correct communication frequencies and how to send commands to those spacecrafts, could put them spinning out into space. No need to mention that this would be feasible by simply listening to previous communications with the satellite. By the way, threats like that were received by HBO.

Another serious incident happened in March, 1999, involving one of the UK Ministry of Defense's (MoD) Skynet satellites. This spacecraft was used by defense planners and military forces around the world and also to coordinate bombing raids in Iraq. Initial reports [56, 72] claimed that an attacker had seized control over the satellite, altering its course. The attackers would have also blackmailed the MoD demanding £3 million in order not to definitively compromise the spacecraft. Other source [126] reported that the group of attackers neither moved the satellite nor blackmailed MoD. However, it was claimed that hackers intercepted the link between the ground station and the satellite and reprogrammed the Skynet's control subsystem. Special precaution was taken by the attackers not to lose control of the spacecraft. The strategy used by the attackers was to follow a sort of recipe, which had been published years earlier by other hackers. Again, it was just a matter of learning control codes and beaming up a custom signal to take control over the spacecraft.

From June 23 to 30, 2002, the Chinese spiritual practice Falun Gong attacked the satellites Sinosat 2A and 3A. In this incident they interfered with the regular transmission of nine channels of the China Central Television Station (CCTV) and ten provincial TV channels for rural and remote areas [138]. In one of the occasions, they substituted the transmission of the final match of the World Cup with Falun Gong's propaganda and material containing reference to the cult. Besides TV signals, the Sinosat satellites also carried signals for weather forecast and telecommunications. If one of those signals were sabotaged, it would have not only bored soccer fans, but also endangered lives and potentially caused a bigger economic impact. In this case, several people were convicted and

punished with long jail sentences. Again, on November 20, 2004, Falun Gong transmitted cult's promotion contents using an AsiaSat 3S satellite from Asia Satellite Telecommunications Co. Ltd (AsiaSat) [124, 49]. A following assault happened on March 14, 2005, when the same AsiaSat 3S satellite was used to transmit Falun Gong's propaganda [142]. This caused the interruption of the regular transmission of provincial TV channels in China. The last two attacks were denied by the Falun Gong cult [85].

Another recent incident involving minority groups refers to the Liberation Tigers of Tamil Eelam (LTTE). The Tamil Tigers is a separatist group in Sri Lanka classified as a terrorist group by 32 countries [141]. In April, 2007, they hacked an Intelsat's satellite orbiting over the Indian Ocean to transmit propaganda [153, 113] referring to the group.

In face of all these attacks, it is quite clear that space systems are at the risk of attacks. Furthermore, as technology evolves, more attacks should be expected. Thus, in order to assure the security of those spacecrafts, strong security mechanisms should be adopted.

2.2.4 Security Requirements

In order to properly address the threats against space systems, security mechanisms should be employed in both ground and space segments. Although CCSDS has been surveying practices for implementing security in space systems [23, 28, 27], there exist only drafts of the final recommended practices for encryption [29], data integrity and authentication [26] and key management [25]. As of April of 2011, no final recommendation has been done yet.

The recommended practices should be employed on communication links and data being processed or stored within a space system. The implementation of the recommended practices are directed by the mission security level, and can be applied on a point-to-point, hop-by-hop, or end-to-end basis. In the point-to-point case, security mechanisms are provided between two communicating entities. For example, a control center and the TT&C subsystem. In the hop-by-hop case, security mechanisms are applied between each hop of the communication path, until the information reaches its final destination. Finally, in the end-to-end case security mechanisms are used between the source and the destination, without the intervention of intermediate entities. For instance, between the spacecraft instrument and the control operator/scientist.

2.2.4.1 Confidentiality

Confidentiality is a service used to address the problem of making information accessible only to an authorized entity. It is usually achieved by using encryption. Encryption can be classified as symmetric or asymmetric. In symmetric systems, the communicating entities uses mathematically related keys (usually the same) to perform encryption and decryption. Asymmetric systems employs a pair of mathematically related keys, named public and private keys. In this case, the public key is used to encrypt a message, whereas the private key is used to decrypt it.

A preliminary CCSDS survey on encryption algorithms [29] basically follow the recommendations of the U.S. National Institute of Standards and Technology (NIST). The Advanced Encryption Standard (AES), as defined in [128], is preliminarily recommended to be used as a standard encryption algorithm for space data systems. The minimum recommendation for the key size is 128 bits, but larger keys such as 192 and 256 bits may also be used for higher levels of security. So far, no recommendations have been done for public-key cryptosystems.

2.2.4.2 Data Integrity and Authentication

Data integrity is a service that allows for the recipient of a message to certify that the received data was not modified, either unintentionally or intentionally, while in transit. Data origin authentication, in turn, allows for the receiver to certify that the received data came from the claimed sender.

Preliminary algorithm survey done by CCSDS [26], indicates that Message Authentication Codes (MACs) and digital signatures could be appropriate for space applications. More specifically, digital signatures should follow the specifications of the Digital Signature Standard (DSS) provided in [134]. HMAC should be adopted as the keyed-hash MAC algorithm as specified in [129]. The hash function used for both DSS and HMAC should be the Secure Hash Standard (SHS) as specified in [133]. More precisely, at least SHA-1 should be utilized, but the use of SHA-2 is highly recommended. Encryption-based MACs, such as DES-CBC-MAC [127], CMAC [131], and CCM [130] are also considered in [26]. It is important to notice that no final recommendation has been done so far for using these algorithms in space.

2.2.4.3 Key Management

A frequent problem with cryptographic systems is the key management. Keys must be established and maintained for all the algorithms being used within a system. Space systems have multiple factors which should be taken into account when deciding on a key management scheme. These factors also vary according to the type of orbit and the type of mission.

The huge distances in space may imply long transmissions delays, which may take tens of minutes. In order to cope with that, the key management handshaking should be kept as simple as possible. Further, communications may be windowed due to the spacecraft orbit and failures, therefore interrupting the execution of protocols. This requires the key management to be performed very quickly and be able to recover from broken communications. Due to the limited bandwidth, the key management should also use the least bandwidth possible. Since spacecraft's hardware does not have as much computational power as their terrestrial counterparts, algorithms must target efficiency. Finally, as spacecrafts often operates for longer periods than originally planned, the key management must support extensions of missions lifetimes.

As a consequence, complex key management schemes involving either several message exchanges or third parties should be avoided. A preliminary recommended practice for the ground segment is based on the Internet Key Exchange (IKE) [25]. For the space segment, which is mainly addressed in this research, the preliminary recommendation is to employ a Secret Key Infrastructure (SKI) [25].

The SKI would be based on master keys installed on-board of the spacecraft prior to launch, whose function is to be employed as Key Encryption Keys (KEKs). More precisely, the KEKs would be used exclusively in the secured communication with the control center while uploading session keys to the spacecraft. The number of KEKs would be determined based on the mission type and lifetime, on the frequency that session keys are uploaded, and also on the lifetime of the KEK itself. The session keys are categorized in different classes according to the security mechanism and type of link that they are involved. They could be divided as Command Authentication Keys, Command Encryption Keys, Telemetry Authentication Keys, and Telemetry Encryption Keys. Each of those can be further sub divided into classes for each of the components and payloads of the TT&C subsystem.

2.2.4.4 Emergency Commanding

Spacecrafts are subject to faults and problems while in space. There are multiple factors that can bring a spacecraft out of control. Faulty sensors and actuators may pro-

vide an erroneous positioning measurement or an under/over actuation on the spacecraft movements. Power outages and temperature may disturb the normal functioning of electronics. Particles coming from space may cause upsets disturbing the correct functioning of the spacecraft. Further, operators on the ground may unintentionally send erroneous commands to the spacecraft. In addition, attackers may send their own commands to intentionally compromise the spacecraft operations. Independently of the case considered, countermeasures must be taken to re-gain the control of the spacecraft.

A common practice usually employed by space agencies is to bring the spacecraft to a "safe mode" when a failure is detected. Actually, if some major failure happens, the spacecraft autonomously put itself in "safe mode" and wait for further commanding from the control station. In other cases, the control station can do this by sending commands to the spacecraft. After that, a reset is performed and/or a backup system is activated. In this case, it is interesting to keep the commands as short and simple as possible, so that minimal transmission time and processing power is consumed while transmitting and decoding the command.

In fact, "safe mode" has been used very often as a recovery technique. Some examples of well known spacecrafts which recently utilized "safe mode" to recover from failures are: Deep Impact [159], Hubble Space Telescope [160], Spirit Mars Rover [154], Mars Odyssey [156, 4, 163, 162, 158, 148], Mars Reconnaissance Orbiter [116, 118, 164], Cassini [155, 161, 98], Kepler Space Telescope [165], and the Dawn Spacecraft [157].

In the event of a tumbling spacecraft, the rotating solar panels may not receive the enough power from the Sun. As a result, almost all the power to keep the spacecraft's operations is drained from the batteries, which start to discharge. Further, tumbling movement can cause the spacecraft's antennas not be correctly pointed to Earth. Actually, the antennas may be oriented correctly for just a very short period of time. In this case, the commands must be as short as possible, and be sent over and over again, so that the spacecraft can hopefully receive them in a narrow communication window.

Emergency commanding typically bypasses the on-board computer and goes directly to the hardware command decoder. However, this may represent a serious security hole if the commands are not authenticated. An attacker, for example, could simply replay previous emergency commands and restart the spacecraft later on. If such an unexpected reset is issued, on-board data may be lost. Even worse, if attitude maneuvering is performed, the spacecraft's orbit may get seriously compromised. That can affect the spacecraft orbital path, potentially dropping it down to the atmosphere or sending it into deep space. On the other hand, authenticated commanding would imply bigger commands, thus increasing communication time. As a result, it is interesting to have authenticated commanding, but

at the same time, to keep them as short as possible so they can be received in the narrow communication imposed by a spinning spacecraft.

2.2.4.5 Other Requirements

Besides all the security issues discussed above, space missions have to address other requirements when implementing security mechanisms. The first one is the use of open standards for algorithms, protocols and implementations. The architecture should also be expandable to allow the use of new technologies and the inclusion alternative approaches. Remote re-programmability/re-configuration is also highly desirable, so that the security architecture can be updated to met new security standards, and also include patches and fixes to bugs and security holes. In addition, the system should be made flexible enough to be integrated with other systems in the future. Moreover, all the security mechanisms should be implemented in a fault tolerant manner so that faults neither compromise data nor disrupt the normal operation of the spacecraft.

2.2.5 Platforms for Space

Many spacecraft subsystems have traditionally been built using ASICs [63]. However, due to the low production volumes of spacecrafts, this kind of approach tends to result in high non-recurring engineering (NRE) costs and long design cycles. This impacts the project budget and has the potential of delaying the mission schedule. With the advent of FPGAs, it became difficult to disregard the utilization of these devices in space applications [74]. Actually, FPGA's debut in space happened in 1996, on-board of the SAMPEX spacecraft [114]. Compared to ASICs, FPGAs favor relatively short design cycles and almost zero NRE costs. Their reconfigurability provide higher levels of flexibility. These devices make possible, for example, post-launch incorporation of features, updates and patches to the spacecraft hardware. Reconfigurability also benefits cryptographic subsystems, which can be updated to meet new security standards and to fix security holes. Moreover, the reconfigurability of FPGAs can be further utilized as a SEU mitigation strategy.

2.2.5.1 FPGAs and Fault Tolerance

Several SEU mitigation techniques for SRAM FPGAs have been proposed in [22]. The first one takes advantage of the high speed configuration capabilities of Virtex FPGAs. In this

approach, the configuration of the FPGA is read and compared with an expected configuration file. Then, if the current configuration of the FPGA is corrupted, a reconfiguration cycle takes place. This scheme is often referred to as "read-back and reconfiguration". For instance, a Virtex V1000 FPGA whose configuration file size is 6.5 Mbits programmed in approximately 20ms. Although that technique is very effective to correct SEUs happening in the configuration elements of the FPGA, it would be desirable to decrease the amount of time spent on the device re-configuration. Therefore, the second technique proposed refers to partial reconfiguration. Partial configuration allows specific frames to be written to the FPGA configuration memory. Furthermore, the entire device is kept active during the partial configuration procedure, but the individual frame being rewritten.

Another technique recommended as a mitigation technique proposed in [21] is scrubbing. The advantage of this method is that it can be performed much faster than read-back and reconfigure, since no comparison of bit-streams is performed. However, this also leads to a disadvantage, which is reconfiguration of the FPGA even when its configuration has not been corrupted. Hence, in order to make this approach efficient, the scrub cycle must be chosen in accordance with an expected upset rate.

In addition to read-back and (partial) reconfiguration, a new method was introduced in [22] to implement majority voting. Instead of using regular LUTs as shown in Figure 2.15 (a), the proposed scheme implements majority voters using tri-state buffers, as illustrated in Figure 2.15 (b).

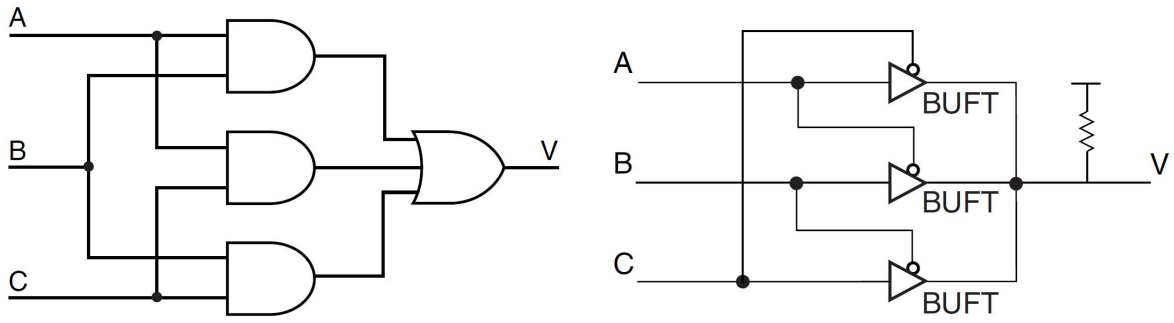


Figure 2.15: Majority Voting Implemented with (a) LUTs and (b) Tri-State Buffers from [22]

The Virtex tri-state buffer primitive, named BUFT, functions as an active low tri-state buffer. Considering Figure 2.15 (b), and inputs A, B, and C all low, it is easy to observe that the output of the voter is going to be low. In contrast, if A, B and C are all high, the

tri-state are all disabled, allowing the resistor to pull the output to high. Further, when two inputs are high, two BUFTs will be disabled but a third one will pull the output to high. Now, if two inputs are low, one BUFT will be disabled, another one will output high, and a third one will output low, therefore causing a contention. However, the Virtex implement its bussing logic such that the described contention does not occur, and the voting scheme works correctly. Although the interconnection of the three BUFTs rely on configuration SRAM cells, an upset in a configuration cell can disconnect one input or output of one of the BUFTs. However, this does not disrupt the functionality of the BUFT-based voter. In a LUT-based voter, an SEU in a LUT can compromise its truth table therefore causing the voter to fail.

Yet another scheme proposed in [22] is TMR. This approach is taken as "rock-solid" by the authors and utilizes four devices. Three devices are used for computation and a forth one employed for mitigation. Actually, SRAM FPGAs could be used as the computation devices, and a small radiation hardened FPGA or ASIC would perform the majority voting. The main drawback of this approach is the implementation area and power consumption demanded by the utilization of three FPGAs along with a mitigation device.

2.2.5.2 Processor-Based Systems and Fault Tolerance

A processor frequently used in space applications is Leon3 [65]. The Leon3 processor is a 32-bit synthetizable processor core based on the SPARC V8 architecture [166], which can be implemented in both ASICs and FPGAs. The Leon3 processor was developed by Aeroflex Gaisler [66] under contract with ESA and is currently distributed under the GNU General Public License (GPL) [78].

A fault tolerant version of this processor, called Leon3-FT [67], is also available. The Leon3-FT architecture is illustrated in Figure 2.16. Leon3-FT is normally implemented on Actel RTAX anti-fuse FGPAs to achieve higher resistance against SEUs. Its memory interface is capable of performing single-error correction and double-error detection in 32-bit data words. Also, its instruction and data cache memories are protected by 4 parity bits per 32-bit words, allowing for the detection of up to 4 bit errors per cache word. Upon error detection, the cache line is flushed and the instruction/data reloaded. Furthermore, the processor register file is protected against errors by employing 4 parity bits per 32-bit of data, plus a duplicated copy of the data word. If the register parity is inconsistent with the register data, the erroneous register is overwritten with the data read from the redundant location. In the case of having the redundant data also corrupted, a trap is generated to indicate error in the register file.

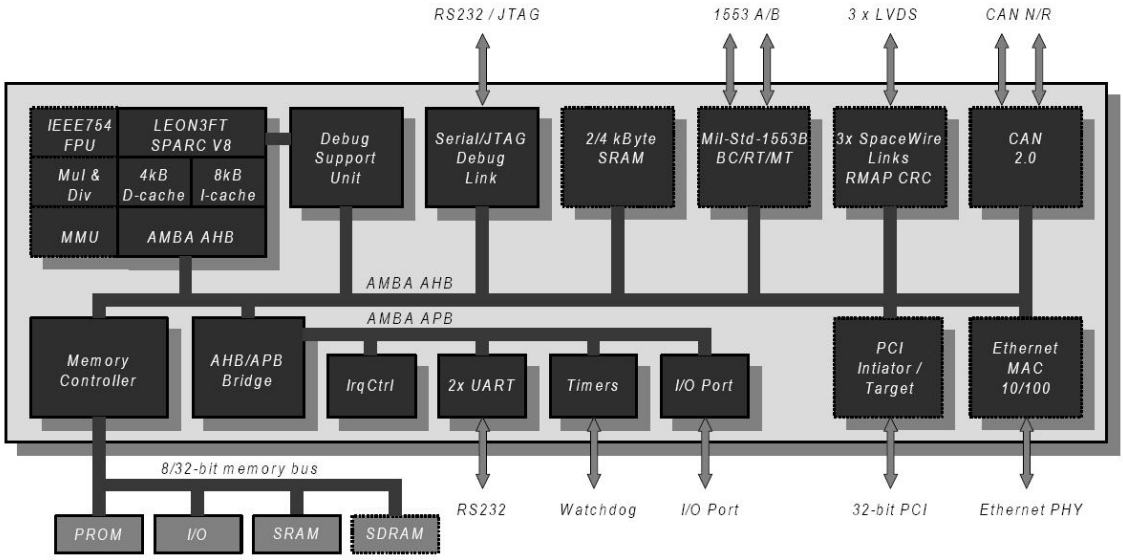


Figure 2.16: Leon3-FT Architecture from [67]

The underlying hardware platform provided by the Actel RTAX FPGA provides extra protection against SEUs through two extra features. The first one is the FPGA configuration, which is protected against SEUs due to anti-fuse technology. Second, the FPGA registers cells, so called R-cells, employs three latches and a voting scheme which protects their registers in a scheme similar to TMR.

An adaptive platform is presented in [99], which dynamically reconfigure itself to match the environment currently faced by the computational platform. Such a system has been prototyped using an Xilinx Virtex 5 FPGA and is based on PicoBlaze soft processors [81]. Depending on the radiation level as well as on the requirements of user applications, the platform can switch among three different modes, namely fault tolerant, parallel processing, and low power. Figures 2.17 (a) and (b) respectively illustrate the architecture of the fault tolerant and parallel processing modes. The architecture of the low power mode is similar to the parallel processing, except that it employs only one processor, only one output logic, and no IRQ module.

A configuration ROM stores three different configuration files corresponding to each of the aforementioned modes of operation. The main drawback of this approach is the entire reconfiguration of the FPGA in order to switch from one mode of operation to another.

An approach to optimize the reconfiguration of the FPGA is proposed in [100]. It consists of organizing and partitioning the configuration elements of the FPGA into tiles,

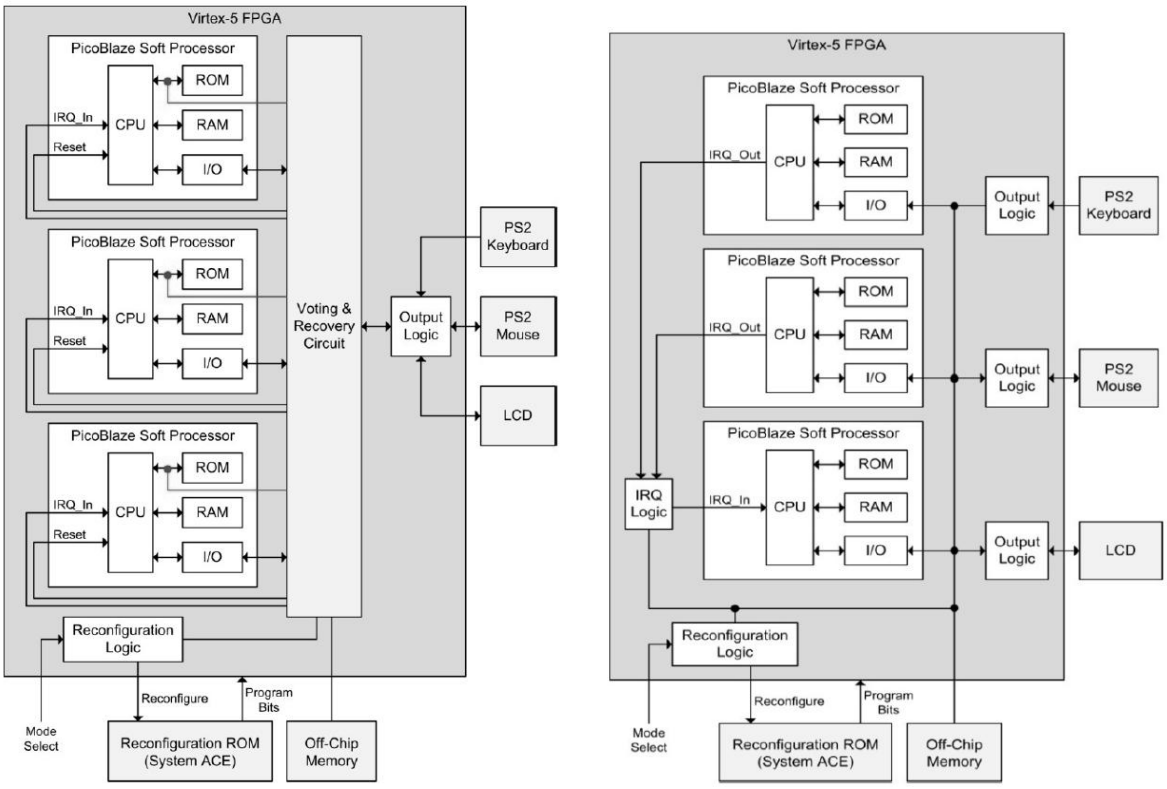


Figure 2.17: PicoBlaze Platform in (a) Fault Tolerant and (b) Parallel Processing Modes from [99]

so that partial reconfiguration can be performed. It also considers the utilization of multiple soft core PicoBlaze processors distributed among the multiple tiles of the FPGA. However, at any time only three processors work in parallel to form a TMR scheme. In the event of failure, a recovery procedure takes place which first attempts to reset, reinitialize and re-synchronize the faulty processor. If this procedure is unsuccessful the tile of the FPGA containing that processor is deactivated and is replaced by a fresh processor residing in another FPGA tile. This procedure is followed by a partial reconfiguration which attempts to recover the tile containing the faulty processor. If such a partial reconfiguration is capable of restoring the integrity of the processor, it is reserved as a spare element and can be re-utilized in the future. In the case of an unsuccessful partial reconfiguration, it may be the case that the FPGA configuration elements may have suffered a permanent fault. Thus, that tile is marked as damaged and is no longer used.

2.3 Cryptographic Algorithms

Even though the implementation of security in space systems may require a large range of cryptographic primitives, this work focuses on integrity checking and message authentication codes. Specifically, the algorithms employed are the SHA-2 family of hash functions as well as the HMAC based on SHA-2. Hardware implementations of these algorithms along with efficient mechanisms for fault tolerance are presented in Chapters 5, 6 and 7. The next two subsections provides a short description of the SHA-2 and HMAC algorithms.

2.3.1 SHA-2 Algorithm Description

The SHA-2 family of hash algorithms [133] comprises four algorithms, namely, SHA-224, SHA-256, SHA-384, and SHA-512. Detailed information on the implementation of the entire SHA-2 family of hash functions, including fault tolerant schemes, can be found in [93]. As can be observed in [93], the implementation of SHA-224 and SHA-256 leads to similar experimental results due to several commonalities between the two algorithms. Similar behavior is observed for SHA-384 and SHA-512. Thus, the remaining of the discussion exclusively focuses on SHA-256 and SHA-512.

The SHA-256 and SHA-512 algorithms are one-way hash functions that able to process messages of up to 2^{64} and 2^{128} bits, respectively. These algorithms can be divided into two computational parts, namely preprocessing and hash computation. The output of the algorithms is an L-bit message digest. Moreover, the datapath bit-width of these functions is denoted by D. Table 2.2 lists some numeric values for some SHA-2 parameters.

Table 2.2: SHA-2 Algorithm Parameters

Parameter	SHA-256 (bits)	SHA-512 (bits)
j	64	80
D	32	64
B	512	1024
L	256	512

A complete set of SHA-2 parameters utilized throughout this document is presented below.

B : SHA-2 input block size (in bits);
L : SHA-2 message digest size (in bits);
D : Datapath width (in bits);
N : Number of message blocks;
i : Message block index, where $1 \leq i \leq N$;
j : Number of algorithm iterations;
t : Iteration index, where $0 \leq t \leq j-1$;
$M^{(1)}, ..., M^{(N)}$: Message blocks;
$H_0^{(0)}, ..., H_7^{(0)}$: Initial hash values;
$H_0^{(i)}, ..., H_7^{(i)}$: Intermediate hash values;
$W_0, ..., W_j$: Message schedule words;
$a, ..., h$: Working variables;
$K_0, ..., K_j$: Constants.

2.3.1.1 Pre-Processing

The preprocessing stage first splits the original message in N blocks, where each block is B bits wide. These data blocks are denoted as $M^{(1)}, M^{(2)}, ..., M^{(N)}$. Then, padding is performed if the message length is not a multiple of the underlying block size. Next, eight initial hash values, $H_0^{(0)}, ..., H_7^{(0)}$, are set. Each algorithm uses a distinct set of initial hash values reported in [133].

2.3.1.2 Hash Computation

The entire computation of the message digest is based on operations over D-bit words. The SHA-2 algorithms comprise j message schedule words $(W_0, ..., W_{j-1})$ and eight working variables (a, b, c, d, e, f, g, h). Besides, eight hash values $(H_0^{(i)}, ..., H_7^{(i)})$ are utilized, where H^i is the i-th intermediate hash value and H^N the final one. In addition, SHA-2 uses D-bit constants $K_0, ..., K_{j-1}$ as specified in [133]. Furthermore, six logical functions are employed, as shown below. The operations $ROTR^n(x)$ and $SHR^n(x)$ are rotation and shift of x by n bits to the right.

SHA-256:

$$\begin{aligned} Ch(x,y,z) &= (x \wedge y) \oplus (\bar{x} \wedge y), \\ Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \\ \sum_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x), \\ \sum_1(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x), \\ \sigma_0(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x), \\ \sigma_1(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x). \end{aligned}$$

SHA-512:

$$\begin{aligned} Ch(x,y,z) &= (x \wedge y) \oplus (\bar{x} \wedge y), \\ Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \\ \sum_0(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x), \\ \sum_1(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x), \\ \sigma_0(x) &= ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x), \\ \sigma_1(x) &= ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x). \end{aligned}$$

Further, for each message block i, $1 \leq i \leq N$, a four-step digest round is performed as follows:

Step 1: Initialize the eight working variables

$$\begin{aligned} a = H_0^{(i-1)}, \quad b = H_1^{(i-1)}, \quad c = H_2^{(i-1)}, \quad d = H_3^{(i-1)}, \\ e = H_4^{(i-1)}, \quad f = H_5^{(i-1)}, \quad g = H_6^{(i-1)}, \quad h = H_7^{(i-1)}. \end{aligned}$$

Step 2: Prepare the message schedule

$$W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}, & 16 \leq t \leq j-1. \end{cases}$$

The number of words processed by the message scheduler is given by j. Actually, j corresponds to the number of iterations performed by the algorithm.

Step 3: For $t = 0$ to $j - 1$ do:

$$
\begin{aligned}
&T_1 = h + \textstyle\sum_1(e) + Ch(e, f, g) + K_t + W_t,\\
&T_2 = \textstyle\sum_0(a) + Maj(a, b, c),\\
&h = g,\\
&g = f,\\
&f = e,\\
&e = d + T_1,\\
&d = c,\\
&c = b,\\
&b = a,\\
&a = T_1 + T_2,
\end{aligned}
$$

Step 4: Compute the i^{th} intermediate hash value $H^{(i)}$:

$$
\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)}, \quad H_1^{(i)} = b + H_1^{(i-1)},\\
H_2^{(i)} &= c + H_2^{(i-1)}, \quad H_3^{(i)} = d + H_3^{(i-1)},\\
H_4^{(i)} &= e + H_4^{(i-1)}, \quad H_5^{(i)} = f + H_5^{(i-1)},\\
H_6^{(i)} &= g + H_6^{(i-1)}, \quad H_7^{(i)} = h + H_7^{(i-1)}.
\end{aligned}
$$

After processing all N blocks of message M, the final message digest is obtained by concatenating the hash values $(H_0^{(N)}, ..., H_7^{(N)})$. More precisely, the message digest for each algorithm is given by the concatenations shown below. The concatenation of words is represented by the symbol $||$.

$$H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}||H_7^{(N)}.$$

2.3.2 HMAC Algorithm Description

The HMAC algorithm receives two inputs, a cryptographic Key and a message (named $Text$), and produces a message authentication code (MAC). One of the goals of the HMAC algorithm is to be independent of a given hash function, so that the latter can be easily replaced to achieve higher security levels.

In this research, the SHA-2 family of hash functions is utilized. From now on, we denote the combination of HMAC with SHA algorithms as HMAC/SHA-x, where x can be 2, 256, or 512.

Some HMAC parameters and symbols are listed below.

B : SHA-2 input block size (in bits);
L : SHA-2 message digest size (in bits);
Key : Secret key of the communicating parties;
K : Size of the Key (in bits);
K_0 : Key after any necessary pre-processing;
Ipad : (Inner pad) Byte `0x36` repeated $B/8$ times;
Opad : (Outer pad) Byte `0x5C` repeated $B/8$ times;
Text : The data on which the MAC is calculated;
T : Final size of the MAC (in bits);
$min(x, y)$: The minimum of x and y;
$Hash(V)$: Hash of variable/value V;
$||(0..)_z$: Padding with z zeros.

Parameters B and L are inherited from SHA-2 algorithms. The hash of variable/value V is symbolized by $Hash(V)$, whereas $||(0..)_z$ indicates the padding with z zeros.

The length K of the Key has to be selected to ensure compatibility with the security level of the underlying hash functions, and ultimately with the application utilizing HMAC. According to [135], the security strength of HMAC is defined as $min(K, 2L)$, and T between 64 and 96 bits is assumed to be sufficient for most applications. In [97], the minimal recommendation is $K \geq L$, whereas the MAC size should obey $T \geq L/2$ and be not less than 80 bits.

The *Text* can be n bits long, where $0 \leq n < 2^{B/8} - B$, whereas the size of a MAC is L bits long. The HMAC algorithm is consisted of seven steps as shown below. Notice that the hashes computed in *Step IV* and *Steps VII* can be split into two parts to facilitate implementation. The MAC is in fact the message digest of *Step VII*.

Step I: Pre-process Key as follows:

$$K_0 = \begin{cases} Key, & K = B, \\ Key||(0..)_{B-K}, & K < B, \\ Hash(Key)||(0..)_{B-L}, & K > B. \end{cases}$$

Step II: Compute $(K_0 \oplus Ipad)$.

Step III: Do $(K_0 \oplus Ipad)||Text$.

Step IV: $Hash((K_0 \oplus Ipad)||Text)$.

Step V: Compute $(K_0 \oplus Opad)$.

Step VI: Do $(K_0 \oplus Opad)||Hash((K_0 \oplus Ipad)||Text)$.

Step VII: $Hash((K_0 \oplus Opad)||Hash((K_0 \oplus Ipad)||Text))$.

The computation of HMAC can be summarized as follows:

$$MAC = Hash((K_0 \oplus Opad)||Hash((K_0 \oplus Ipad)||Text)).$$

Precisely, the L-bit MAC is obtained from the hash variables H. A common practice is to truncate the MAC by using only its T leftmost bits. In this work, we consider that the MAC is L bits long, and any truncation is performed at the user level obeying the security requirements of the application.

2.4 Summary

This chapter introduced basic concepts on fault tolerance, described the radiation effects on electronic circuits, and also presented several mitigation techniques. Although most of design goals for fault tolerance are crucial for space systems, this research prioritizes reliability. Reliability is especially important to keep the system operating for its whole lifetime, otherwise, a fault could permanently disrupt all services provided by the spacecraft. Though numerous SEEs can occur in space applications, SEUs are one of the most common results of radiation effects and are addressed in this research. Furthermore, mitigation techniques considered in this research are mostly architectural. More specifically, techniques such as parity prediction, Hamming codes, TMR and re-computation are the main mitigation techniques considered. Combinations of the previous techniques are also taken into account, so that the resulting mitigation scheme is more efficient than the traditional TMR. Reconfiguration is considered as an option for recovery in some cases, but not the main strategy. Unlike most related work which mainly target only error detection, this research focuses on both error detection and correction.

This chapter has also shown how complex and delicate spacecrafts' subsystems are. Any minor problem in the attitude control has the potential of putting a spacecraft out of its orbital path or spinning out of control. Obviously, that has a high probability of definitively compromising the spacecraft and the entire mission. From recent events, it became clear that attackers on the ground represents a constant and serious threat to space systems. In order to provide means for more secure operations of spacecrafts, this research addresses the problem of data integrity and authentication, as well as emergency commanding and recovery. In spite of targeting mostly the TT&C subsystem, those primitives can also be

applied to the payload and other subsystems. Besides, all the cryptographic mechanisms aim at low power consumption in order to satisfy the severe restrictions imposed by the power and thermal subsystems. Although this research is based on Altera SRAM FPGAs, the proposed schemes are generic enough to be utilized by other FPGAs such as Xilinx and Actel. Finally, this research follows the recommendations and standards proposed by CCSDS as much as possible, given their widespread acceptance by the space community.

Next chapter presents previous research in hardware implementations of cryptographic primitives, including hash, message authentication codes and trusted platforms. As highlighted in this chapter, fault tolerance is a mandatory feature in space systems. Therefore, next chapter also presents some related work in fault tolerant implementation of cryptographic algorithms, e.g. hash and encryption primitives. It also includes a section on fault injection attacks, that could potentially be performed while systems are still on the ground waiting for launch or being tested.

Chapter 3

Previous Research

This chapter presents the state-of-the-art in security for space systems as well as a wide variety of hardware implementations of cryptographic primitives. These implementations, however, cannot be directly applied to space systems since they do not include any sort of fault tolerance. Finally, fault tolerant hardware implementations of cryptographic primitives are presented.

3.1 Hardware Implementation of Cryptographic Primitives

Hardware implementation of hash functions have been proposed by several works. One of the earliest SHA-256 implementation on FPGA is presented in [169], whose two main goals are minimum implementation area and high performance. In order to achieve that, a shift register approach was used to implement the SHA-256 algorithm [133]. Actually, the algorithm is divided into three main blocks: Message Scheduler, Compression Function, and Intermediate Hash.

The message scheduler, as shown in Figure 3.1, is implemented as a chain of sixteen 32-bit registers. It is responsible for receiving the incoming 512-bit message (M_t) and computing the intermediate message schedule (W_t). The message scheduler is also in charge of adding W_t with the constant K_t, and forwarding the result to the Compression Function.

Similarly to the message scheduler, the compression function is also based on shift registers, as depicted in Figure 3.2. The compression function receives the message schedule

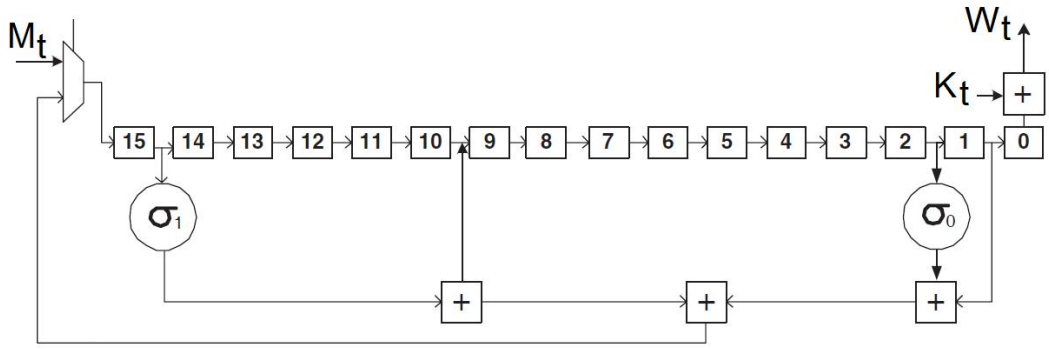


Figure 3.1: SHA-256 Message Scheduler Module from [169]

W_t and operates over the registers a through h during 64 clock cycles. In order to improve the throughput, new blocks of data can be loaded in parallel to the processing of the compression function.

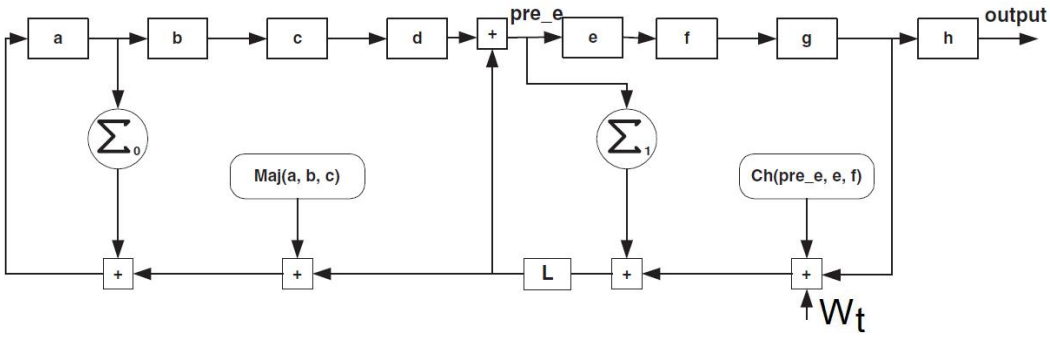


Figure 3.2: SHA-256 Compression Function Module from [169]

The third block is the intermediate hash computation. As illustrated in Figure 3.3, it obtains the eight variables a, b, c, d, e, f, g, h to compute the intermediate hash. If no additional block of data is to be processed, the final hash is obtained from registers $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$. The design was implemented in a Xilinx Virtex XCV300E-8 FPGA, and occupied 1261 slices. It obtained a throughput of 87Mbps and a maximum frequency of operation of 88MHz. Notice that this implementation considers that message padding is performed in software.

A more elaborate implementation of hash functions on FPGAs is provided in [104], where a single chip implementation of SHA-384 and SHA-512 is introduced. That is possible due to the similarities of SHA-384 and SHA-512 algorithms. More precisely, both

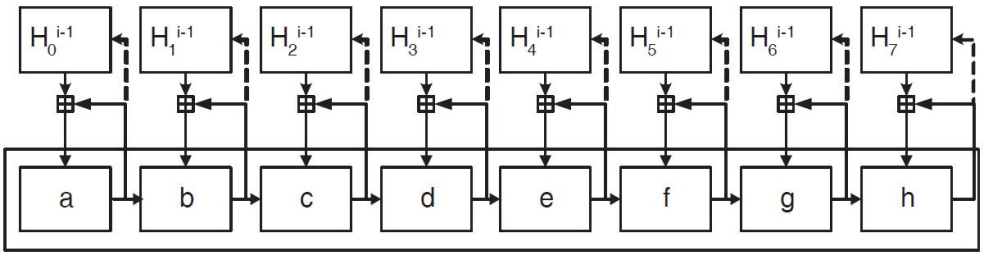


Figure 3.3: SHA-256 Intermediate Hash Module from [169]

algorithms use the same set of functions σ_0, σ_1, $\sum_0$, $\sum_1$, Maj, and Ch. Further, both use a 64-bit datapath which is implemented similarly to the one proposed in [169]. As a matter of fact, the central idea of the proposed approach is to share the entire datapath. The only difference, though, refers to the SHA-384 and SHA-512 initialization variables which are provided to the datapath through the use of multiplexers. Furthermore, the constants K_t are stored in block RAMs (BRAMs). The design presented in [104] was implemented on a Xilinx Virtex-E XCV600E-8 FPGA, and occupied 2914 slices. While operating at the maximum frequency of 38MHz it could achieve a throughput of 479Mbps. The more innovative aspect of this design is the reutilization of the datapath, which makes possible the computing of both SHA-384 and SHA-512 utilizing the same chip.

Similarly to [104], a shared datapath implementation for SHA-256, SHA-384, and SHA-512 was reported in [151]. The authors do not introduce any innovative technique to increase throughput and decrease implementation area. However, since all algorithms are implemented in the same platform, the results provide a good comparison in terms of implementation area, frequency of operation and throughput. Specifically, all algorithms implemented on a Xilinx Virtex V200PQ240-6 device required 2384 Configurable Logic Blocks (CLBs). The maximum throughput achieved for SHA-256, SHA-384, and SHA-512 was, respectively, 291Mbps, 350Mbps, and 467Mbps.

In [8] alternative architectures are explored for the implementation of SHA-2 functions. The proposed approach reduce the area requirements by narrowing down the bit width of adders, e.g. 32, 16, and 8 bits. As a consequence, more than one addition becomes necessary to add two 64-bit operands. In order to balance area and throughput, the proposed architecture utilizes Carry Save Adders (CSAs). Precisely, CSAs are formed by a set of Full Adder Array (FAA) for the intermediate additions, and a Carry Look-Ahead Adder (CLA) to perform the final sum. Figures 3.4 and 3.5 show, respectively, the proposed design of the SHA-512 message scheduler and compression function using CSAs. By using CSAs, the authors implemented a SHA-512 using 32 and 64-bit adders on

an Altera Stratix EP1S10F484C5 FPGA. The SHA-512 utilizing a 32-bit adder employed 2800 Logical Elements (LEs) and achieved a throughput of 892.8Mbps. In turn, when a 64-bit adder was used, the implementation employed 4229 LEs and reached a throughput of 1226.2Mbps.

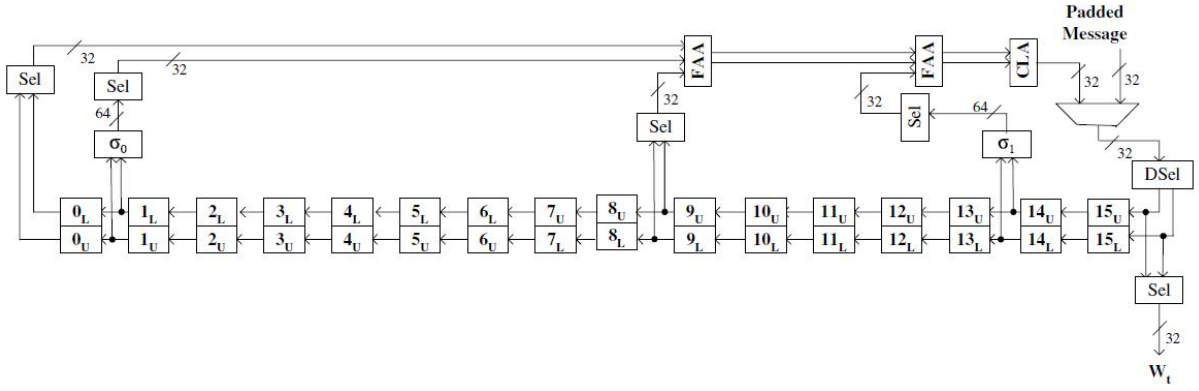


Figure 3.4: SHA-512 Message Scheduler using CSAs from [8]

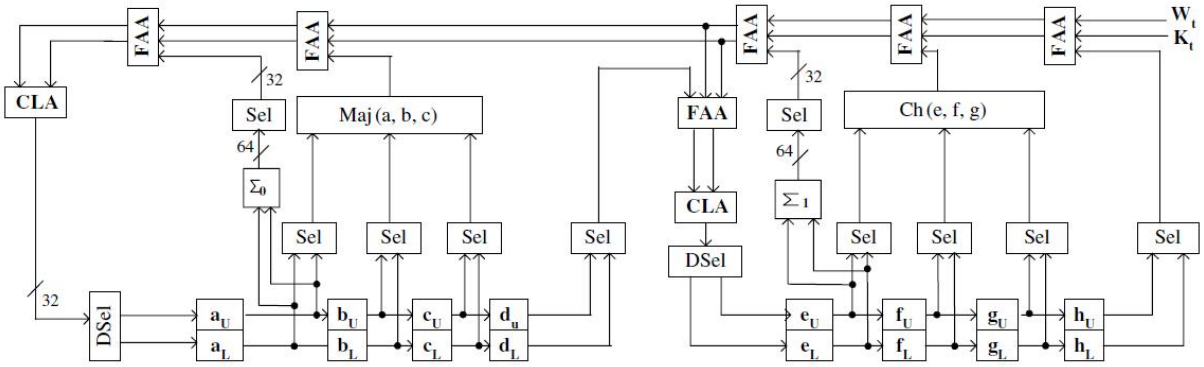


Figure 3.5: SHA-512 Compression Function using CSAs from [8]

There exist several others hardware implementations of SHA-2. Also, some optimizations based on pipelining, loop unrolling, operation rescheduling, hardware/software approaches, and hardware reutilization has also been proposed, e.g. in [109, 31, 90].

One of the earliest hardware designs of HMAC based on the SHA-1 hash function is reported in [105]. This work introduces a single-chip processor for the IPsec protocol which implements both HMAC/SHA-1 and AES on a Xilinx Virtex-E FPGA [1]. Although AES is currently used as a standard, the use of SHA-1 may not longer satisfy the security

requirements of several applications. Unfortunately, the area requirements of the individual modules are not reported, so that it is not possible to determine the area requirements of HMAC alone. Both cores can operate independently from each other, which results in a minor impact (if any) on each other throughput. Actually, the throughput achieved by this processor is 78Mbps when operating at 50MHz.

Another processor capable of performing both HMAC and SHA-1 is proposed in [149], but low throughput is achieved. The authors in [170] propose an HMAC processor which integrates both SHA-1 and MD5 algorithms, and is implemented on an Altera Apex 20K [2]. This processor occupies 5329 LEs and achieves a throughput of 34.7Mbps.

Yet another HMAC design is proposed in [94] and implemented on a Virtex-II [3] device. Due to the several hash algorithms included (SHA-1, MD5, and RIPEMD-160), this design utilizes a large implementation area (14911 slices). Besides, this processor achieves a throughput of 137.40Mbps when operating at 43.47MHz.

An ASIC implementation of HMAC, also based on SHA-1, is presented in [96]. Even though this design is implemented as an ASIC, it achieves low throughput. The architecture of this processor is shown in Figure 3.6 and is consisted of a hash module, padding and concatenation circuitry, controllers, constants and registers. Actually, the HMAC processors presented in [92], [105], [149], and [170] follows the similar architectural organization.

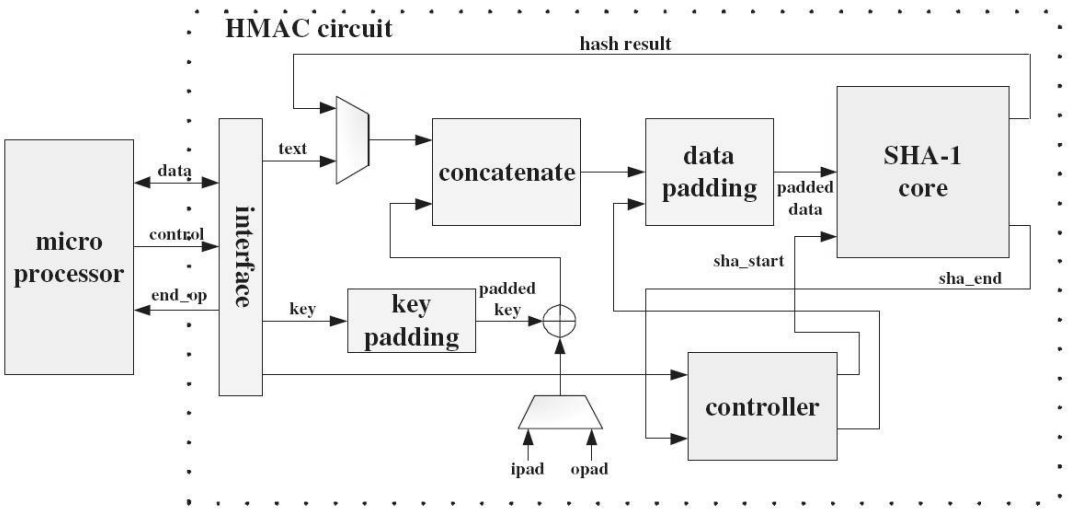


Figure 3.6: HMAC Processor Architecture from [96]

In [108] a high-performance HMAC/SHA-1 design is presented and implemented on a Xilinx Virtex-E FPGA, whose architecture is shown in Figure 3.7. This processor has

the highest throughput for HMAC hardware implementation reported in the literature (1587Mbps) when operating at 62MHz. Such a high throughput is due to the employment of two SHA-1 cores and pipelining techniques. However, the main drawback of this design is its large implementation area (6011 slices).

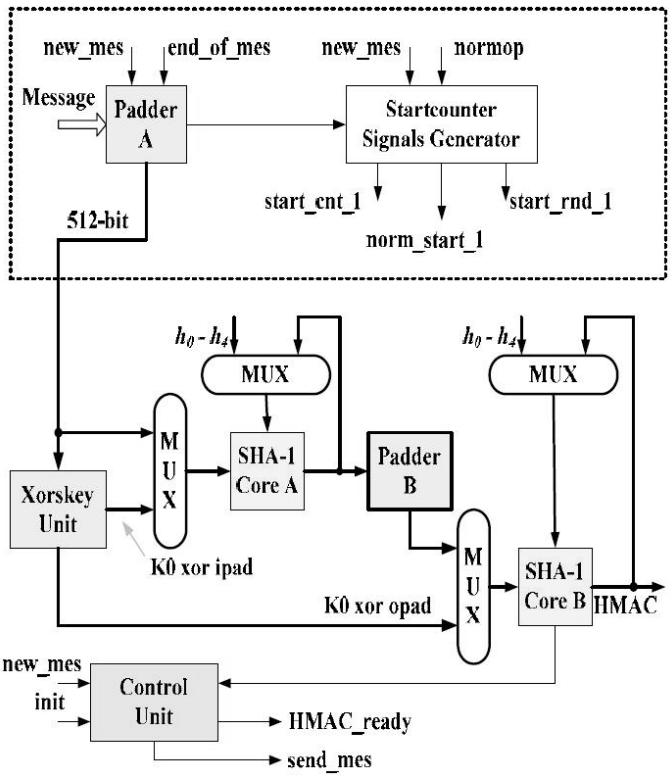


Figure 3.7: HMAC Processor Architecture from [108]

In contrast, an area-efficient HMAC hardware implementation is introduced in [171], which utilizes only 686 slices and achieves a throughput of 710.4Mbps. This high throughput is due to architectural improvements such as loop unrolling. Unfortunately the authors do not provide a discussion on how such a small area was achieved using loop unrolling. Furthermore, the capabilities of the hardware module are not specified, which makes it hard to know whether the module is fully compatible with the HMAC specification [129, 133].

Although the aforementioned researchers have explored architectural improvements to increase throughput and reduce implementation area, they have mainly focused on SHA-1 and MD5 hash algorithms. By the time being, HMAC/SHA-1 and HMAC/MD5 may no longer satisfy the security requirements of many applications. Hence, it becomes important

to consider the utilization of stronger hash functions such as SHA-2 in order to satisfy higher security requirements of recent applications.

As reported in [92], the proposed design outperforms all previous implementations of HMAC processor, such as in [170, 104, 94], in terms of implementation area and throughput. For the sake of fair comparisons, the proposed designs were cross-compiled to different FPGAs (Altera Apex 20K, Xilinx Virtex-E and Virtex-II) in order to match the exact device utilized by the aforementioned related work.

In recent years several desktop and laptops have utilized Trusted Platforms Modules (TPMs) for the provision of security functions. TPMs have been specified by the Trusted Computing Group (TCG) [73], which is an non-profit organization that aims at developing, defining and promoting open standards for trusted computing. Moreover, TPMs are currently defined as a standard by the International Organization for Standardization (ISO) [61] under standards ISO/IEC 11889-1 [57], 11889-2 [58], 11889-3 [59], and 11889-4 [60].

TPMs have been provided as integrated circuits by a number of manufacturers, e.g. Atmel [44], Infineon [6], Broadcom [46], and ST Microelectronics [111]. In spite of slight platform variations from one manufacturer to another, they usually follow the TPM architecture specified by ISO as depicted in Figure 3.8.

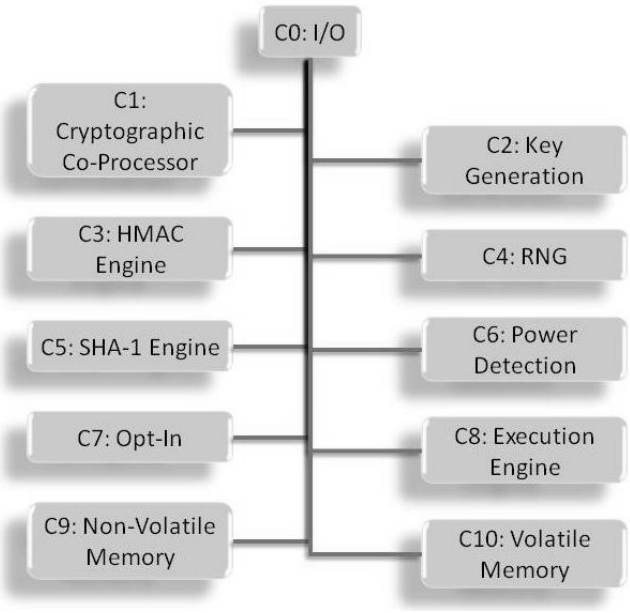


Figure 3.8: TPM Architecture from [57]

A common set of features found in [43, 20, 45, 110] is the utilization of RSA as asymmetric primitives for encryption and signatures as well as SHA-1 as the hash algorithm. Furthermore, non-volatile on-chip storage is implemented using EEPROM and sometimes ROM, whereas it is common to find SRAM for the implementation of volatile memories. These TPMs are designed for ground applications and cannot be directly employed in space applications.

Considering the overall space system security, some research [86, 146, 12] has proposed the use of satellites for key distribution and authentication in communication systems. However, the satellite security itself has not been addressed. There is limited research focusing on incorporating security on satellites in spite of constant growth in applications using them. Only in [143] a proposal is presented to generate cryptographic keys from features and properties directly associated with the actual satellite. Hence, on-board key storage could be eliminated. Even though this looks like a very interesting technique, no algorithms or specific procedures were determined on how cryptographic keys could be generated. Moreover, such a technique is only valid if we assume that an attacker never discover the features/properties of a given satellite. Otherwise, an attacker could use that information to derive future cryptographic keys.

3.2 Fault Tolerant Cryptographic Primitives

Even though the aforementioned work present a wide variety of hardware implementations of cryptographic primitives, none of them consider fault tolerance. Actually, there is very limited research on efficient fault tolerant hardware implementations of security algorithms for space applications.

To the best of our knowledge, fault tolerance was considered for SHA-512 in FPGAs only in [9]. This work is based on the same architecture presented in [8], and as such, utilizes CSAs. Fault tolerance is achieved through parity prediction. Basically, the technique consists of appending 8 parity bits to each SHA-512 64-bit word, i.e. 1 parity bit per byte. These parity bits are used to predict the output parity of the SHA operations. If the output parity does not match the predicted one, it means that either the data word or the operation was corrupted. Thus, errors can be detected.

The design of the proposed scheme applied to the compression function is illustrated in Figure 3.9. The proposed scheme was implemented on an Altera Stratix EP1S20F780C5 FPGA. Implementation results show utilization of 5038 LEs and a throughput of 372Mbps for SHA-512. This scheme supports only error detection, while error correction is not addressed at all.

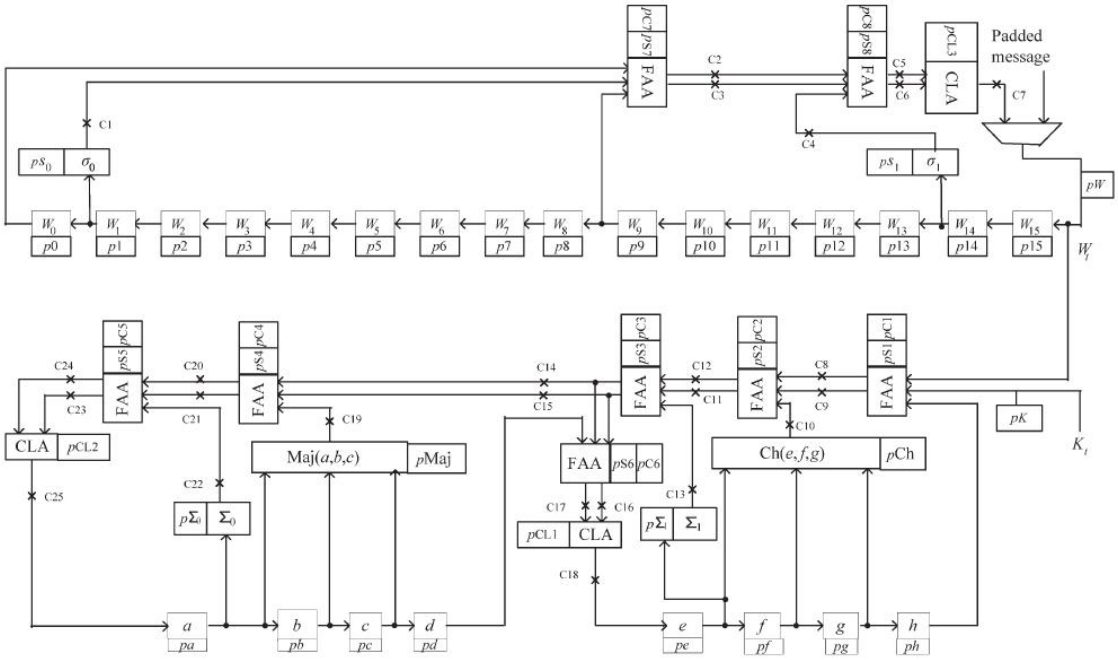


Figure 3.9: SHA-512 Datapath with Error Detection from [9]

Another fault tolerant cryptographic primitive found in the literature is presented in [15]. It aims at providing AES with fault detection and correction capabilities so that it can be used in space. Fault detection is achieved by employing parity prediction for each round of AES, as illustrated in Figure 3.10 (a). In the end of each round, the predicted parity is compared with the calculated parity. If they do not match, some remedial action takes place.

The approach for fault detection and correction presented in [15] is based on Hamming codes. Specifically, it encodes the word bytes with Hamming code. Also, as shown in 3.10 (b), this scheme predicts the Hamming code of a given round and compares with the calculated code. If they differ, some error has happened, which demands the execution of a remedial procedure to correct the bit-flip. Although the Hamming code scheme seemed very interesting, no implementation data was provided by the authors. As a consequence, its real implementation feasibility could not be evaluated.

Some fault tolerant techniques aim at performing re-computation with encoded operands. A description of such a technique can be found in [87] and [101]. This approach is capable of detecting permanent and transient faults happening in the circuit. One of the earliest works employing such a technique is presented in [144], in which error detection in ALUs

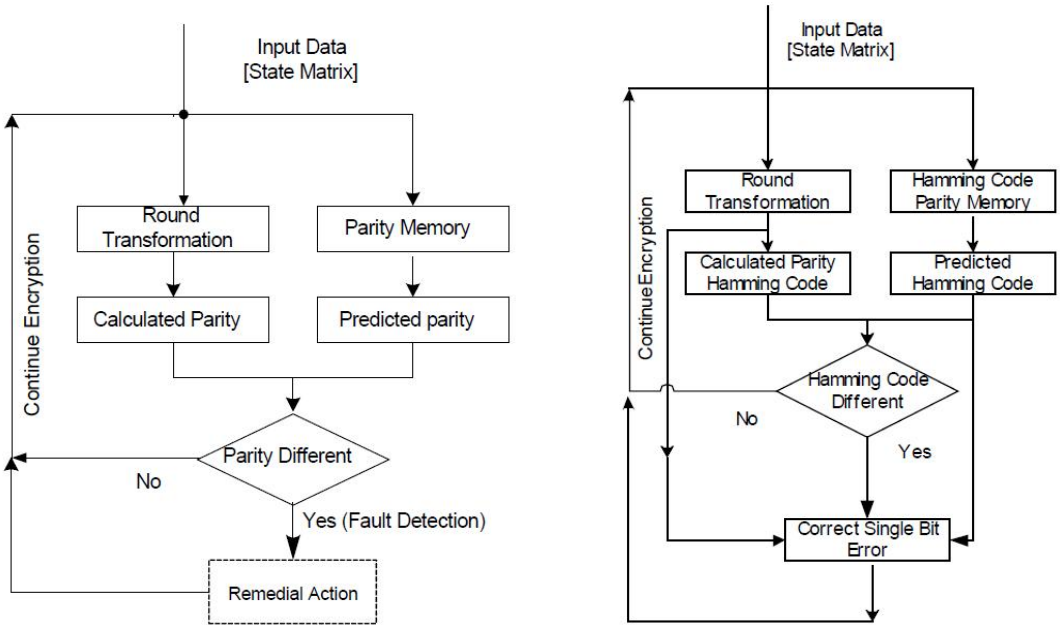


Figure 3.10: AES Round with (a) Error Detection and (b) Error Detection and Correction from [15]

are detected through the recomputation with shifted operands.

A more recent approach applied to scalar multiplication frequently used in elliptic curve cryptography (ECC) is presented in [50]. Even though ECC is not currently used in space applications, the fault tolerance technique presented in [50] can be useful in future implementations of such primitives on-board spacecrafts.

The elliptic curve scalar multiplication (ECSM) takes as inputs a point (P) and a scalar (k) and computes another point (Q), i.e. $Q = kP$. The encoding technique proposed in [50] relies on randomization of point P and scalar k in order to generate different points and scalars. Due to the efficient way the point randomization is proposed, no point decodification is required if projective coordinates are utilized. Hence, no point decoders are necessary. A block diagram of an ECSM using full recomputation with point and scalar randomization is shown in Figure 3.11.

The proposed encoding scheme, combined with the traditional TMR and DMR techniques, makes it possible for the derivation or a wide variety of fault tolerance methods for ECSM, as shown in Figures 3.12 (a) and (b).

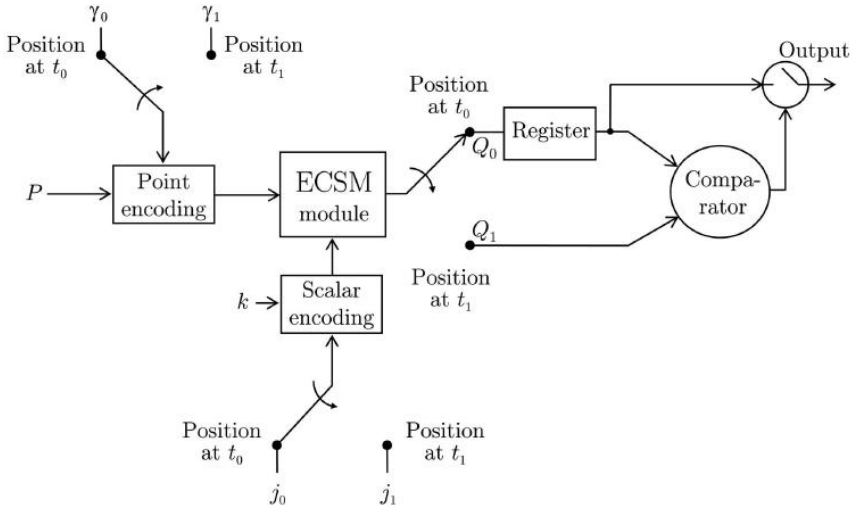


Figure 3.11: ECSM using Full Recomputation with Point and Scalar Randomization from [50]

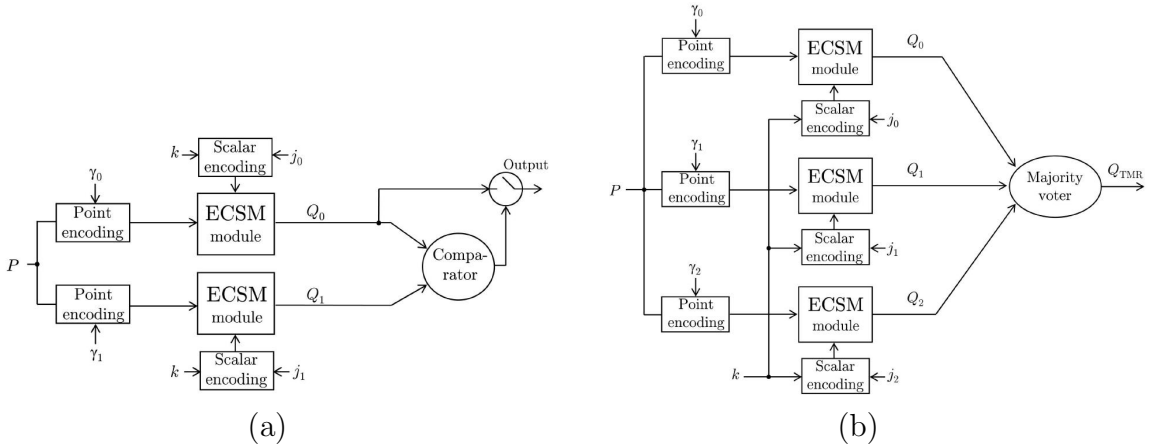


Figure 3.12: ECSM using Point and Scalar Randomization with (a) DMR and (b) TMR from [50]

Yet another scheme, denoted as parallel and recomputation, is proposed in [50] so that both low area requirements (as in DMR) and low probability of incorrect results (as in TMR) can be obtained. A block representation of such a technique is shown in Figure 3.13.

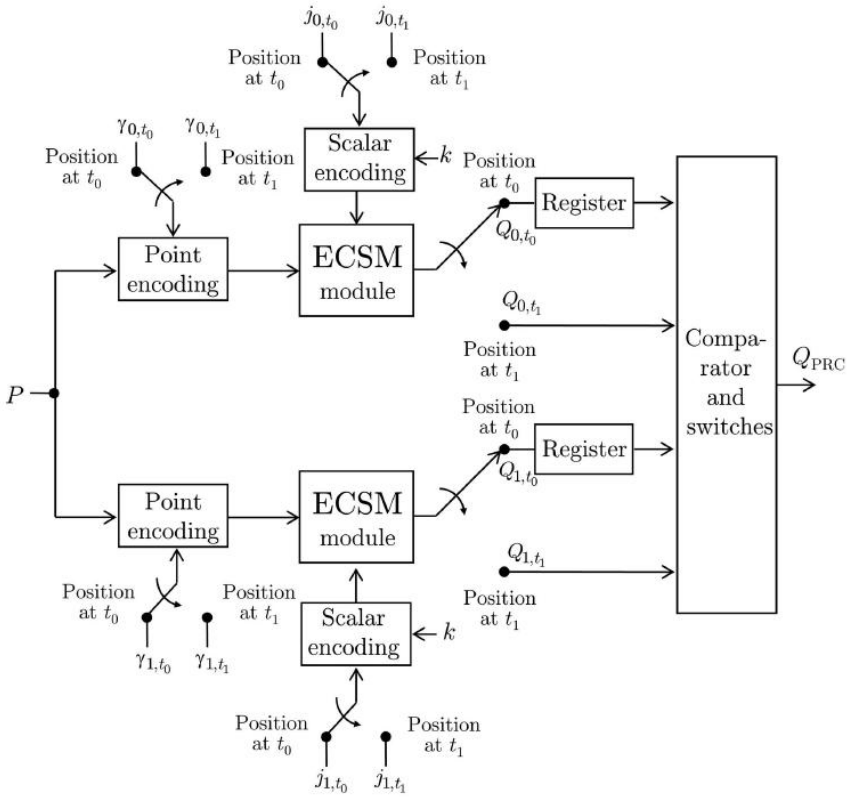


Figure 3.13: ECSM using Parallel and Recomputation with Point and Scalar Randomization from [50]

3.3 Fault Injection Attacks

The main issue being addressed by the fault tolerance techniques presented in Chapters 6 and 7 are random faults caused by SEUs. On the other hand, there are multiple ways for an attacker to intentionally inject faults in order to compromise a cryptographic system. Attackers may target a specific bit or byte, exploit injection of specific or random data, and aim at data or control errors [95]. According to [76], fault injection can be categorized as fault injection *with contact* or *without contact*.

In [16] a wide variety of types of attacks are listed, such as perturbations to the power supply and the clock frequency, variations of temperature, bombardment by light, lasers, eletromagnetic pulses and waves (although light can be considered electromagnetic radiation), X-rays and ion beams. Perturbations to the power supply has the potential

of causing the processor to skip or misinterpret instructions. Variations in the external clock can cause the cause the processor to lose synchronism with the system bus and advance/delay a memory read as well as advancing to the execution of next instruction before completing the previous one. Moreover, temperature variations beyond the nominal thresholds of the target device can cause random data modification of RAM cells. In addition, most non-volatile memories have different temperature thresholds for read and write cycles. Therefore, attacks could rely on temperature variation to exploit this feature and cause the memory to perform read but not write cycles, and vice-versa.

Photo-electric effects on electric circuits can also be exploited. In other words, silicon illumination can potentially ionize semiconductor regions and cause a transistor to conduct. As a result, a transient fault can be induced. The authors in [152], for instance, that this attack can be performed with cheap equipment, such as a $30 photo-camera flash or a $8 laser pointer. The first step consisted in removing the top packaging of a micro-controller Microchip PIC16F84 [79] to expose its 68 bytes of SRAM. The attack consisted in illuminating the memory region of the chip with a Vivitar 550FD photo flash lamp. Through a mask made of aluminum foil, the authors were capable to change the state of a single SRAM cell. This represents a fine granularity and give an attacker the power to change any memory bit he/she may want to. This is specially important in attacks that require a single bit-flip in a pre-determined position of a storage element, instead of having a region of the device set or preset.

Another attack based upon illumination is presented in [147], in which the contents of a non-volatile is erased through the utilization of ultra-violet light ($254n$m). Although the presented approach is destructive (causing damages to the device), it shows that such an attack multiple micro-controllers utilizing a variety of memory technology (EPROM, EEPROM, Flash, Fuse bits) can be compromised. Furthermore, the authors have shown how to apply such a technique to attack software implementations of AES by manipulating 256-bit S-box tables. They show through an 8-bit micro-controller that, by changing a single bit in the AES S-box, it is possible to obtain 2500 pairs of correct and faulty ciphertexts are enough to recover the key with a probability of 90%.

Although a number of other attacks targeting DES and AES have been proposed [18, 51, 19, 68, 112], there are, to the best of our knowledge, no fault injection attacks against SHA-2 and HMAC reported in the literature.

3.4 Summary

Apart from the research outlined above, there is very limited research addressing the recovery of spacecrafts that have suffered failures due to SEUs or attacks. Thus, efficient recovery techniques should be thoroughly investigated. Recovery techniques and integrity checking may involve the computation of hash functions. However, due to the sensitivity of cryptographic algorithms to bit-flips, and given the radiation effects found in space, any cryptographic mechanism employed on-board should include fault tolerance. Although hardware implementations of hash functions has been extensively researched, no error detection and correction was ever considered for this cryptographic primitive. Even though TMR has been traditionally used as an SEU mitigation technique in space applications, it usually imposes the triplication of implementation area and power consumption. As a consequence, more efficient schemes should be investigated to reduce implementation area and power consumption of fault tolerant integrity checking and authentication schemes.

Next chapter introduces a trusted platform to enable control centers to recover spacecrafts from failures and attacks. Furthermore, Chapters 6 and 7 presents efficient fault tolerance techniques to perform both error detection and correction in hardware implementations of SHA-2 and HMAC algorithms.

Chapter 4

Secure System Recovery

This chapter addresses the problem of securely recovering spacecrafts' computational platform in case of major failures and attacks. Two approaches are proposed which are based on different threat and trust models. The first approach is based on hash functions and can be employed in current systems already in space. The second one is based trusted modules and aims at achieving higher levels of security therefore addressing harsher threat models. The trusted modules approach is analyzed against exhaustive search attacks which, to the best of our knowledge, is the only way to break the system. Exhaustive search attacks against the proposed mechanism are shown to be infeasible, independently of the computational power and communications capabilities of ground-based attackers. Through experimental results it has been shown that the proposed recovery strategy allow for the secure recovery of spacecrafts, including contingency situations, demanding minimum implementation area, power consumption and communication capabilities.

4.1 Reference Platform

The spacecraft's computational platform considered in this chapter is assumed to be comprised of a general-purpose processor, an FPGA, volatile and non-volatile memories, communication modules, a reconfiguration circuit, as well as payload modules. Those elements are inter connected through a common bus. Additionally, a Joint Test Action Group (JTAG) connection links the reconfiguration circuit to the FPGA. Such a platform is illustrated in Figure 4.1. Since this platform does not involve any trusted module, it is denoted as *Untrusted Computational Platform*.

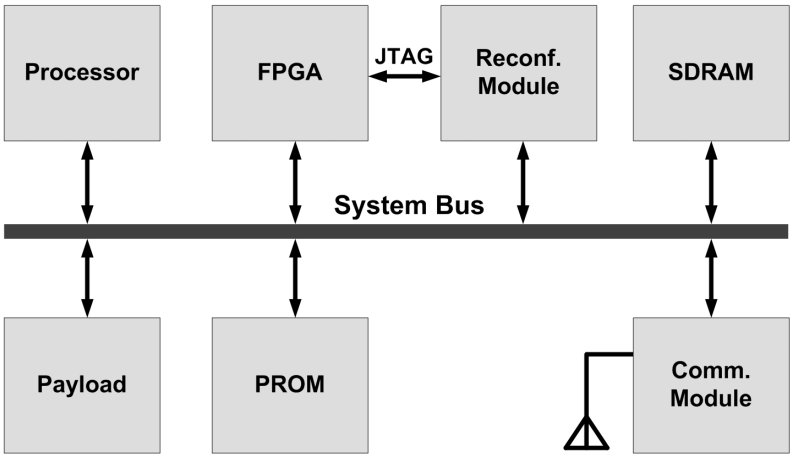


Figure 4.1: Untrusted Computational Platform

The processor is employed in general-purpose computational tasks, including communications activities, commands and telemetry processing, collection and processing of mission data from payload modules, and management of payload activities, to cite a few. The processor is connected to the system bus, where it can access all the peripheral modules.

The FPGA provides a flexible hardware platform that can complement the processor capabilities and provide hardware acceleration to more complex tasks. Like the processor, the FPGA connects to the system bus. In addition, a JTAG connection links the FPGA to the Reconfiguration Module.

A volatile memory, usually SDRAM, is available for storing the processor's program as well as application's data. This memory can be accessed by the processor, FPGA, communication modules, and payload modules through the system bus.

Besides, a non-volatile memory, usually a ROM, provides permanent storage of basic functionalities of the spacecraft, which includes a standard program as well as a configuration file to provide basic functionalities for both the processor and the FPGA.

The reconfiguration module is in charge of initializing the computational platform, i.e. the processor's program memory and the FPGA. The processor programs are copied from the ROM to the SDRAM memory through the system bus. Besides, this module configures the FPGA through a JTAG connection.

The communication modules performs the tasks of sending and receiving radio frequency (RF) signals to and from ground stations. As such, these modules have a direct connection with the spacecraft antennas. These modules are also in charge of interfacing

the system bus by encoding and decoding RF signals into digital signals. The security level of each mission will dictate the security requirements of the communications system, which may or may not include encryption, data origin authentication and data integrity. These functions can be provided by cryptographic mechanisms accommodated within the communication modules, or within the processor stack of protocols for secure communications. Security mechanisms and implementation details are individually defined in accordance with mission-specific requirements.

Last but not least, the payload modules are responsible to execute the specific tasks for which the spacecraft was designed. For instance, meteorological and scientific spacecrafts may monitor different wave lengths coming from Earth and deep space. Communication and navigation satellites may receive and send signals to conduct their respective tasks and therefore may also connect to antennas. In sum, their functions varies with the mission type and mission-specific requirements.

4.2 Hash Based Approach

It is important to mention that the hash-based approach targets contingency situations where no secured channel is in place due to the loss of cryptographic key. Put differently, if the spacecraft has the capability of performing key establishment based on private-key or public-key primitives, that should be adopted as the preferred strategy.

4.2.1 Trust and Threat Model

The trust model considered in this approach assumes that:

- the spacecraft has some hardwired information, represented by k', built-in during the construction of its computational platform;
- k' is known by the institution building the spacecraft's hardware and is disclosed solely to the agency controlling the spacecraft;
- k' has to be readable by the processing elements of the computational platform such as microprocessors and FPGAs;
- the medium storing k' should be immune to SEUs; and
- the spacecraft has access to a hash algorithm on-board.

Notice that it may happen in multiple occasions that the agency controlling the spacecraft is the same as the one that has built it. In other cases, the controlling agency can provide hardware modules to the contractor, therefore reducing the risk of leaking k'.

Considering current spacecrafts, k' could be serial numbers, ID numbers, or any other data that only the spacecraft and the control center know. It would be even better if k' was randomly created during the spacecraft construction, so that the likelihood of guessing k' would be minimized. Moreover, a bigger k' could be formed by concatenating more than one source of information. In new spacecrafts, k' should be randomly generated.

It would not be appropriate to rely on k' if it is stored in a volatile FPGA since it could get corrupted by SEUs or lost during reconfiguration. Furthermore, k' must never be used as a key, but as a seed for deriving temporary keys.

The threat model considered for the hash-based approach assumes that an attacker:

- can listen to all RF signals used for communications (ground station–spacecraft, spacecraft–spacecraft);
- has the capability of sending his/her own RF signals to the spacecraft;
- knows the data format of an authentic control signal used by a valid control center;
- does not know or have access to k'; and
- never breaks into the spacecraft. In other words, it is assumed that the attacker cannot tamper or reconfigure elements of the on-board computational platform (e.g. program memory and FPGA configuration).

The hash-based scheme is completely dependent on the secrecy of k'. Hence, this piece of information must be kept secret at all times. Once this value is discovered or leaked by any means, the hash based scheme can never be reused to re-establish a secured channel with the control center. Once k' is read by an attacker, he or she has learned the most important piece of information that is used to create cryptographic keys. Actually, this is the reason to assume in this threat model that an attacker cannot break into the spacecraft. If someone is able to reconfigure the processor or the FPGA for example, it would be trivial to command those elements to read k' and send it to the attacker via RF.

4.2.2 Key Generation Protocol

Given the aforementioned trust and threat model, the recovery can be performed by the protocol shown in Figure 4.2. The control center and the spacecraft maintain an l-bit counter, e.g. $l = 32$ bits, which is used to determine n. The control center sends n to the spacecraft. Upon the reception of n, the spacecraft compares the received n with the value of its internal counter. If they match, the key generation takes place.

To construct a new symmetric key k, the spacecraft concatenates k' with n, and computes its hash (represented by $H()$). More precisely, $k := H(k'||n)$. In addition, the control center performs the same computation. At this point, the spacecraft and the control center share the same session key k. Upon finishing such a computation, the spacecraft sends a status message to the control center informing that it is possible to re-establish a secured channel. If a secured channel can be in fact established, both control center and spacecraft increments the counter defining n.

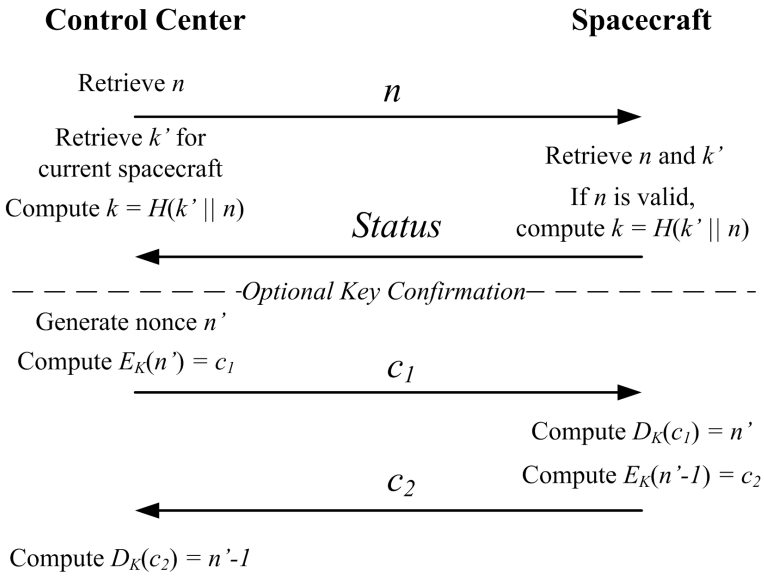


Figure 4.2: Protocol for Hash-Based Key Generation

This approach is relatively simple, only requiring the computation of one hash. For the procedure above, no encryption is utilized which contributes to speedup the procedure, save processing power and reduce power consumption.

Encryption is only needed in the case of requiring key confirmation. Depending on the space mission, such a step may be optional. It could be performed through the use

of a nonce, n', generated by the control center. The control center would perform the encryption of n', represented by $E_K(n')$, and send the cyphertext c_1 to the spacecraft. The spacecraft would decrypt c_1, represented by $D_K(c_1)$, to recover n'. Next, the spacecraft would create a cyphertext c_2 by encrypting $(n' - 1)$, and send c_2 to the control center. Finally, the control center could verify that it shares the same key with the spacecraft by decrypting c_2 and checking whether it received $(n' - 1)$ from the spacecraft.

Observe that n brings freshness to the key generation, ensuring that every time n changes, so does the key k. Thus, it is possible for the spacecraft to frequently change the cryptographic keys. This feature is also useful to prevent replay attacks. The bit-length of k' and n has not been specified since those parameters are application-specific and directly dependent on the security level of the space mission. The generated key k should be long enough to match the mission security level and a hash function should be chosen accordingly.

Assuming the utilization of cryptographic hash functions, it would be infeasible for an attacker to compute k from n, if k' remains unknown to the attacker. As a consequence, only the control center and the spacecraft can generate the same key k since they are the only two entities knowing k'. Due to its simplicity, it can be used in contingency situation in either existing or new spacecrafts, and also reused as many times as it becomes necessary by simply changing the nonce n.

The main disadvantage of the hash-based approach is that it does not tolerate any attacker breaking into the spacecraft. Also, it always relies on cryptographic primitives available for use such as hash functions and encryption. In this approach there is no way to hide the k' from someone controlling the processing elements of the computational platform. Besides, it is not possible to recover the spacecraft from the hands of an attacker since he/she can reconfigure the computational platform at his/her own benefit.

4.3 Trusted Module Approach

This section aims at providing a reliable methodology to recover the spacecraft from failures or even if an attacker has gained control over it. The trusted platform allows for the computational platform to be brought to a safe state as well as for the recovery of its cryptographic capabilities, even when an attacker has gained control over the spacecraft.

4.3.1 Trust and Threat Model

The trust model considered in this approach relies on on-board secrets to issue trusted resets and temporary cryptographic keys. The trust model assumes that:

- the trusted modules store a set of secret information;
- only the control center (authentic entity) knows the secret information;
- once a secret information is used, it is destroyed;
- a recovery mechanism handles corrupted on-board secrets, so that the system stay synchronized with the control center;
- a cryptographic hash is available to perform the integrity check of the untrusted elements of the computational platform;
- there is a direct connection between the trusted modules and the communication module; and
- the recovery can rely on a non-volatile memory that contains a default system configuration data.

The secure path between the communications module and the trusted platform is mandatory so that a man-in-the-middle attack within the spacecraft's circuitry is impossible. This way, all the commands received from a control center are guaranteed to be sent to the trusted modules unmodified.

A default system configuration is stored in a non-volatile memory (e.g. PROM), and it contains basic functionalities that will provide the spacecraft with a safe operational state. More precisely, the PROM stores a minimum program to be loaded into the processor's program memory and the FPGA configuration file. Even though this threat model assumes that FPGA reconfiguration may occur, addressing the reconfiguration details is out of scope for this research.

This threat model allows for the consideration of harsher scenarios compared to the hash-based approach. However, it requires the utilization of trusted modules on-board spacecrafts. Thus, the trusted module approach can only be applied to spacecrafts under planning, and eventually, under construction.

Since the trusted modules described are meant to operate in space, they are made fault tolerant. As such, besides being immune to SEUs their functionalities should not be

modified by an attacker. The remaining of the computing platform, which might include processors and FPGAs, is considered as untrusted. Therefore, the threat model assumes that an attacker can:

- can listen to all RF signals used for communications (ground station–spacecraft, spacecraft–spacecraft);
- has the capability of sending his/her own RF signals to the spacecraft;
- knows the data format of an authentic control signal used by a valid control center;
- explore security breaches and eventually break into the spacecraft to modify the FPGA configuration and the program memory, which would give him/her plenty control over the spacecraft's untrusted computational platform;
- interface directly with the trusted modules.

Due to the way the trusted modules are implemented, such an attacker would not have the ability to access the secrets stored within them. Even exhaustive search would be infeasible for an attacker with direct access to the module interface. Hence, the trusted modules can be used as a trust point in the event of a system recovery.

4.3.2 Trusted Computational Platform

The trusted computational platform is based upon the platform introduced in Section 4.1 with the addition of trusted modules (TMs). Man-in-the-middle attacks in the system bus are avoided by splitting the system bus in two main sections which are interconnected by the (TM), as shown in Figure 4.3.

Notice the all elements of the platform may be tampered with by an attacker, but the TM. Therefore, the TM is always capable of performing integrity check, ordering system reconfiguration, as well as receiving untampered commands from control centers without the (man-in-the-middle) bus intervention of an attacker.

Furthermore, the TM divides the system bus in two segments. During normal operation of the computational platform, the TM simply by passes the signals from one segment to another. However, during recovery procedures, the TM can isolate the system bus from crucial components of the platform such as the communication modules and SDRAM in order to isolate the range of possibilities of an attacker controlling the processor or FPGA.

Therefore, it is possible for the TM to perform two fundamental tasks: 1) receive authentic commands from the ground station to initiate the recovery process; 2) check the integrity of the SDRAM and FPGA configuration, the latter performed through the JTAG connection.

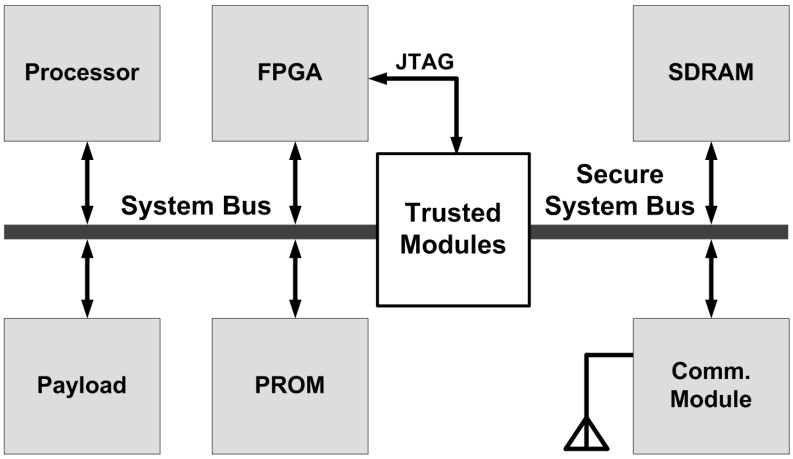


Figure 4.3: Trusted Computational Platform

Main goal of each TM is to be as simple as possible to provide high reliability, efficient hardware implementation, and allow for fast recovery procedures. The recovery sequence is: 1) detect lack of integrity on the computational platform; 2) bring the platform to a reliable state; and 3) restore a secured communication channel with the control center.

The TM is further subdivided in three main modules, namely *Trusted Hash and Configuration Module* (THCM), *Trusted Reset Module* (TRM), and *Trusted Key Recovery Module* (TKM). A random number generator (RNG) is also considered part of the platform, but since RNGs hardware modules are easily found nowadays, they are not discussed in this research.

4.3.2.1 Trusted Hash and Configuration Module

The Trusted Hash and Configuration Module is responsible for checking the integrity of the computational platform. More precisely, it is responsible for hashing the contents of the program memory and the current FPGA configuration through the utilization of a secure hash function such as SHA-2. This trusted module also contains some configuration circuitry employed in the re-initialization of the processor's program memory and in the FPGA re-configuration. Basically, this is a similar approach to the read-back and

reconfiguration technique introduced in Chapter 2. As can be noticed, fault tolerant hash functions must be employed in this trusted module to perform cryptographic hashes. Efficient schemes for fault tolerant hash functions are presented in Section 6.

By relying on the THCM it becomes possible to detect when one of the spacecraft processing elements gets corrupted. The following step is the recovery of the spacecraft integrity. This process starts with the issue of a trusted reset. Next, the key recovery takes place, which allows for the recovery of a secured channel with the control center. Although the trusted key issue usually follows a trusted reset, the following two sections present first the TKM and then the TRM. The reason for that, is the simplicity in understanding the idea behind the TRM once the TKM is presented.

4.3.2.2 Trusted Key Recovery Module

The Trusted Key Recovery Module is responsible for recovering cryptographic keys. This module employs a table to store l keys $(k_1, k_2, ..., k_l)$. This table is indexed by a number n, which is b bits long, thus leading to an address space of size 2^b. Actually, n is denoted as *One-Time Key Recovery Secret* and is sent by a control center to the spacecraft in the event of a key recovery (the recovery protocol is discussed in detail in Section 4.3.5). If the memory position indexed by n holds a key, such a key is output. It is important to mention that n should never be used as a key. Otherwise, any entity listening to the communication could use n to decrypt future encrypted messages.

The strength of this module relies on two fundamental features: i) the l cryptographic keys are randomly distributed throughout an address space of size 2^b, and ii) b is big enough to make it infeasible for an attacker to find a position containing a key through exhaustive search. An abstraction of this module is presented in Figure 4.4.

Even though a huge address space is considered for this approach, only l elements are necessary for key storage. Precisely, the number of key recoveries planned for the system defines l. Therefore, the greater l, the longer the spacecraft protection against key losses. On the other hand, the smaller l, the less storage is needed on-board. Some scenarios considered in this chapter, for example, employs $l = 64, 128$, and 256, along with $b = 64, 128$, and 256 bits and 128-bit keys.

A fundamental feature that implies in the security of TKM is that, once a given key is read, it is removed from the secrets table and never used again. Thus, the reutilization of n is automatically forbidden by the TKM. As a result, replay attacks are impossible to be performed against the system.

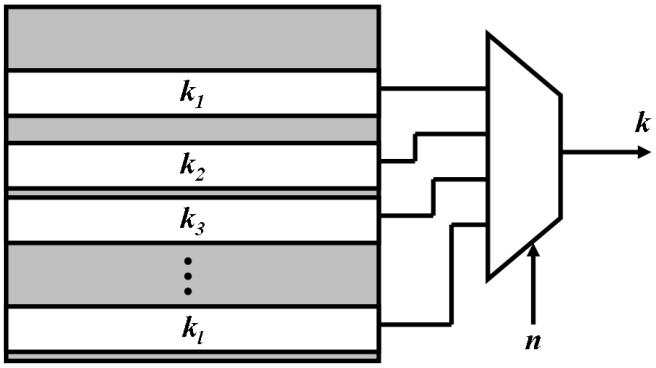


Figure 4.4: Trusted Key Recovery Module

4.3.2.3 Trusted Key Reset Module

The Trusted Reset Module is responsible for issuing authenticated reset signals, which is subsequently used to bring the spacecraft to a reliable state from which the rest of the system can be recovered. The reset signal causes the FPGA to be reconfigured (with minimum capabilities) and the program memory to be restored. Although not always necessary, the trusted reset would usually precede a trusted key recovery.

The TRM works similarly to the TKM. It receives an b-bit secret s, which is now denoted as *One-Time Reset Secret*. Similarly to the TRM employs a table to store reset secrets, which is in turn indexed by s. However, instead of outputting a key it issues a trusted reset. More precisely, the reset signal is issued when the trusted module receives a valid s.

Furthermore, once a reset position is used, it is automatically destroyed and therefore can never be reutilized. This prevents an attacker from using an old reset secret s to perform a replay attack.

4.3.3 Trusted Modules Hardware Design

The design of the TRM and TKM consists of main four components: control unit, address counter, secrets table and compare secret unit. In particular, the TKM also employs a table to store the cryptographic keys. The internal architecture of the trusted modules is shown in Figure 4.5.

The control unit is responsible for receiving the one-time secret and coordinating its evaluation. The whole process takes four clock cycles to: 1) access to the secrets table; 2) compare the stored secret with the received one; 3) destroy the used secret; and 4) increment the address counter. Even though the TRM and TKM have different functions, their architecture are very similar. The only difference, represented by the dashed lines in Figure 4.5, is the additional circuitry for managing the cryptographic keys within the TKM.

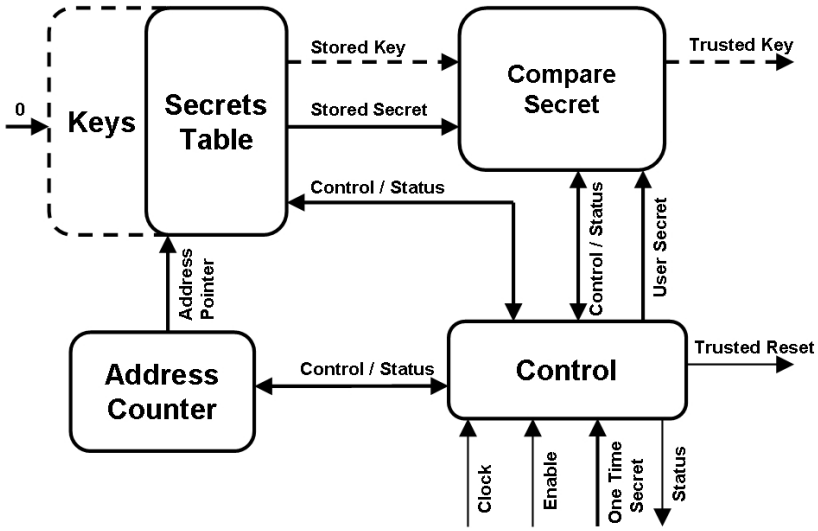


Figure 4.5: Trusted Reset and Key Recovery Modules Architecture

The secrets table stores l secrets, each of them b bits wide, which are protected against errors by the use of hamming codes. Actually, all secrets are stored in an encoded form, by adding p parity bits for each storage element. When the memory is read, an internal Hamming decoder corrects an eventual bit-flip caused by SEUs and sends the secret to the secret compare unit. The implementation of TRM considered in this chapter explores secrets as large as 64, 128 and 256 bits, which require the addition of 7, 8, and 9 parity bits, respectively. Hence, the memory requirement to store the secrets is given by $l*(b+p)$.

In the case of the TKM, secrets and keys are stored together in the same storage element. This joint storage allows for the saving of parity bits in its Hamming encoded form, thus requiring only 8, 8, and 9 parity bits, respectively to encode the 64, 128 and 256-bit secrets along with the 128-bit keys. The memory requirement of TKM to store the secrets along with the keys is $l * (128 + b + p)$.

Furthermore, if an unrecoverable error is found in the stored secret or key, e.g. two bit-flips, the control unit is notified by the Hamming decoder. Consequently, the address counter is incremented and the control center instructed to use the next one-time secret. One-time secrets are indexed sequentially through a pointer generated by the address counter unit.

The address counter unit consists of a fault tolerant ($log_2 l$)-bit counter, allowing for up to l resets/key recoveries. It relies on a Hamming encoder/decoder pair to keep the counter in an encoded form, so that it can correct any bit-flip that might have occurred in the counter register.

Once the stored secret is read from the secrets table, it is sent to the compare secret unit. The compare secret unit performs a bit-wise comparison between the stored secret and the one-time secret under test. From the comparison results, the control unit determines whether or not to issue a reset signal (a key, in the case of the TKM). If the one-time secret was "guessed" correctly, i.e. it is a valid secret, the secrets table position where it was stored is zeroed. In the case of the TKM, the corresponding key is also destroyed in the keys table. Finally, the control unit increments the address counter therefore preparing the trusted module for the next recovery process. In each counter increment, the counter register is re-encoded to reflect its new value.

Yet another fault tolerant mechanism is adopted to protect the address counter, and therefore increase the reliability of the trusted modules. In case of an uncorrectable error in the address counter, its value can be reset by the control unit. In the sequence, the control unit executes a series of counter increments, until it detects that the secret coming from the secrets table is non-zero (all used secrets have been zeroed). This mechanism also protects from the event of having a bit-flip in a zeroed element, which could potentially be mistaken as a valid secret. However, its parity bits would not match and the Hamming decoder would automatically discard such an invalid secret. At this point, it knows that the address pointer has been recovered to its correct position.

4.3.4 Experimental Results

In order to analyze the feasibility of the proposed architectures, hardware implementations of TRM and TKM were carried out utilizing an Altera CycloneII EP2C35F672C6 FPGA. The tool employed for the hardware description, synthesis, place-and-route, as well as for power estimations was QuartusII version 7.2.

As shown in Table 4.1, the simplest implementation of TRM is based on 64-bit secrets and can issue up to 64 reset signals. In this configuration, the module occupies 502 LEs,

uses 4608 memory bits to implement the secrets table. When operating at 62.24MHz this module consumes 10.78mW of dynamic power. The sum of I/O power dissipation with static power dissipation for TRM is about 81mW.

Table 4.1: TRM Implementation Results

Secret (bits)	# of Resets	Area (LEs)	Memory (bits)	Frequency (MHz)	Dynamic Power (mW)
64	64	502	4608	62.24	10.78
	128	507	9216	63.50	11.03
	256	512	18432	63.03	12.25
128	64	887	8768	59.01	14.83
	128	892	17536	58.97	16.17
	256	901	35072	58.83	16.80
256	64	1642	17024	57.14	17.37
	128	1647	34048	56.06	19.28
	256	1654	68096	54.18	20.56

The most secure version of TRM, which uses 256-bit secrets and is able to issue up to 256 reset signals, occupies 1654 LEs and utilizes 68096 memory bits. This module runs at 54.18MHz and at that frequency it consumes 20.56mW.

Compared to the least secure version, the more secure one demands 3.3 times more area, almost 14.8 times more memory bits, and consumes 1.9 times more dynamic power.

For the sake of comparison, similar hardware implementation is performed for TKM, i.e. using the same secret sizes and number of key recoveries as the TRMs. The results reported in Table 4.2 are based on the same hardware implementation parameters employed for the TRMs.

Furthermore, the results shown in Table 4.2 refer to the TKM issuing 128-bit keys. In its simplest version, the module uses 64-bit secrets and can issue up to 64 keys. This module occupies 1213 LEs and employs 12864 memory bits. While operating at 56.73MHz this module consumes 21.34mW. Again, the I/O and static power dissipation of all TKMs is about 81mW.

On the other hand, the most secure version of TKM is based on 256-bit secrets and allows for 256 key recoveries. This module utilizes 2372 LEs and requires 100864 memory bits. It consumes 40.65mW when operating at 50.73MHz. Moreover, this module occupies 1.95 times more area, and consumes 1.9 more dynamic power than its simplest version.

Table 4.2: TKM Implementation Results (128-bit Keys)

Secret (bits)	# of Keys	Area (LEs)	Memory (bits)	Frequency (MHz)	Dynamic Power Power (mW)
64	64	1213	12864	56.73	21.34
	128	1219	25728	55.34	20.92
	256	1224	51456	58.33	23.33
128	64	1674	17024	55.78	29.81
	128	1686	34048	55.19	29.97
	256	1691	68096	57.57	31.96
256	64	2356	25216	52.97	38.05
	128	2356	50432	51.75	39.23
	256	2372	100864	50.73	40.65

The processing time of each module is listed in Table 4.3, from which it is possible to observe that all modules are quite efficient in terms of processing time. The slowest and the fastest TRM have processing times 73.83ns and 64.27ns, respectively. Similarly, the slowest TKM executes a key recovery in 78.85ns, while the fastest one performs the same operation in 68.58ns.

Table 4.3: TRM and TKM Processing Times

Secret (bits)	# of Keys/Resets	Processing Time (ns) TRM	TKM
64	64	64.27	70.51
	128	62.99	72.28
	256	63.46	68.58
128	64	67.79	71.71
	128	67.83	72.48
	256	67.99	69.48
256	64	70.00	75.51
	128	71.35	77.29
	256	73.83	78.85

4.3.5 Recovery Protocols

From time to time, during the normal operation of the spacecraft, the THCM checks the integrity of the SDRAM and the FPGA's current configuration. This is based on a cryptographic hash and cannot be disturbed by any attacker, even if he/she gained control over the processor. That happens due to the bus isolation provided by the TM as well as by the JTAG bus. Hence, if any lack of integrity of the program memory or FPGA configuration will be discovered. Again, the lack of integrity may have been cause unintentionally due to radiation, or intentionally by an attacker.

The computed hashes are then sent to the control center. Since the TM have a direct hardware connection with the communications modules there is no means for an attacker to disrupt the communication with the control center. Thus, the control center will be informed of any data corruption on the system's components, when it happens. Given that the control center knows the hashes of the FPGA configuration and program memory, a simple comparison determines whether the spacecraft has some crucial component corrupted. If necessary, the recovery procedure will be initiated, as depicted by the challenge-response protocol in Figure 4.6.

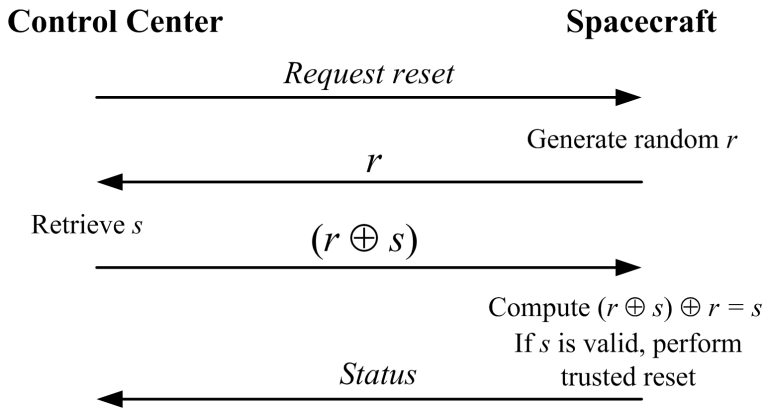


Figure 4.6: Protocol for Trusted Reset

In order to proceed with the trusted reset, the control center first requests the initiation of the trusted reset protocol. Then the spacecraft generates a random number r and sends it to the control center. After that, the control center picks a fresh one-time secret s, performs an exclusive-or ($\oplus$) of s with r, and responds $(r \oplus s)$ to the spacecraft. Since the spacecraft knows r and $(r \oplus s)$, it performs $(r \oplus s) \oplus r$, and recovers s.

After determining s, the TRM indexes its secrets table. If s corresponds to a valid

address, a general reset signal is issued. Next, the content of the accessed address is destroyed. In other words, an attacker could listen to s but could never reuse it in a replay attack to issue a new system reset. Therefore, this action is performed only once and uniquely by the holder of the one-time reset secret s. As a result of the reset signal, the THCM restores the contents of the FPGA and the program memory to a default configuration. The default configuration includes cryptographic primitives that can be used later to re-establish a secured channel with the control center.

In the sequence, the spacecraft will send a status message to the control center, which contains information on the success of the reset operation. The spacecraft could optionally compute a new hash of the FPGA and memory contents therefore providing confirmation that the contents of the aforementioned elements have been restored to their default configuration.

After these steps, the spacecraft has recovered the integrity of its computational platform, i.e. it is in a safe state. However, it does not have a secured channel with the control center. At this point the key recovery protocol takes place, as illustrated in Figure 4.7.

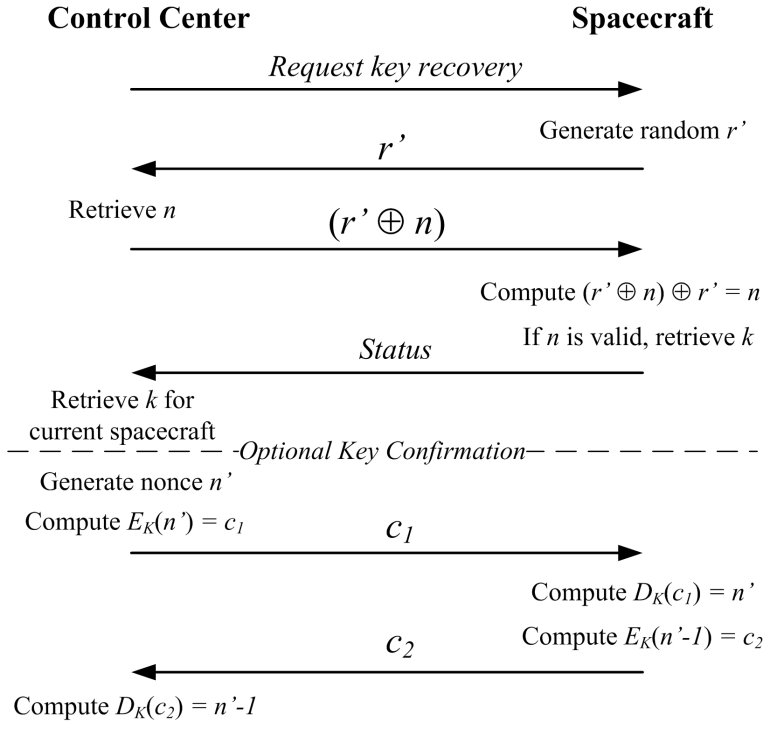


Figure 4.7: Protocol for Trusted Key Recovery

Similarly to the trusted reset, the key recovery procedure starts with the control center requesting the initiation of the protocol. For this protocol, the one-time secret is denoted as n. The spacecraft generates a new random number r' and sends it to the control center. Then, the control center computes $(r' \oplus n)$ and sends it back to the spacecraft. Next, by knowing r' and $(r' \oplus n)$, the spacecraft recovers n. In the sequence, the TKM uses n to index its secrets table. Finally, the secrets table outputs a cryptographic key, which is followed by the re-establishment of a secured and authenticated channel with the control center. Likewise the hash-based approach, key confirmation is optional and dependent on the requirements of the space mission. Key confirmation would proceed exactly the same as presented for the hash-based approach.

Additionally, the control center can use the (new) secured channel to send new program and FPGA configuration files to the spacecraft. This may be an important step, since attacks or failures may have occurred due to bugs or security holes in the previous system's embedded hardware (FPGA) or software (program memory).

Although a worst case scenario has been described, the recovery can be simpler depending on the type of the failure. For example, it may be necessary only to recover a cryptographic key, where the spacecraft reset may not be needed.

4.3.6 Resistance Against Attacks

The previous section discussed the recovery procedure considering a 3-way challenge-response protocol. One may argue that it would be much faster and cheaper not to use the challenge-response protocol, and instead of that, send s and n directly to the spacecraft. That is a valid argument, however the challenge-response protocol helps to increase the security of the system, and simultaneously lower implementation requirements by taking into account the time t spent in the communications between the ground station and the spacecraft. More precisely, the communication time t (in seconds) is determined by $t = d/c$, where d is the distance (in Km) between Earth and the spacecraft, and c is the speed of light in vacuum (299,792.458 Km/s).

This point becomes much clearer when observed from the perspective of an attacker. Without the challenge-response protocol, an attacker could speedup a exhaustive search attack by sending a stream of (invalid) one-time secret, one after the other, to the spacecraft. In fact, the delay between two trials would be the time spent by the spacecraft to process the one-time secret, which is performed quite quickly. For instance, the trusted modules take at maximum $78ns$ to perform a trusted key recovery.

Now, considering the challenge-response protocol, an attacker cannot perform any better than exchanging 3 messages with the spacecraft. As a consequence, by discarding the status message, an attacker is forced to spend $3t$ seconds in communication time between two trials.

Table 4.4: Recovery Protocol Delay

Orbit Type	Distance d from Earth (Km)	Protocol Delay (3 messages) (s)
LEO	80 (minimum)	$2.40\text{x}10^{-3}$
MEO	1,700 (minimum)	$5.10\text{x}10^{-2}$
GEO/GSO	35,700 (minimum)	1.07
Mars	78,338,750 (average)	$2.35\text{x}10^{3}$

Table 4.4 shows the protocol delay, considering the exchange of 3 messages, for different kinds of satellites [137], e.g. Low Earth Orbit (LEO), Medium Earth Orbit (MEO), and Geostationary/Geosynchronous Orbit (GEO/GSO) satellites. An estimate of the average communication time with a satellite orbiting Mars [122] is also included given the interests of many space agencies on that planet. A control center that knows the one-time secrets would spend $2.4ms$ on the challenge-response protocol while recovering a LEO satellite. If the satellite was orbiting Mars, it would spend 2350s (about 39 minutes). We are assuming that these communication delays are acceptable for control centers in most cases, and is worth doing because of the extra security that it brings to the system.

Actually, the main point in utilizing an approach based on the challenge-response protocol is to reduce secrets size and as consequently implementation area. This approach may not favor a fast recoveries in contingency situations. However, it does not necessarily mean that control centers are forced to adopt the challenge-response protocol. The secret size can be perfectly adjusted to provide the secure recovery without compromising the security of the system. The appropriate secrets size would then be defined in accordance to the security level required by the associated space mission.

In order to determine the strength of the modules against exhaustive search attacks, the delay caused by the recovery protocol must be taken into account. Because of the trusted modules' processing times are quite small in face of the protocol delay, the former can be disregarded in the determination of the total time spent in an exhaustive search attack discussed next. Additionally, the total number of trials that an attacker must perform in an exhaustive search attack is 2^b, where b is the number of bits of the one-time secret. Moreover, an attacker must wait for the end of each protocol run in order to launch another trial. Thus, the total time (in seconds) spent in an exhaustive search attack is $2^b * 3t$, where

$3t$ is the time spent exchanging 3 protocol messages as given by Table 4.4. The total time (in years) spent in an exhaustive search attack against the trusted modules are presented in Table 4.5. Notice that such an exhaustive search is completely independent of the computational power of attackers.

Table 4.5: Exhaustive Search Attack against Trusted Modules

Secret	Exhaustive Search Attack (years)			
(bits)	LEO	MEO	GEO	Mars
64	$1.40\text{x}10^{9}$	$2.99\text{x}10^{10}$	$6.27\text{x}10^{11}$	$1.38\text{x}10^{15}$
128	$2.59\text{x}10^{28}$	$5.51\text{x}10^{29}$	$1.16\text{x}10^{31}$	$2.54\text{x}10^{34}$
256	$8.82\text{x}10^{66}$	$1.87\text{x}10^{68}$	$3.94\text{x}10^{69}$	$8.64\text{x}10^{72}$

The least secure system is obtained when 64-bit secrets are used in conjunction with LEO satellites. In this case, an exhaustive search attack would take $1.40\text{x}10^{9}$ years. However, the farther the spacecraft is, the more secure the scheme becomes for the same secret size. For instance, a spacecraft implementing the same modules, but orbiting Mars, would raise the total time of an exhaustive search attack to $1.38\text{x}10^{15}$ years. If the construction constraints of the spacecraft allows for the use of more hardware area, 256-bit secrets could be used. As a result, an exhaustive search attack would become in the order of 10^{57} harder. For example, it would take $8.82\text{x}10^{66}$ for an attacker to break such a system implemented in LEO satellites, where as this number would increase to $8.64\text{x}10^{72}$ years for satellites orbiting Mars.

Man-in-the-middle attacks would require an attacker spacecraft to be positioned in such a way that it is possible to intercept RF signals that an authentic spacecraft is receiving/transmitting. Besides, jamming should be performed with precise timing. Specifically, he/she should be capable of receiving the incoming signal to acquire data (e.g. r,r',$r \oplus n$, or $r' \oplus n$), as well as jamming it in the direction of the target entity so that it is not properly received. If these conditions are met, such an attacker could discover the one-time secret for future attacks, e.g. to improperly perform a reset. Notice that, if the attacker is not in the proper orbital position, such a signal interception may not be possible. In addition, since spacecrafts may follow different orbital paths, such an attacker would have a limited time frame to perform a man-in-the-middle attack. This research considers that anti-jamming techniques are employed in the communication channels between the spacecraft and the control center, so that man-in-the-middle attacks are not feasible.

4.4 Summary

This chapters introduces a set of techniques to recover spacecrafts from attacks or severe failures caused by SEUs. Hardware implementation of the trusted modules were performed using FPGAs, which took into account various levels of security and number of possible recoveries.

It has been shown that a TRM working with 256-bit one-time secrets and allowing for 256 trusted resets, utilizes 1654 LEs and 68096 memory bits. It issues a reset signal in only $73.83ns$ when operating at 54.18MHz, while its dynamic power consumption is 20.56mW. In contrast, a TKM allowing for 256 key recoveries and using 256-bit one-time secrets, occupies 2372 LEs and 100864 memory bits. This module can recover a key in $78.85ns$ when operating at 50.73MHz, with dynamic power consumption is 40.65mW.

Considering the implementation of these modules in LEO satellites, an exhaustive search attack against them would take $8.82\text{x}10^{66}$ years. Notice that the time spent in an exhaustive search attack is completely independent of the computing power of attackers.

In summary, this research efficiently addresses the problem of bringing spacecrafts to a safe state after major failures or attacks with the subsequent restoration of their cryptographic capabilities.

Since the trusted platform relies on cryptographic hash functions to perform integrity checks, efficient fault tolerant hardware implementation of such a primitive should be researched. Besides, integrity checks and message authentication codes are both utilized to achieve secure communications. Therefore, HMAC algorithm is also investigated. Non-fault tolerant versions of SHA-2 and HMAC are presented in Chapter 5, which also includes efficient HW/SW partitioning for such primitives. Fault tolerant versions of SHA-2 and HMAC are introduced in Chapters 6 and 7, respectively.

Chapter 5

Hardware/Software Approaches for SHA-2 and HMAC

This chapter explores several levels of hardware/software (HW/SW) partitioning to implement SHA-2 and HMAC. The lowest level counts with simple operations implemented as custom instructions, whereas the highest one relies on the whole SHA-2 and HMAC algorithms implemented as peripherals. It shows that custom instructions can provide considerable computational speed up with minimum implementation area, whereas the peripherals approach can lead to a better trade-off in terms of speedup per area utilization and provide higher throughput.

5.1 Hardware/Software Partitioning

High performance systems, e.g. servers, can perform fast cryptographic processing by employing multi-processors and co-processors. However, that comes at the cost of a considerable increase in implementation area and power consumption/dissipation. Although servers may have enough resources to afford such an approach, this may not be the case for most constrained environments. Embedded systems, for instance, comprise limited computational environments, where area and power is a premium. Thus, the utilization of cryptographic mechanisms must be done in such a way that it does not cause large demands in terms of implementation area, though providing satisfactory processing performance.

Instruction set customization can be appropriate to constrained environments. In this approach, custom hardware modules are inserted into the processor datapath to acceler-

ate application-specific functions. Hence, it provides a good trade-off between custom-hardware designs and general-purpose processors. Moreover, since the module is within the processor datapath, interfacing with the hardware module is relatively simple therefore resulting in minimum communication overhead. The main drawback is that implementing complex functions as custom instructions may slow down the entire processor datapath.

In contrast, co-processors (often used as peripherals) can execute its functions independently from the main processor. As a result, this approach allows for a higher level of freedom to implement more complex operations in hardware. The downside of peripherals is that they communicate with the processor through a bus. Consequently, the lack of efficiency of the bus can impose considerable overheads while transferring data to and from peripherals.

Given the aforementioned context, this chapter focuses on the specialization of a computational platform aiming at the processing requirements of SHA-256 and HMAC. By employing a reconfigurable environment, it becomes possible to experiment with different strategies to accelerate hash and MAC computations, such as custom instructions and peripherals. Moreover, through hardware/software implementations it is possible to determine efficient ways to implement SHA-2 and HMAC, not only in terms of speedup, but also in implementation requirements.

The prototyping platform utilized is based on the NIOS2 processor [40]. This platform allow us to write programs in C and easily move parts of the application execution to the hardware modules. Hardware modules, in turn, can be implemented as custom instructions or peripherals. Thus, all communication overhead with the hardware modules are considered, allowing us to precisely determine the gains of performance achieved with each HW/SW partitioning.

The architecture specialization presented in this chapter considers five HW/SW partitioning levels for HMAC and SHA-256 algorithms. Depending upon the partitioning level, certain operations are implemented in hardware, which vary from very simple functions as custom instructions to entire algorithms as peripherals.

NIOS2 custom instructions comprise two 32-bit input ports (`dataa` and `datab`), whereas peripherals have only one (`data`) port. Both approaches receive a selector (`n`) and output one 32-bit value (`result`).

Although SHA-256 utilizes symbols as functions names, the remainder of this chapter utilizes the written form *Sum0*, *Sum1*, *Sig0* and *Sig1* when referring to functions $\sum_0$, $\sum_1$, σ_0 and σ_1, respectively. Rotations and shifts to the right are represented, respectively, by $>> n$ and $>>^* n$, where n specifies the number of bits rotated/shifted.

5.2 Custom Instructions

The first three partitioning levels explores several functions utilized in the SHA-2 algorithm, namely *ROTR*, *Sum0*, *Sum1*, *Sig0* and *Sig1*. These functions were implemented as NIOS2 customs instructions, as described in the following sub-sections.

5.2.1 Level 1: Instruction `ROTR`

The first partitioning level comes from the fact that rotations consume 39% of the SHA-256 execution. Therefore, the *ROTR* operation is implemented in hardware as a NIOS2 custom instruction. Precisely, the hardware module contains all rotations used by the SHA-256 algorithm, as shown in Figure 5.1. Each rotation is individually selected through selector `n`. A single instruction call is needed to perform a rotation. The argument is sent through port `dataa`, and the rotated argument obtained from port `result`.

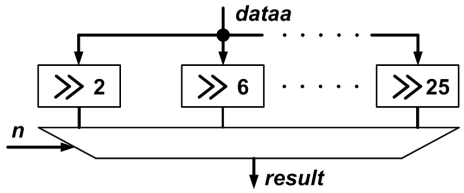
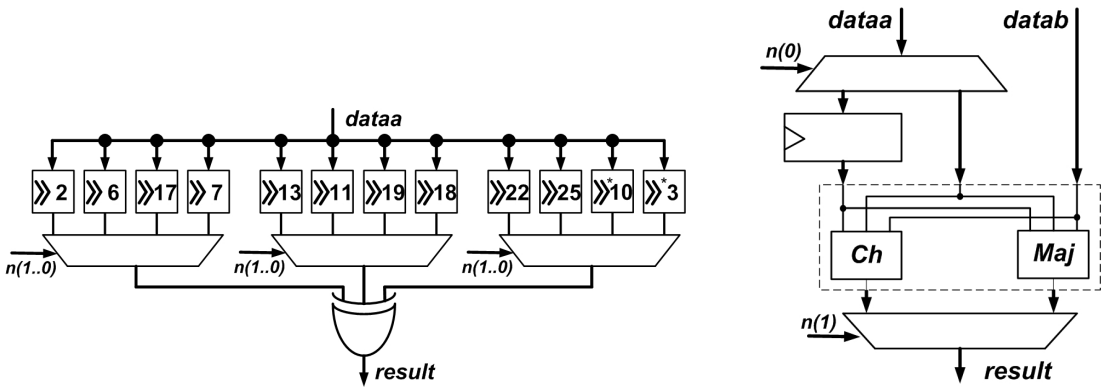


Figure 5.1: Block Diagram of Instruction `ROTR`

5.2.2 Level 2: Instructions `Sum_Sig` and `Ch_Maj`

Besides the 39% of execution time spent in the *ROTR* operation, the SHA-256 execution is divided among the functions *Sum0*, *Sum1*, *Sig0*, *Sig1*, *Ch*, *Maj*, which consume 11%, 11%, 7%, 7%, 10%, and 6% of the processing time respectively. Hence, the second partitioning level targets the implementation of those functions in hardware. The shift operation *SHR* consumes 9% of the execution time. However, the NIOS2 general-purpose instruction set already includes such an operation. Furthermore, implementation area can be saved by merging similar functions. As a result, functions *Sum0*, *Sum1*, *Sig0* and *Sig1* are merged into a single custom instruction called `Sum_Sig`, as shown in Figure 5.2 (a). Also, functions *Ch* and *Maj* are merged into `Ch_Maj` as illustrated in Figure 5.2 (b).

Instruction `Sum_Sig` receives a 32-bit operand, which can be variables a, e, W_{t-15} and W_{t-2}. Selector `n` defines the corresponding operation (*Sum0*, *Sum1*, *Sig0*, *Sig1*) to be



(a) (b)

Figure 5.2: Block Diagrams of Instructions (a) `Sum_Sig` and (b) `Ch_Maj`

performed. Instruction `Ch_Maj` operates upon three operands, which can be either a, b, c or e, f, g. More precisely, the first instruction call sends the first variable (either a or e) to be stored into an internal register. Next, the second call sends the remaining two variables (either b, c or f, g), and returns the result of the computation.

5.2.3 Level 3: Instructions `Sig` and `Sum_Ch_Maj`

The third HW/SW partitioning level aims at improving performance by reducing the number of instruction calls. For instance, the SHA-256 algorithm adds *Sig0* and *Sig1*. Hence, one instruction could perform $Sig0(W_{t-15}) + Sig1(W_{t-2})$. Figure 5.3 (a) depicts the block diagram of such an instruction denoted `Sig`. Moreover, notice that variable e is used in both *Sum1* and *Ch* functions. Similarly, a is employed in both *Sum0* and *Maj*. The hardware module illustrated in Figure 5.3 (b) results from merging the aforementioned functions, where selector `n` specifies which operation is to be performed, i.e. either $Sum1 + Ch$ or $Sum0 + Maj$. This instruction, named `Sum_Ch_Maj`, requires two calls. The first call sends the first variable (either a or e) which is stored into an internal register. Next, the second call sends the remaining two variables (either b, c or f, g), and returns a 32-bit value corresponding to the result of the computation.

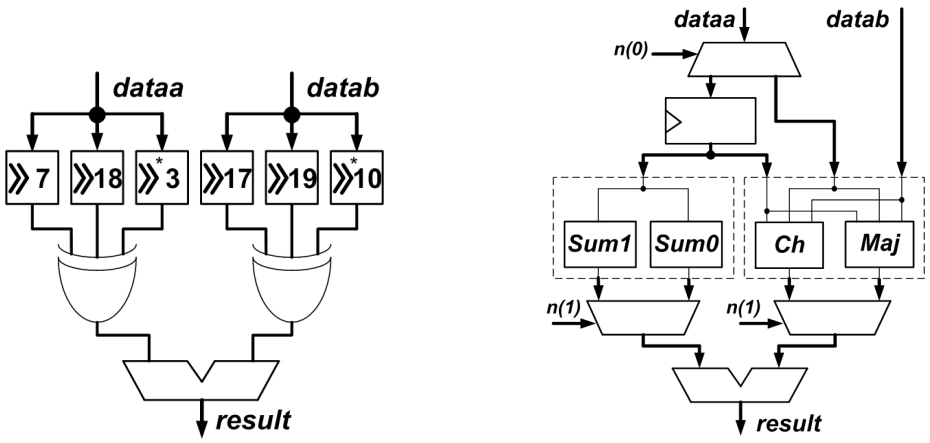


Figure 5.3: Block Diagrams of Instructions (a) `Sig` and (b) `Sum_Ch_Maj`

5.3 Peripherals

This section presents the SHA-2 and HMAC algorithms implemented as a NIOS2 peripheral. The custom instruction is not utilized in this case since this module introduces significant delay to the processor datapath, thus reducing its frequency of operation.

5.3.1 Level 4: Peripheral `SHA_256`

The fourth partitioning level relies on the utilization of the entire hash algorithm as a NIOS2 peripheral. When a message authentication code computation utilizes this peripheral, the hash is performed in hardware and the remainder of the computation is executed in software. Similarly to other SHA-2 hardware implementations cited in Chapter 3 this hardware module does not perform message padding in hardware.

Although this chapter focuses on SHA-256, similar hardware architecture can be utilized for SHA-512. Actually both SHA-256 and SHA-512 were implemented, and will be further discussed in the next chapters. Besides, the hardware module discussions presented below are applicable to both SHA-256 and SHA-512 designs.

The architectural elements of the SHA-2 implementation is shown in Figure 5.4. It basically consists of shift-registers, logical operations, D-bit adders, and a memory to store the algorithm's initialization values and constants. For SHA-256 $D = 32$, whereas $D = 64$ for SHA-512. The hardware module is divided into four main blocks: Intermediate Hash

Computation, Compressor, Message Scheduler and Constants Memory. It is important to notice that this hardware design is non-fault tolerant, and as such is denoted as `NoFT`.

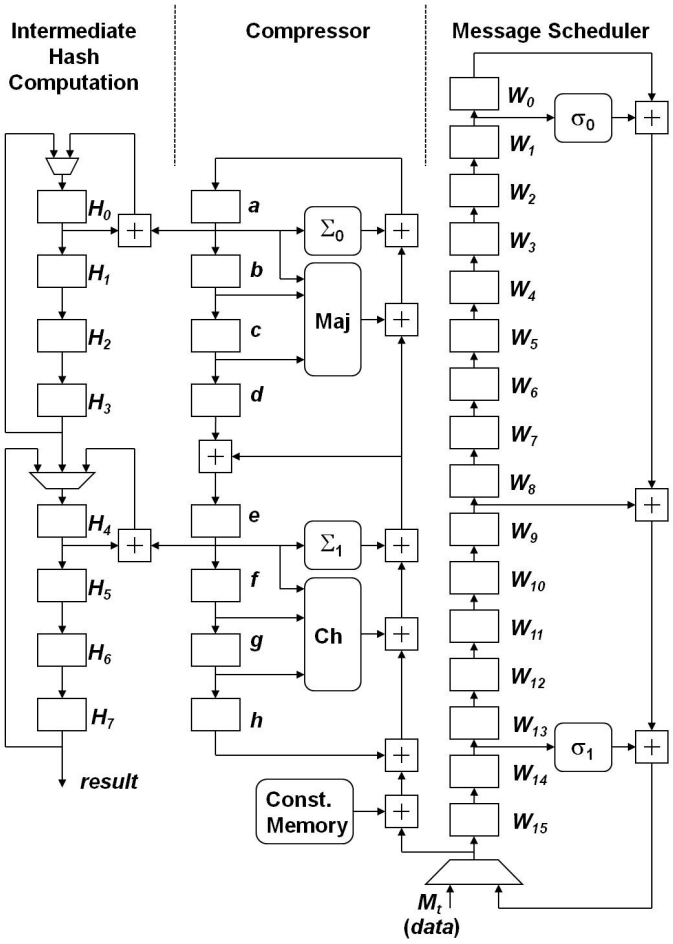


Figure 5.4: SHA-2 `NoFT` Architecture

The block of text to be processed is written into the hardware module through port `data`, while selector `n` indicates the number of the W register being written. For instance, a B-bit text requires a total of $B/32$ write calls, where B is a multiple of 512 and 1024 respectively for SHA-256 and SHA-512. The message scheduler's registers $W_0, ..., W_{15}$ are initialized by shifting in the first 16 words of the message M; this processing takes 16 iterations. Simultaneously, the constants memory provides the initialization values for the working variables $(a, ..., h)$. Initial hashes $(H_0, ..., H_7)$ are also set within this period of time. After writing the entire text into register W the execution starts automatically.

During execution, the compressor employs the current values of $a, ..., h$, as well as W_j and K_j to determine the new values of $a, ..., h$. This is performed in j iterations. In each iteration, registers $W_0, ..., W_{15}$ and $a, ..., h$ are shifted in the direction of the arrows shown in Figure 5.4.

In the end of j iterations, the intermediate hash computation must be performed. This operation could be executed in a single clock cycle, where eight additions would be performed in parallel. However, such an approach would require eight adders. Alternatively, in order to save implementation area, only two adders are utilized. This way, the computation of the intermediate hash is spread over the last 4 iterations by computing two additions per clock cycle. More precisely, the additions are performed when $t = j - 4, ..., j - 1$. For instance, in SHA-256, when $t = 60$, H_3 and H_7 are computed, when $t = 61$, H_2 and H_6 are computed, and so on.

Whenever hashing a multi-block message, the new execution cycle initiates with 16 more D-bit words being shifted into the module, and the same procedure described above is executed. After the last message block is processed, it is necessary to perform 8 read calls to the peripheral to obtain a the message digest from register H.

The total memory requirements to store the constants K_j and $H_0^{(0)}, ..., H_7^{(0)}$ is 2304 bits for SHA-256, and 5632 bits for SHA-512. In addition, registers $W_0, ..., W_{15}$, $a, ..., h$ and $H_0, ..., H_7$ utilized in SHA-256 and SHA-512 require a total of 1024 and 2048 register bits, respectively.

5.3.2 Level 5: Peripheral `HMAC_SHA_256`

The last partitioning level considers the entire HMAC as a peripheral. Again, although this section focuses on HMAC/SHA-256, an HMAC/SHA-512 hardware module was also implemented and will be further discussed in the next chapters. The HMAC module is capable of performing both HMAC and SHA-2 functions. In addition, while performing HMAC, the module is capable of processing long messages and keys of different sizes ($K \leq B$ and $K > B$). Moreover, the module allows for efficient key reutilization therefore favoring increased throughput. In order to achieve key reutilization, internal registers K_0*_Ipad_Hash* and K_0*_Opad_Hash* must be employed.

As illustrated in Figure 5.5, the proposed HMAC architecture consists of a SHA-2 core, multiplexors, logical operations, and registers. In Figure 5.5, the SHA-2 registers $W_0, ..., W_{15}$, $a, ..., h$, and $H_0, ..., H_7$ are referred to as W, $a..h$, and H respectively. Besides, HMAC utilizes three additional registers, namely K_0*_Ipad_Hash*, K_0*_Ipad_Text_Hash*, and K_0*_Opad_Hash*.

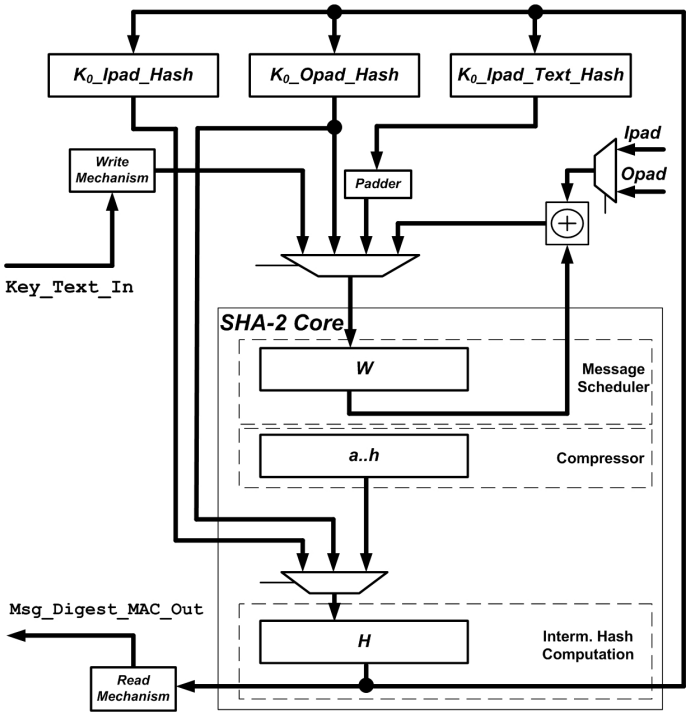


Figure 5.5: HMAC `NoFT` Architecture

Similarly to the SHA-2 module, the HMAC peripheral receives both key and text through the 32-bit port `data`, and reads the hash value and MAC from port `result`. In the actual implementation, the input port `data` is connected to the module's input port `Key_Text_In`, whereas `result` is connected to its output port `MsgDigest_MAC_Out`.

Register $K_0\_Ipad\_Hash$ stores the first half of the hash computation performed in *Step IV* of the HMAC algorithm. Since this first half of *Step IV* depends only on the current key, register $K_0\_Ipad\_Hash$ can be utilized when the same key is reutilized. The final hash of *Step IV*, whose value is text dependent, is stored in register $K_0\_Ipad\_Text\_Hash$. Thus, such a computation has to be performed every time a new key or a new text is used. In turn, register $K_0\_Opad\_Hash$ stores the first half of the hash in *Step VII*, which is also text-independent. Therefore, $K_0\_Opad\_Hash$ can also be utilized while reusing the same key.

So, key reuse is supported by utilizing registers $K_0\_Ipad\_Hash$ and $K_0\_Opad\_Hash$. Furthermore, register $K_0\_Opad\_Hash$ is also employed for temporary key storage in the beginning of the HMAC computation, which helps to decrease register requirements. The

sizes of the aforementioned registers vary with the hash algorithm used and are listed in Table 5.1.

Table 5.1: HMAC Register Names and Sizes

Register	HMAC/SHA-256 (bits)	HMAC/SHA-512 (bits)
$a..h$	256	512
H	256	512
W	512	1024
$K_0\_Ipad\_Hash$	256	512
$K_0\_Opad\_Hash$	512	1024
$K_0\_Ipad\_Text\_Hash$	256	512

The memory requirements of HMAC are based upon the memory requirements of the underlying SHA-2 algorithm. Since a D-bit constant is utilized in each of the j iterations of the SHA algorithm, a total of $D * j$ bits are necessary to store the algorithm constants. Due to efficiency reasons, HMAC module does not utilize memory to store the initialization constants of the underlying hash function.

As a result, HMAC/SHA-256 utilizes 2048 memory bits, where as HMAC/SHA-512 employs 5120 bits. It can be noticed that HMAC has lower memory requirements than SHA-2. The register requirements of the non-fault tolerant HMAC/SHA-256 and HMAC/SHA-512 hardware modules utilize, respectively, 2048 and 4096 registers.

The interfacing with the HMAC module is very similar to the SHA-2 one. More precisely, a K-bit key requires $K/32$ write calls, where K is a multiple of 512 and 1024 respectively for SHA-256 and SHA-512. Text to be processed, is sent to the module in the same way as it is done for the SHA-2 module. A B-bit text, for example, requires a total of $B/32$ write calls, where B is a multiple of 512 and 1024 respectively for SHA-256 and SHA-512.

Besides, the HMAC module allows for the user to directly interface with the built-in SHA-2 core. In the case of a hash computation, the message is written into register W through port `Key_Text_In`, similarly to the SHA-2 module. After the last word is written, the hash computation starts, which utilizes j clock cycles. Upon its completion, the message digest can be read from register H through port `MsgDigest_MAC_Out`.

The HMAC processing is divided into five stages which are denoted as `NewKeyHash`, `KeyIpadHash`, `TextHash`, `KeyOpadHash`, and `MACHash`. The stages are executed in that order, but not all stages are necessarily used. In fact, the number of stages utilized is

determined by the size of the key, the size of the text, and whether or not a key is being reused. The five aforementioned stages are represented as states in the state machine diagram represented in Figure 5.6.

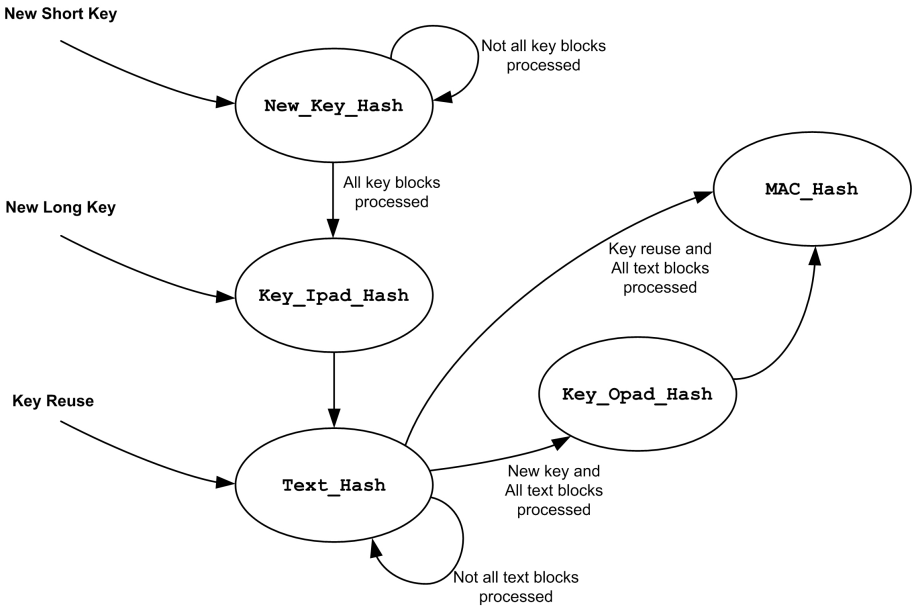


Figure 5.6: HMAC Finite State Machine Diagram

5.3.2.1 Short Keys

If a new key is used and $K = B$, then no padding is needed and $K_0 = Key$. However, if $K < B$, padding is performed to create K_0 as described in Section 2.3.2. Since the key is written into W, this register is temporarily used to perform any necessary pre-processing, such as padding.

The execution proceeds to the `KeyIpadHash` stage. Note that, since stage `KeyOpadHash` utilizes K_0 (which has been written into W) as well, its value is also temporarily stored into register $K_0\_Opad\_Hash$; otherwise it would be erased during the hash computation. Stage `KeyIpadHash` performs one hash computation, which corresponds to the execution of $Hash(K_0 \oplus Ipad)$. The result is stored into register $K_0\_Ipad\_Hash$.

Next, the message text is written into the module, followed by the execution of stage `TextHash`. Stage `TextHash` can be re-entered as long as there are message blocks to be processed. More specifically, if the message is N_m blocks long, `TextHash` is executed N_m

times. This stage is responsible for the computation of $Hash((K_0 \oplus Ipad)||Text)$, whose result is stored into register $K_0\_Ipad\_Text\_Hash$.

In the sequence, `KeyOpadHash` is performed, which corresponds to the computation of $Hash(K_0 \oplus Opad)$. Before of its execution, however, it loads K_0 (previously stored in $K_0\_Opad\_Hash$) onto W. This stage computes a single hash and stores its result back into $K_0\_Opad\_Hash$. Notice that at this point, registers $K_0\_Ipad\_Hash$ and $K_0\_Opad\_Hash$ hold the hash of K_0 with *Ipad* and *Opad*, respectively. As mentioned before, those hashes can employed in future computations utilizing the same key.

Finally, stage `MACHash` is performed. In order to accomplish that, $K_0\_Ipad\_Text\_Hash$ is padded with `0x8000...0300` to form a 512-bit value when computing HMAC/SHA-256. In the case of HMAC/SHA-512, it is padded with `8000..0600` to form a 1024-bit string. After that, the padded value is loaded onto W and the last hash computation of the HMAC algorithm begins. Upon the completion of this stage, the MAC is available in register H. The MAC can be read through port `MsgDigest_MAC_Out`.

5.3.2.2 Long Keys

In the case of utilizing a new long key ($K > B$), the computation begins in stage `NewKeyHash` in order to perform key pre-processing. Before each `NewKeyHash` execution, one block of the key must be written into W. Furthermore, if a N_k-block key is used, this stage is executed N_k times. After the `NewKeyHash` completion, K_0 becomes available in register H. At this point, K_0 is padded with zeros and loaded onto register W, thus allowing for stage `KeyIpadHash` to start. The remaining HMAC processing is identical to the one described for short keys.

5.3.2.3 Key Reuse

Finally, in the case of key reuse, the module takes advantage of previous computations to speed up the HMAC execution. The only computation needed are those dependent on the new message being processed. Since $K_0\_Ipad\_Hash$ and $K_0\_Opad\_Hash$ were previously computed, the processing starts in the `TextHash` stage. Before its execution, though, $K_0\_Ipad\_Hash$ is loaded onto H. Again, this stage is executed as long as there are message blocks to be processed. In other words, if the message is N_m blocks long, `TextHash` is executed N_m times. The result of this stage is not stored into $K_0\_Ipad\_Text\_Hash$, but loaded onto W instead. Next, $K_0\_Opad\_Hash$ is loaded onto H and stage `MACHash` starts its execution. Upon the completion of `MACHash` the MAC is made available in H.

5.4 Experimental Results

The experimental results of the proposed partitioning schemes are presented in this section, which takes into account the execution of SHA-256 and HMAC/SHA-256 algorithms.All partitioning levels are analyzed in terms of implementation area (in LEs), program size (in bytes), execution performance (in μs), throughput (in Mbps), speedup, and speedup per area ratio. In order to obtain precise comparisons between hardware and software, a standardized platform is utilized. The platform is consisted of a six-staged pipelined NIOS2 processor [40] running at 85MHz. All peripherals, in turn, operate at 70.83MHz. The program code and data are stored in a 32KB on-chip memory implemented internally to the FPGA. Furthermore, a 4KB instruction cache and a 2KB data cache are employed. Similarly to most related work, message padding is performed in software.

Performance measurements are standardized by utilizing all operations as C functions. Within each C function there is either a block of C code utilizing the NIOS2 general-purpose instruction set, or a set of calls to the custom instructions and peripherals. All programs are compiled with the GCC compiler (version 3.4.6) ported to the NIOS2 (version 9.0) processor, utilizing optimization level `-O2`. This optimization level was employed since it provided better results (small size and high performance) for most of the programs utilized in the experimental setup. Moreover, all custom NIOS2 systems were implemented on an Altera DE-2 prototyping platform [10] based on an Altera CycloneII EP2C35F672C6 [38], and synthesized using the QuartusII (version 9.0) tool [11].

5.4.1 Custom Instructions and Peripherals Implementation

As Table 5.2 shows, a basic NIOS2 system (without custom instructions and peripherals) occupies 3426 LEs. When the `ROTR` instruction is used the total system area increases 1.09 times (3734 LEs). For the second HW/SW partitioning level, the `Ch_Maj` and `Sum_Sig` instructions cause an area increase of 1.06 and 1.10 times, respectively. If both instructions are implemented in the same NIOS2 system, the total area requirements becomes 3852 LEs, thus occupying 1.12 times more area than the basic system. In the case of level 3, the maximum area increase (1.13 times) results from the inclusion of instructions `Sig` and `Sum_Ch_Maj` into the NIOS2 datapath. When the entire SHA-256 algorithm is implemented as a peripheral, the system area increases 1.62 times, therefore occupying 5547 LEs. Finally, by utilizing the HMAC/SHA-256 hardware module as a NIOS2 peripheral, 11695 LEs are needed, which represents an area increase of 3.41 times.

Besides implementation area, other two parameters to be taken into account are speedup

Table 5.2: HW/SW System Implementation Area

HW/SW Part. Level	Custom Instructions and Peripherals	System Area (LEs)	System Area Increase
-	None	3426	-
1	`ROTR`	3734	1.09
2	`Ch_Maj`	3626	1.06
	`Sum_Sig`	3760	1.10
	`Ch_Maj` & `Sum_Sig`	3852	1.12
3	`Sig`	3612	1.05
	`Sum_Ch_Maj`	3746	1.09
	`Sig` & `Sum_Ch_Maj`	3855	1.13
4	`SHA_256`	5547	1.62
5	`HMAC_SHA_256`	11695	3.41

and the ratio speedup/area. These parameters are discussed in detail in the next two subsections. A good reference model for the speedup/area ratio is the utilization of two modules in parallel to achieve a speedup of 2. Then, by doubling implementation area, its speedup/area ratio becomes 1. Hence, speedup/area lower than 1 for the proposed schemes indicate that it is not as appropriate as modular duplication. From now on, we refer to the *speedup per area increase* ratio as *speedup/area* ratio for short.

5.4.2 SHA-256 Results

Table 5.3 lists the execution results referring to the hashing of a 1024-bit random text (two 512-bit data blocks) with SHA-256. A basic NIOS2 processor executes such an operation in 136.49μs, which represents a throughput of 7.50Mbps. The utilization of the `ROTR` instruction in HW/SW partitioning level 1 accelerates the computation in 1.09 times, i.e. a hash is computed in 125.54μs. In this case there is a slight increase in the program size. Also, the area used to implement the instruction leads to a speedup per area increase ratio of 1.

Considering partitioning level 2, one can notice that the best performance is obtained by implementing only the `Sum_Sig` instruction. In this case, a SHA-256 is computed in 102.29μs, which can be translated to a throughput of 10.01Mbps. Compared to the basic NIOS2 processor, this level achieves a speedup of 1.33 therefore resulting in a speedup/area ratio of 1.22. In turn, partitioning level 3 offers slightly better performance with `Sig` and `Sum_Ch_Maj` instructions. More precisely, a SHA-256 computation is performed in 98.72μs

Table 5.3: HW/SW SHA-256 Results (1024-Bit Text)

Part. Level	Custom Instructions and Peripherals	Prog. Size (bytes)	Prog. Size Reduction (%)	Execution (μs)	Throughput (Mbps)	Speedup	Speedup / Area
-	None	5864	-	136.49	7.50	-	-
1	ROTR	5872	-0.14	125.54	8.16	1.09	1.00
2	Ch_Maj	5844	0.34	137.51	7.45	0.99	0.94
	Sum_Sig	5804	1.02	102.29	10.01	1.33	1.22
	Ch_Maj & Sum_Sig	5804	1.02	105.07	9.75	1.30	1.16
3	Sig	5784	1.36	119.66	8.56	1.14	1.08
	Sum_Ch_Maj	5800	1.09	113.42	9.03	1.20	1.10
	Sig & Sum_Ch_Maj	5760	1.77	98.72	10.37	1.38	1.23
4	SHA_256	4900	16.44	12.67	80.79	10.77	6.65
5	HMAC_SHA_256	4896	16.51	12.70	80.64	10.75	3.15

which represents a throughput of 10.37Mbps. In other words, level 3 achieves a speedup of 1.38 and a speedup/area ratio of 1.23. Besides, this HW/SW partitioning level reduces the program size in 1.77%. Notice that the area increase is similar for level 2 (`Sum_Sig`) and level 3 (`Sig` and `Sum_Ch_Maj`). Respectively, 1.10 and 1.13 times as much area as the basic NIOS2 system.

A hash is computed in 12.67μs when utilizing peripheral `SHA_256`, resulting in a throughput of 80.79Mbps and a speedup of 10.77. Although it occupies 1.62 as much area as the basic module, this peripheral provides the highest speedup/area ratio (6.65) amongst all hardware modules. Furthermore, the program size is reduced in 16.44%. Peripheral `HMAC_SHA_256` is also capable of computing hashes. Actually, the performance of `HMAC_SHA_256` is very close to the `SHA_256` one, i.e. a hash is computed in 12.70μs. The main difference is that `HMAC_SHA_256` employs 3.41 as much are as the basic system therefore reducing its speedup/area ratio to 3.15.

5.4.3 HMAC/SHA-256 Results

The execution results referring to the MAC computation along with a 512-bit key and a 512-bit text (message) are organized in Table 5.4. When utilizing the original general-purpose instruction set of a NIOS2 processor, a MAC is computed in 276.98μs therefore corresponding to a throughput of 1.85Mbps. Instruction `ROTR` provides a 1.08 speedup in the MAC computation, but leads to a low speedup/area ratio (0.99). Partitioning levels 2 and 3 result in similar speedups and speedup/area to the ones obtained in the computation of SHA-256. For instance, level 3 with `Sig` and `Sum_Ch_Maj` allow for a MAC computation to be performed in 203.58μs, which represents a speedup of 1.36 and a speedup/area ratio of 1.21.

Table 5.4: HW/SW HMAC/SHA-256 Results (512-Bit Key, 512-Bit Text)

Part. Level	Custom Instructions and Peripherals	Prog. Size (bytes)	Prog. Size Reduction (%)	Execution (μs)	Throughput (Mbps)	Speedup	Speedup / Area
-	None	7560	-	276.98	1.85	-	-
1	`ROTR`	7568	-0.11	255.56	2.00	1.08	0.99
2	`Ch_Maj`	7540	0.26	280.39	1.83	0.99	0.93
	`Sum_Sig`	7500	0.79	210.44	2.43	1.32	1.20
	`Ch_Maj` & `Sum_Sig`	7500	0.79	216.12	2.37	1.28	1.14
3	`Sig`	7480	1.06	245.03	2.09	1.13	1.07
	`Sum_Ch_Maj`	7496	0.85	232.02	2.21	1.19	1.09
	`Sig` & `Sum_Ch_Maj`	7456	1.38	203.58	2.52	1.36	1.21
4	`SHA_256`	6692	11.48	37.31	13.72	7.42	4.58
5	`HMAC_SHA_256`	5572	26.30	14.62	35.02	18.94	5.55

When peripheral `SHA_256` (level 5) is employed, a MAC is performed in 37.31μs. That means a 7.42 speedup over the basic NIOS2 processor thus leading to a 4.58 speedup/area ratio. Additionally, the program size could be reduced in 11.48%. Finally, partitioning level 6 utilizes the peripheral `HMAC_SHA_256` and computes a MAC in 14.62μs. In other words, this approach is 18.94 faster than the basic NIOS2, which results in a speedup/area ratio of 5.55. Moreover, this level also allows for a program size reduction of 26.3%.

5.5 Summary

This chapter presents several HW/SW partitioning schemes to accelerate the computation of SHA-256 and HMAC/SHA-256. A thorough analysis of each HW/SW partitioning based on a NIOS2 processor is performed so that it is possible to precisely determine the gains of performance and the costs involved with the hardware implementations. We conclude that if the main task of the processor is to execute general-purpose applications, where integrity check and MAC computations are not performed very often, the custom instruction approach can accelerate the latter computations with low area footprint. Our experimental results show that the best performance and area utilization is obtained with partitioning level 3 through the utilization of instructions `Sig` and `Sum_Ch_Maj`. In this case, the SHA-256 and HMAC/SHA-256 algorithms can be accelerated respectively in 38% and 36%, with a system area increase of only 13%.

On the other hand, peripherals are well suited to processors dedicated to hash and MAC computations. Moreover, if integrity checks are the most frequent operation to be performed, the proposed `SHA_256` peripheral can speedup the computation 10.77 times, while reducing the program binary code in 16.44%. The implementation of this peripheral increases the system area by a factor of 1.62. Peripheral `HMAC_SHA_256` can also be used for

integrity checks, but it is best suited to applications performing mainly MAC computations. Although this approach employs 3.41 times as much area as the basic NIOS2 system, it reduces the program size in 26.30% and allows for a computational speedup of 18.94 times.

These results allow for the tailoring of a computational platform of constrained environments to the processing requirements of integrity checks and message authentication codes based on SHA-256 and HMAC. To the best of our knowledge, it introduces the first implementation of a HMAC processor based on the SHA-2 family of hash functions.

Due to high speedup/area provided by the peripherals, this is the approach adopted in the remainder of this research project. Besides, the non-fault tolerant SHA-2 and HMAC hardware designs will serve as a base platform to conduct the implementation of fault tolerance schemes as described in Chapters 6 and 7.

Chapter 6

Fault Tolerant SHA-2

This chapter focuses on the proposal of efficient fault tolerant mechanisms for SHA-2 algorithms. Fault tolerant mechanisms are presented, which includes modular redundancy at the architectural and register levels as well as information redundancy based schemes employing Hamming codes. Further, FPGA implementations of the aforementioned schemes are analyzed in terms of implementation requirements, area utilization, frequency of operation, throughput and power consumption. In summary, this chapter demonstrates how information redundancy can be employed to devise efficient fault tolerant implementations of SHA-2 with lower area and power requirements than traditional methods such as TMR.

6.1 Fault Tolerance Schemes

The fault tolerant schemes investigated in this research are based on hardware redundancy and information redundancy, as well as on hybrid schemes combining both techniques. Differently from previous work that proposed only error detection, this research targets the achievement of both error detection and correction.

The main goal of the fault tolerant designs presented next is to decrease implementation area and power consumption, as well as to achieve higher resistance against SEUs than traditional approaches such as TMR. The condition for failure and resistance against SEUs of each fault tolerant scheme are discussed in detail in Chapter 8.

Three fault tolerant schemes are analyzed in this research, namely Triple Modular Redundancy (`FullTMR`), Register TMR with Hamming-encoded memory (`TMRRegs&HCMem`), and Hamming-encoded registers and memory (`HCRegs&HCMem`). Given that `FullTMR` has

been a common technique utilized in space, this scheme is utilized as a baseline for comparisons with the proposed schemes.

Each of the aforementioned schemes are applied to both SHA-256 and SHA-512. The memory and register requirements mentioned in the following sections are summarized in Tables 6.1 and 6.2.

Table 6.1: SHA-2 Memory Requirements

Scheme	SHA-256 (bits)	SHA-512 (bits)
NoFT	2304	5632
FullTMR	6912	16896
TMRRegs&HCMem	2736	6248
HCRegs&HCMem	2736	6248

Table 6.2: SHA-2 Register Requirements

Scheme	SHA-256 (bits)	SHA-512 (bits)
NoFT	1024	2048
FullTMR	3072	6144
TMRRegs&HCMem	3072	6144
HCRegs&HCMem	1216	2272

6.1.1 Full Triple Modular Redundancy

Triple modular redundancy consists in triplicating the circuit and using a voter to determine the output. In this work, three SHA-2 hardware modules are instantiated sharing the same inputs as depicted in Figure 6.1. During the data writing cycles, all three modules receive the same data in parallel. Likewise, the message digest of each module are read and sent to the voter. Based upon the three received results, the voter defines the output of the computation. The condition for failure of this scheme is described in Section 8.1.1.

During the implementation of `FullTMR`, special attention was paid to the design partitioning in order to avoid the synthesizer to merge common circuitry and registers, which would lead to misleading synthesis results.

Since FullTMR is widely used in space systems, it is taken as a reference model for further comparisons with the proposed schemes in terms of implementation area, frequency of operation, power consumption, throughput, and resistance against SEUs.

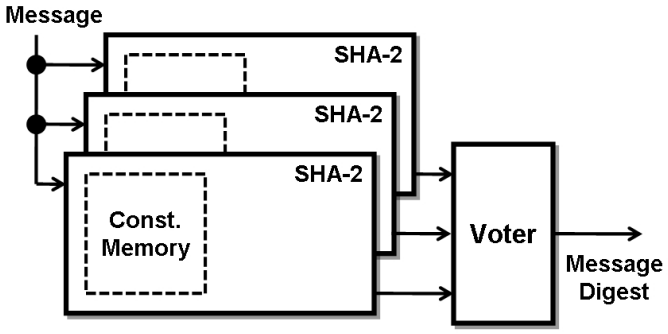


Figure 6.1: SHA-2 FullTMR Architeture

Since FullTMR uses three instances of NoFT, it employs, as expected, three times as much memory and registers as the non-fault tolerant module. Precisely, the memory requirements of FullTMR are 6912 bits for SHA-256, and 16896 bits for SHA-512. The register requirements become 3072 bits for SHA-256, and 6144 bits for SHA-512.

Due to parallelism, an advantage of FullTMR is that it does not cause a big impact on the module's frequency of operation. On the other hand, a drawback is the big area penalty imposed by replication. Thus, other schemes are proposed to reduce the memory requirements and to achieve smaller implementation area and lower power consumption.

6.1.2 TMR for Registers and Encoded Memory

Given the concern in protecting only the data being processed, an optimization to FullTMR is to move the redundancy from the modular level to the register level. Additionally, in order to scale down the number of memory bits, only one memory is utilized in this scheme. However, in order to protect the memory contents against SEUs, fault tolerance techniques must be used. Such a scheme is named TMRRegs&HCMem.

Instead of triplicating entire SHA-2 modules, TMRRegs&HCMem employs only one module, but triplicates all its registers as illustrated in Figure 6.2. Besides, each word of memory is encoded with Hamming Codes. Specifically, the memory of SHA-256 employs a (38,32) Hamming code, whereas the SHA-512 one utilizes a (71,64) Hamming code. For each

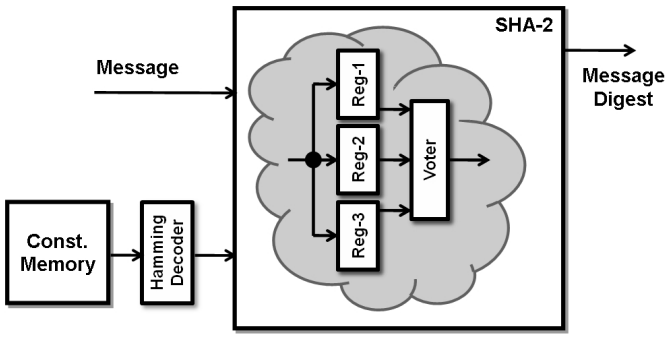


Figure 6.2: SHA-2 `TMRReg&HCMem` Architecture

memory read, a Hamming decoder detects and corrects any potential bit-flip and then sends the value to the SHA-2 module.

Even considering all the parity bits that are attached to each memory word, the memory requirements of `TMRRegs&HCMem` are decreased in comparison with `FullTMR`. As a result, `TMRRegs&HCMem` requires 2736 memory bits for SHA-256 and 6248 memory bits for SHA-512. A downside of utilizing Hamming codes for error detection and correction is the inclusion of a Hamming decoder between the memory and modules. Such decoder causes a longer critical path in the circuit therefore decreasing its frequency of operation.

The register requirements are exactly the same as in `FullTMR`, i.e. 3072 bits for SHA-256 and 6144 bits for SHA-512. Incidentally, this scheme demands one voter for each trio of registers in order to mask out registers errors. As a result, a total of 32 D-bit voters is needed ($D = 32$ for SHA-256 and $D = 64$ for SHA-512).

6.1.3 Encoded Registers and Memory

As an alternative to modular redundancy, Hamming codes can also be used to protect registers. This technique leads to the efficient fault tolerant mechanism proposed in this research denoted as `HCRegs&HCMem`. Likewise `TMRRegs&HCMem`, this scheme utilizes Hamming codes to protect the memory. The goals are to detect and correct potential bit-flips happening in SHA-2 registers and achieve efficiency by optimizing when error detection and correction is performed.

A design decision is how often error detection and correction should be performed. They can be done, for example, in every clock cycle, therefore increasing resistance against SEUs. However, that demands a higher number of Hamming decoders and encoders and reduces

the module speed. On the other hand, if they are performed less often, higher throughput and lower implementation area can be achieved at the cost of less SEU resistance.

Besides, there are multiple ways of encoding registers. For instance, the thirty two 32-bit registers of SHA-256 can be encoded individually using a (38,32) Hamming code. Using that strategy, a total of 192 parity bits would be necessary. Alternatively, all SHA-256 registers could be merged into a single 1024-bit register. By treating the registers as one 1024-bit register, a (1035,1024) Hamming code can be used and only 11 parity bits are needed, therefore reducing register requirements. The savings are even bigger in the case of SHA-512. Encoding all 64-bit registers separately using a (71,64) Hamming code, a total of 224 parity bits would be needed. In contrast, merging all registers into a single 2048-bit register and using a (2060,2048) Hamming code, only 12 parity bits are needed. Notice, however, that although larger Hamming codes reduce the number of parity bits required, they demand more complex encoders and decoders. As a consequence, the circuit critical path will be increased resulting in slower modules.

The utilization of large Hamming codes along with error detection and correction performed in every clock cycle has been evaluated in [89, 93]. It has been shown that, although such a scheme is extremely resistant against SEUs, it results in larger area utilization, higher power consumption and lower throughput when compared to TMR. Such an experiment motivates the optimization on the frequency in which error detection and correction is performed, as well as in the size of the Hamming code.

The proposed optimization was devised by analyzing how registers are utilized during the SHA-2 execution. The schedule of register utilization influences directly the way the registers are encoded and how often error detection and correction should be performed. By observing Figure 5.4, it can be noticed that in a given algorithm iteration, only registers $a, ..., h$, H_0, H_4, W_0, W_1, W_9, and W_{14} are involved in the SHA-2 functions computations. Thus, those are the only registers that need to be decoded for error detection and correction. By the same token, only a, e, H_0, H_4, and W_{15} are updated with new values. Therefore, it is only necessary to encode the data to be written into those registers. This feature leads to the architecture depicted in Figure 6.3.

In this scheme, each D-bit register is encoded individually. The main reason for encoding individual registers is to facilitate the shift of their contents, along with the associated parity bits, through the D-bit datapath without the need of encoding/decoding data. Specifically, SHA-256 encodes each of the thirty two 32-bit registers separately using a (38,32) Hamming code, demanding a total of 192 parity bits. Similarly, SHA-512 encodes each of the thirty two 64-bit registers separately using a (71,64) Hamming code, which requires a total of 224 parity bits. Hence, the total register requirements are 1216 bits for

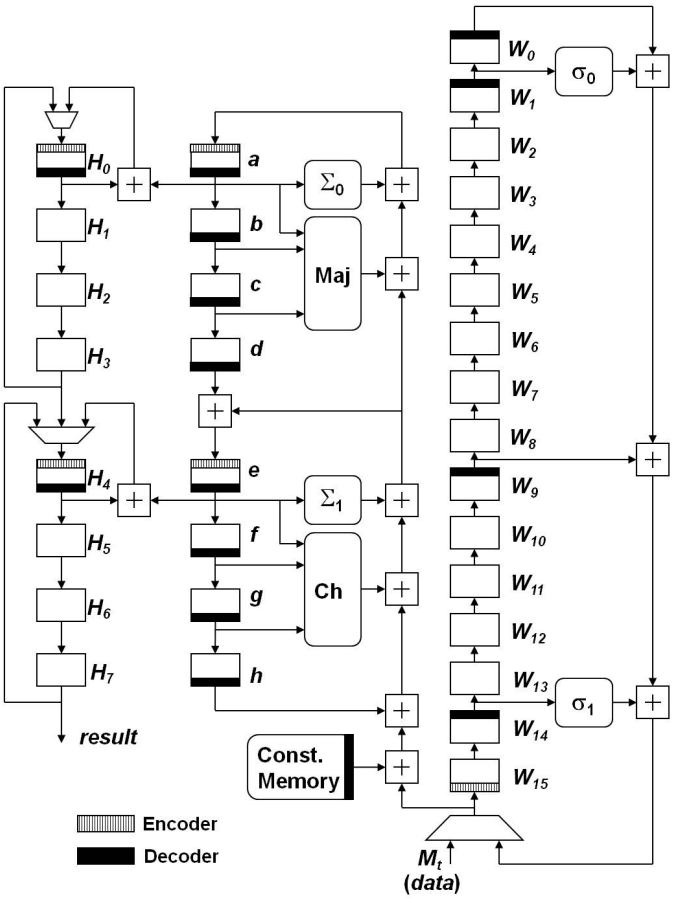


Figure 6.3: SHA-2 `HCRegs&HCMem` Architecture

SHA-256, and 2272 bits for SHA-512.

Since all registers are kept encoded, they are protected against SEUs until the next decoding operation. From Figure 6.3 it can be observed that error detection and correction is only performed when reading registers $a, ..., h$, H_0, H_4, W_0, W_1, W_9, and W_{14}. The data present in these registers have been waiting for a number of clock cycles. The period of time between encoding and decoding operations are denoted as *idle period*.

The idle period varies from 1 (for $a, ..., h$) to 60 (for H_0 and H_4) for SHA-256, or 76 in the case of SHA-512. A failure will occur if two bit-flips happen in the same data word ($a, ..., h$, H_0, H_4, W_0, W_1, W_9, W_{14}) while they are in their idle period. This case is further explained in Section 8.1.3.

HCRegs&HCMem uses an encoded memory to keep the constants protected against SEUs, exactly as in TMRRegs&HCMem. For that reason, the memory requirements of SHA-256 and SHA-512 are respectively 2736 and 6248.

6.2 Experimental Results

In order to better evaluate the advantages and disadvantages of each fault tolerant scheme, a thorough analysis was performed in terms of implementation area, throughput, frequency of operation, and power consumption. Hence, the proposed schemes were described using the VHDL hardware description language and implemented on FPGA. The tool employed in the description, synthesis, simulation and power consumption estimation of all hardware modules was QuartusII version 9.0 [11]. The target device for all implementations was an Altera SRAM FPGA CycloneII EP2C35F672C6 [38]. The synthesis parameters targeted low implementation area and low power consumption.

6.2.1 Implementation Area

Implementation area is measured in terms of the number of logic elements (LEs) used to implement a given scheme in the FPGA. According to Table 6.3, the non-fault tolerant (NoFT) SHA-256 occupies 2166 LEs. In turn, SHA-512 uses 4279 LEs.

Table 6.3: SHA-2 Implementation Area

Scheme	SHA-256 (LEs)	SHA-512 (LEs)
NoFT	2166	4279
FullTMR	6530	12892
TMRRegs&HCMem	6811	13530
HCRegs&HCMem	4471	8400

Not surprisingly, modular replication causes FullTMR to occupy approximately three times as much area as NoFT. More precisely, FullTMR SHA-256 and SHA-512 occupies, respectively, 6530 and 12892 LEs.

Although TMRRegs&HCMem does not triplicate the SHA-2 datapath, it occupies slightly more area than FullTMR. In other words, the implementation area of SHA-256 is 6811 LEs, whereas SHA-512 utilizes 13530 LEs. Such increase in implementation area is justified by

the number of registers and voters employed in the hardware design. Hence, LEs end up being exclusively for implementing registers and voters.

The HCRegs&HCMem scheme provides considerable savings in implementation area. The SHA-256 module utilizing this scheme employs only 2.06 times as much area as NoFT, i.e. 4471 LEs. Moreover, SHA-512 employs 8400 LEs, which means 1.96 as much area as NoFT. In other words, HCRegs&HCMem allows for the implementation of SHA-256 and SHA-512 to employ respectively 32% and 35% less area than FullTMR.

6.2.2 Frequency of Operation

The frequency of operation of the schemes considered in this research is measured in MHz and is obtained after the synthesis tools have performed place-and-route. Due to the high estimation precision of the QuartusII timer analyzer, the reported frequencies are very reliable and thus can be used as a reference in further stages of system design.

Table 6.4: SHA-2 Frequency of Operation

Scheme	SHA-256 (MHz)	SHA-512 (MHz)
NoFT	74.46	60.60
FullTMR	74.16	59.30
TMRRegs&HCMem	45.93	37.61
HCRegs&HCMem	44.94	36.08

As listed in Table 6.4, SHA-256 and SHA-512 NoFT run respectively at 74.46MHz and 60.60MHz. It is possible to notice that FullTMR can operate almost as fast as NoFT due to parallelism allowed by the former. For instance, SHA-256 operates less than 1% slower than FullTMR, whereas SHA-512 faces only a 2% slowdown.

Because TMRRegs&HCMem uses a Hamming decoder between the memory and the datapath, its critical path is impacted negatively. Besides a memory decoder, TMRRegs&HCMem has its frequency of operation slightly impacted by register voters placed after each trio of registers. As a consequence, it operates about 38% slower than NoFT and 37% slower than FullTMR.

Multiple encoders and decoders in HCRegs&HCMem causes a similar impact on the frequency of operation to the multiple voters in TMRRegs&HCMem. Precisely, SHA-256 and SHA-512 HCRegs&HCMem operate respectively at 44.94 and 36.08MHz. In the average, HCRegs&HCMem runs 40% slower than NoFT, and 39% slower compared to FullTMR.

6.2.3 Throughput

The throughput of SHA-2 is defined as the amount of text processed per unit of time, which is given in bits per second (bps) in this research. It can be determined from the message block size (B), module's frequency of operation (F), and the number of clock cycles (c) utilized to compute a message digest.

Specifically, the hash function throughput is defined as

$$Throughput_{SHA-2} = \frac{B * F}{c}. \tag{6.1}$$

For the purpose of computing the throughput of SHA-2 modules, the hash of one block of message was considered. The block size of SHA-256 is 512, whereas SHA-512 utilizes 1024 bits. Moreover, SHA-256 and SHA-512 take respectively 64 and 80 clock cycles to compute a message digest of a block of text.

Table 6.5: SHA-2 Throughput

Scheme	SHA-256 (Mbps)	SHA-512 (Mbps)
`NoFT`	595.68	775.68
`FullTMR`	593.28	759.04
`TMRRegs&HCMem`	367.44	481.41
`HCRegs&HCMem`	359.52	461.82

As can be noticed from Table 6.5, the frequency of operation has a strong influence on the modules' throughput. In other words, the higher the frequency, the higher the throughput. Not surprisingly, `FullTMR` has highest throughput among the fault tolerant modules. The throughput of SHA-256 using this scheme (593.28Mbps) is is about 1% lower than `NoFT` (595.68Mbps). In turn, the SHA-512 throughput is only 2% below the throughput of `NoFT`, i.e. 759.04Mbps.

Further, the throughput of `TMRRegs&HCMem` suffers a reduction of 38% when compared to `NoFT`, for both SHA-256 and SHA-512. Moreover, the throughput of `HCRegs&HCMem` is about 40% lower than the `NoFT` throughput. More precisely, when SHA-256 and SHA-512 employs this scheme, their throughput achieve, respectively, 359.52Mbps and 461.82Mbps. This corresponds to a reduction of 39% in the throughput when compared to `FullTMR`.

6.2.4 Power Consumption

Power is yet another very important parameter to be considered due to the severe constraints in power consumption and dissipation in space systems. The power consumption of each module is obtained from the PowerPlay Power Analyzer Tool [11] through the utilization of random text with the module running at the maximum frequency of operation. The power analyzer tool estimates the FPGA static, dynamic, and input/output power.

Static power is defined as the power consumed by the FPGA regardless of signal activity (no switching) and results from current leakage inside transistors. It can be interpreted as the power consumed to maintain the configuration of the FPGA. Dynamic power is defined as the power consumed due to signal activity (switching). Put differently, this is the power consumed to perform data processing. Input and output power consumption is the power consumed by the FPGA interfacing elements (I/O). Since the designs considered in this research may be implemented in a system-on-a-chip, I/O power is of minor importance and can be discarded in the analysis. The average static power consumption of SHA-256 and SHA-512 modules are, respectively, 82.6mW and 84.4mW.

Table 6.6: SHA-2 Dynamic Power Consumption

Scheme	SHA-256 (mW)	SHA-512 (mW)
`NoFT`	101.07	187.87
`FullTMR`	280.89	513.54
`TMRRegs&HCMem`	148.38	297.83
`HCRegs&HCMem`	141.85	300.63

Table 6.6 reports the dynamic power consumption of the implementations performing one hash computation at their maximum frequency of operation. For instance, SHA-256 `FullTMR` (280.89mW) consumes 2.8 times more power than `NoFT` (101.07mW). The power increase for SHA-512 is 2.7 times, as expected, since `FullTMR` triplicates the datapath.

By moving redundancy to the register level and encoding memory, the power consumption of SHA-256 utilizing `TMRRegs&HCMem` was 1.47 times higher than `NoFT`, i.e. 148.38mW. Moreover, `HCRegs&HCMem` consumes 141.85mW, which means an increase of 1.4 times in relation to `NoFT`. In the case of SHA-512, the power consumption of the last two fault tolerant schemes is 1.6 times as high as `NoFT`.

If `FullTMR` is taken as a reference, it is possible to notice that `HCRegs&HCMem` allows for a power consumption reduction of 42% and 50%, respectively, for SHA-256 and SHA-512 implementations.

6.3 Summary

This chapter presented three approaches for achieving fault tolerance in hardware implementations of SHA-2 algorithms, namely `FullTMR`, `TMRRegs&HCMem` and `HCRegs&HCMem`. The main goal was to show that information redundancy can provide more efficient FPGA implementations than the traditional modular redundancy. Experimental results were obtained through FPGA implementation, which permitted the characterization of performance and implementation requirements of each module. As a consequence it was possible to determine the advantages and disadvantages of each fault tolerance scheme.

Not surprisingly, `FullTMR` generally demands three times as much resources as the non-fault tolerant approach. Its main advantage, though, is a very high throughput provided by modular parallelism.

In an attempt to reduce memory requirements and provide higher resistance against SEUs, `TMRRegs&HCMem` moved the triple modular redundancy to the register level and encoded the memory with Hamming codes. This approach allowed for an average decrease of 62% in memory requirements. Besides, it allows for an average power reduction of 45%. However, its main drawback was the extra number of logical elements required to exclusively implemented all the triplicated registers and the associated registers.

By taking advantage of some features of the SHA-2 algorithm `HCRegs&HCMem` allowed for an efficient fault tolerance technique to be devised. This scheme resulted in the best trade-off amongst register and memory requirements, implementation area, power consumption and throughput. Furthermore, it allowed for an average saving of 62% in memory and register requirements. Moreover, in the average, its implementation employs 34% less area and consumes 46% less power than `FullTMR`. Although its average throughput is 39% lower than the ones provided by `FullTMR`, it may be high enough for most space applications.

As a result, it has been shown that `HCRegs&HCMem` can successfully provide fault tolerance in FPGA implementations of the SHA-2 family of hash functions. Besides, it favors lower power and lower implementation area, which are crucial in space applications. To the best of our knowledge, this is the first implementation of the SHA-2 family of hash functions providing both error detection and correction reported in the literature.

Next chapter will explore the aforementioned schemes to provide fault tolerance for FPGA implementation of the HMAC algorithm. Before we can recommend the proposed scheme for space applications, a resistance analysis must be carried out to determine the robustness of the fault tolerance mechanisms in face of SEUs. Such a resistance analysis is described and performed in Chapter 8.

Chapter 7

Fault Tolerant HMAC

This chapter focuses on efficient fault tolerant mechanisms for the Keyed Hash Message Authentication Code (HMAC). Fault tolerant mechanisms are presented, again comprising modular redundancy at the architectural and register levels as well as information redundancy based schemes employing Hamming codes. Besides, the discussion of experimental results takes into account implementation requirements, area utilization, frequency of operation, throughput and power consumption. In summary, this chapter shows that the proposed fault tolerance scheme based on information redundancy provides even better efficiency when applied to more complex hardware designs such as the HMAC algorithm.

7.1 Fault Tolerance Schemes

The fault tolerant schemes presented in this section are the same as the ones introduced in Chapter 6. Basically, Triple Modular Redundancy (`FullTMR`), TMR for Registers with Encoded Memory (`TMRRegs&HCMem`), and Encoded Registers and Memory (`HCRegs&HCMem`).

Again, the main goal of the fault tolerant designs is to minimize implementation area and power consumption, and maximize resistance against SEUs. Detailed analysis of the condition for failure and resistance against SEUs of each scheme can be found in Chapter 8.

The aforementioned fault tolerant schemes were applied to HMAC utilizing SHA-256 and SHA-512 as the underlying hash functions. The memory and register requirements mentioned in the following sections are listed in Tables 7.1 and 7.2, respectively.

Table 7.1: HMAC Memory Requirements

Scheme	HMAC/SHA-256 (bits)	HMAC/SHA-512 (bits)
`NoFT`	2048	5120
`FullTMR`	6144	15360
`TMRRegs&HCMem`	2432	5680
`HCRegs&HCMem`	2432	5680

Table 7.2: HMAC Register Requirements

Scheme	HMAC/SHA-256 (bits)	HMAC/SHA-512 (bits)
`NoFT`	2048	4096
`FullTMR`	6144	12288
`TMRRegs&HCMem`	6144	12288
`HCRegs&HCMem`	2432	4544

7.1.1 Triple Modular Redundancy

The TMR design of HMAC consists of three instances of the `NoFT` module, which share the same input (`Data_In`), as illustrated in Figure 7.1. Their outputs are sent to a D-bit voter so that the result (`Result_Out`) can be determined ($D = 32$ in HMAC/SHA-256 and $D = 64$ in HMAC/SHA-512). This design is named `FullTMR` for short.

Special attention was paid to the design partitioning in order to guarantee that the synthesis tool would not merge subcomponents of the three instances of `NoFT`, which would consequently lead to misleading results. Precisely, the hardware design was partitioned into four blocks: one partition for each `NoFT` module and one partition for the voter.

Given that `FullTMR` utilizes three `NoFT` modules along with a voter, it triplicates the register and memory requirements of the `NoFT` module. Similarly to SHA-2, the `FullTMR` implementation of HMAC is used as a reference model for the comparisons performed in the next sections.

Specifically, as listed in Table 7.1, the memory requirements for HMAC/SHA-256 are 6144 bits, and for HMAC/SHA-512 they are 15360 bits. Moreover, as reported in Table 7.2, the register requirements of `FullTMR` are 6144 and 12288 bits, respectively, for HMAC/SHA-256 and HMAC/SHA-512.

Although there are replication penalties in terms of implementation area, `FullTMR`

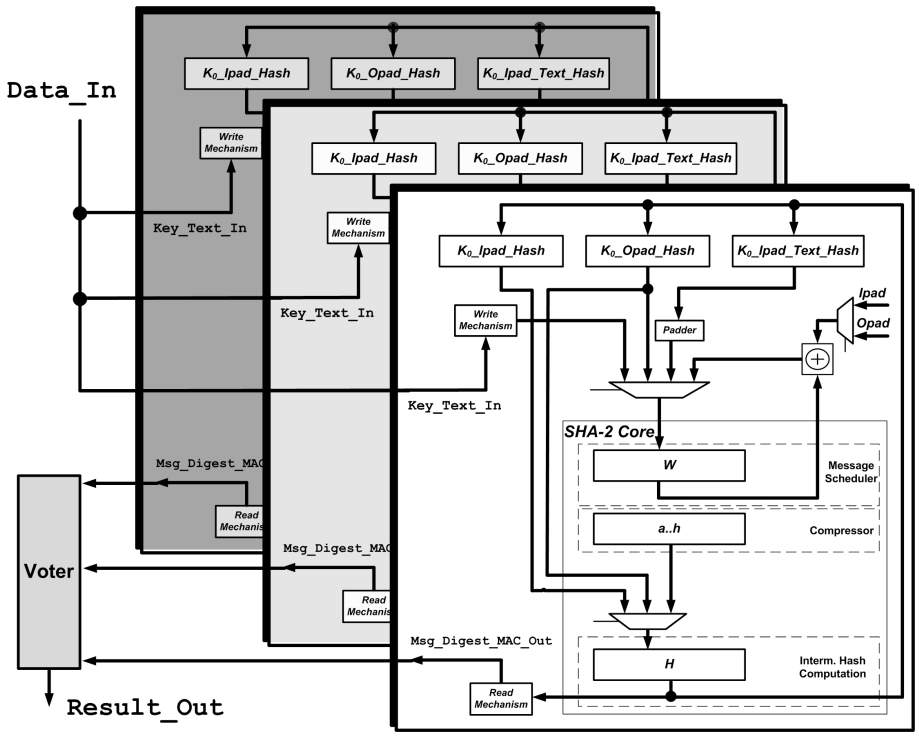


Figure 7.1: HMAC `FullTMR` Architecture

allows for three modules to run in parallel therefore causing little impact on the modules' performance. Notice, however, that error detection and correction is performed by the voter at the end of the HMAC computation. The conditions for failure of this scheme is described in Section 8.1.1.

7.1.2 TMR for Registers and Encoded Memory

This scheme moves redundancy to the register level instead of keeping it at the modular level. The underlying hash module is exactly the same as the one presented in Section 6.1.2. This approach is denoted as `TMRRegs&HCMem`.

As shown in Figure 7.2, this scheme triplicates all registers and utilizes voters (striped boxes) to perform error detection and correction. Precisely, a voter is used for each trio of registers, whose bit width varies according to the bit width of the associated registers, as listed in Table 7.2.

Since all HMAC registers, besides all the underlying SHA-2 registers, are triplicated, it is easy to notice that this scheme results in the same register requirements as `FullTMR`. For instance, HMAC/SHA-256 and HMAC/SHA-512 employs, respectively, 6144 and 12288 bits. Furthermore, error detection and correction is executed in each clock cycle therefore being performed more often than in `FullTMR`. The conditions for failure of this scheme is discussed in Section 8.1.2.

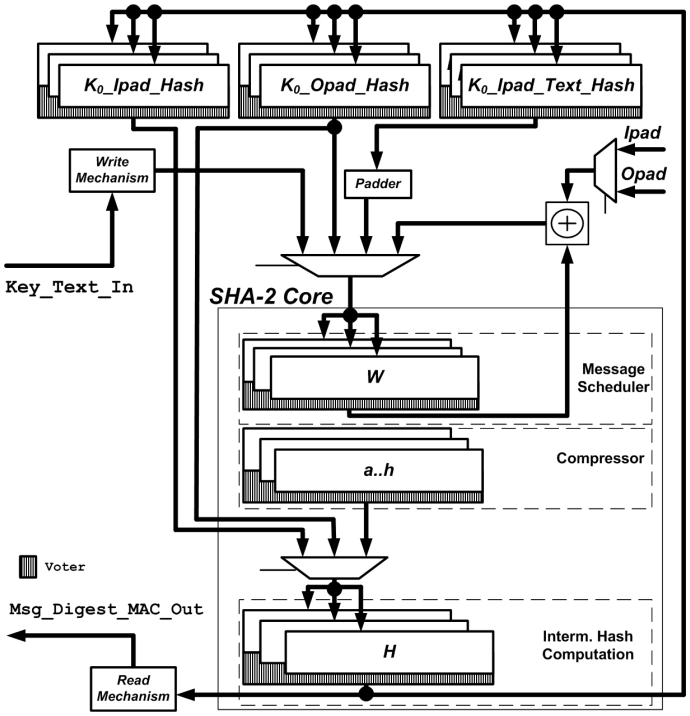


Figure 7.2: HMAC `TMRRegs&HCMem` Architecture

The memory requirements are reduced by encoding the constants memory of the underlying SHA-256 and SHA-512 with Hamming codes, exactly as presented in Section 6.1.2. Essentially, the constant memories of the underlying SHA-256 and SHA-512 utilize, respectively, $(38, 32)$ and $(71, 64)$ Hamming codes. Due to the utilization of Hamming codes, `TMRRegs&HCMem` demands considerable less memory bits than `FullTMR`. Specifically, `TMRRegs&HCMem` utilizes respectively 2432 and 5680 bits.

7.1.3 Encoded Registers and Memory

Differently from modular redundancy, this approach employs information redundancy as a technique to protect registers and memory against SEUs. The strategy utilizes Hamming codes to keep HMAC and SHA-2 registers encoded all the time. This scheme is denoted as `HCRegs&HCMem` for short.

The number of encoding and decoding operations are optimized so that it is possible to reduce the number of Hamming encoders/decoders while keeping a high level of protection against SEUs. Such a level of optimization is possible by analyzing when specific registers are used in the HMAC and SHA-2 execution, similarly to the description in Section 6.1.3. However, the utilization of multiple encoders and decoders in the HMAC/SHA-2 datapath increases the circuit's critical path. Consequently, lower frequencies of operation are expected for `HCRegs&HCMem` compared to the ones achieved in `FullTMR` and `TMRRegs&HCMem`.

The type of Hamming code to be used for SHA-2 registers depends directly on how HMAC interfaces the underlying hash function. It is true that the encoding large amounts of data result in big savings in terms of the number of Hamming parity bits. For example, 1024-bit registers could be encoded with $(1035, 1024)$ Hamming code, which would use only 11 additional bits to store the parity. Similarly, 2048-bit registers would require only 12 additional bits if the $(2060, 2048)$ Hamming code is employed. Notice, however, that the number of parity bits is only one factor in the definition of the Hamming code.

A more important aspect to be considered is how tightly coupled HMAC and SHA-2 become given a specific Hamming code. In other words, the size of the code has to be defined in such a way so that it facilitates the movement of data between HMAC to SHA-2 registers. As a result, the Hamming code plays a fundamental role in achieving efficiency in terms of area, processing time, which ends up resulting in lower power consumption and higher throughput.

As explained in Section 5.3.2, HMAC constantly interfaces registers H and W in read and write operations. For instance, stage `KeyOpadHash` loads a pre-processed K_0 (stored in register $K_0\_Opad\_Hash$) onto W. Stage `TextHash` loads $K_0\_Ipad\_Hash$ onto H. Stage `MACHash` loads $K_0\_Ipad\_Text\_Hash$ onto W. If a key is reused, stage `TextHash` loads $K_0\_Opad\_Hash$ onto H. After a hash computation is completed, register H is loaded onto HMAC registers $K_0\_Ipad\_Hash$, $K_0\_Opad\_Hash$ and $K_0\_Ipad\_Text\_Hash$.

In order to allow for the utilization of larger Hamming codes, registers $a..h$, H and W could be merged together. However, loading values onto H or W would require decoding of the entire merged register. Similarly, to transfer a value from those registers to one of the HMAC registers would require an encoding operation. That approach would cause

inefficient 32 and 64-bit rotations, which are often utilized by the respective SHA-256 and SHA-512 algorithms. Such a scheme would demand for the merged register to be decoded and re-encoded prior and after each rotation. On the other hand, the encoding of small portions of data such as 4, 8 and 16 bits facilitates the aforementioned rotations and register transfers, but ends up utilizing too many Hamming parity bits.

The utilization of $(38, 32)$ codes for HMAC/SHA-256, and $(71, 64)$ for HMAC/SHA-512 results in a better trade-off amongst parity bits, area efficiency, and minimization of encoding/decoding. Precisely, HMAC/SHA-256 encodes its registers in 32-bit blocks therefore demanding a total of 192 parity bits. Likewise, HMAC/SHA-512 encodes 64-bit blocks, which ends up utilizing a total of 224 parity bits.

Hence, the main reasons to choose codes $(38, 32)$ and $(71, 64)$ were: 1) Encoding small blocks of data causes encoding and decoding to be fast and efficient; 2) It makes it possible to re-utilize the optimized fault tolerant SHA-2 design therefore minimizing the number of encoding and decoding; 3) It facilitates the frequent D-bit rotations of SHA-2 without decoding and re-encoding; 4) Facilitates complete or partial loads of HMAC registers and associated parity bits onto SHA-2 registers therefore dispensing decoding/re-encoding.

The total register requirements for HMAC/SHA-256 and HMAC/SHA-512 are respectively 2432 and 4544 bits, as shown in Table 7.2. Figure 7.3 show all HMAC and SHA-2 registers with their associated parity bits (striped boxes).

Another optimization of this fault tolerant scheme refers to the computation of parity bits of W after performing the `xor` with `Ipad` and `Opad` in the `KeyIpadHash` and `KeyOpadHash` stages. The binary Hamming parities of a 32-bit word of `Ipad` and `Opad` used in HMAC/SHA-256 are respectively `110111` and `011001`, where the rightmost digit corresponds to the least significant bit. In the case of HMAC/SHA-512, the respective parities are `0110011` and `0011111`. Since `Ipad`, `Opad` and the message pad are constants, it is possible to avoid the computation of the Hamming parity bits of W by properly adjusting them. This is accomplished by performing the `xor` of the parity bits of W with the ones of `Ipad` and `Opad`.

The parity bits of W have also to be adjusted when the message is padded at the beginning of the `MACHash` stage. In this stage, the contents of $K_0\_Ipad\_Text\_Hash$ is loaded onto the first half of W, while the second half of receives the message pad, which is `0x8000...0300` for HMAC/SHA-256 and `0x8000...0600` for HMAC/SHA-512. Since the registers are encoded in D-bit blocks the parity bits of $K_0\_Ipad\_Text\_Hash$ are copied into the first half of the parity bits of W. Given that the message pad is constant, its parity bits are also constant and therefore loaded onto the second half of the parity bits of W. The binary value of the message pad parity bits are `100110000000...000011` and

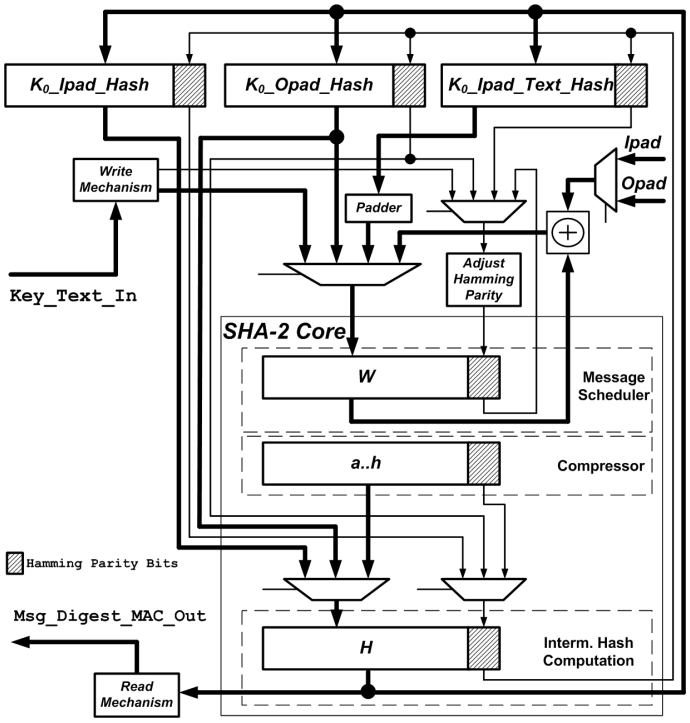


Figure 7.3: HMAC `HCRegs&HCMem` Architecture

`10001110000000...0000001` respectively for HMAC/SHA-256 and HMAC/SHA-512. The HMAC stages of this scheme utilize one additional clock cycle after each hash computation to update the parity of register H.

In order to corrupt the computation of HMAC, two bit-flips must happen in the same encoded register within the time frame that no error detection and correction is performed. During the HMAC processing the maximum idle period is faced by the K_0_$Opad$_$Hash$. This register is written in the end of the `NewKeyHash` stage and read in the beginning of `KeyOpadHash` stage, which also loads its value onto W. Hence, the total idle period corresponds to the duration of the execution of stages `KeyIpadHash` and `TextHash`, plus 7 clock cycles referring to the transfer time between W_8 and W_1 where it is finally decoded. Thus, the condition for failure is to have two bit-flips in the same register within that time frame. Further discussion on the conditions for failure is provided in Section 8.1.3.

Similarly to `TMRRegs&HCMem`, `HCRegs&HCMem` uses an encoded memory to keep the SHA-2 constants protected against SEUs. The memory requirements of HMAC/SHA-256 and HMAC/SHA-512 are respectively 2432 and 5680 bits.

7.2 Experimental Results

The HMAC hardware modules are analyzed identically to SHA-2. Again, implementation area, frequency of operation, throughput and power consumption are the parameters considered the comparisons and evaluations throughout this section.

It was fundamental to maintain the same experimental setup (e.g. FPGA device, tools version and settings) to make it possible to perform fair comparisons between HMAC and SHA-2 hardware implementations. The tool utilized for the hardware description, synthesis, simulation and power estimation of all hardware modules was QuartusII version 9.0 [11]. The synthesis and simulation parameters are exactly the same as the ones utilized for SHA-2. The target device was also maintained, i.e. an Altera SRAM FPGA CycloneII EP2C35F672C6 [38].

7.2.1 Implementation Area

The FPGA implementation of HMAC/SHA-256 `NoFT` employs 4266 LEs, as listed in Table 7.3. HMAC/SHA-512, in turn, occupies 9265 LEs. As expected, `FullTMR` scheme utilizes approximately three times as much area as `NoFT`. Precisely HMAC/SHA-256 and HMAC/SHA-512 `FullTMR` demands respectively 12899 and 28744 LEs.

Table 7.3: HMAC Implementation Area

Scheme	HMAC/SHA-256 (LEs)	HMAC/SHA-512 (LEs)
`NoFT`	4266	9265
`FullTMR`	12899	28744
`TMRRegs&HCMem`	14006	27838
`HCRegs&HCMem`	6840	13477

The area utilization of HMAC/SHA-256 `TMRRegs&HCMem` (14006 LEs) is 3.3 times as high as the area of the `NoFT`, which is justified by the large number of logical elements to implement all triplicated registers and associated voters. Surprisingly, the synthesis of HMAC/SHA-512 utilizing such fault tolerant scheme resulted in slightly lower area (27838 LEs) than `FullTMR`, however still 3 times bigger than `NoFT`.

Once again, `HCRegs&HCMem` leads to considerable savings in implementation area. Precisely, HMAC/SHA-256 employs 6840 LEs, which represents an area increase of 1.6 times

over NoFT. The results is slightly better in the case of HMAC/SHA-512, whose implementation area (13477 LEs) is only 1.5 times as big as NoFT. Remarkably, the utilization of this scheme allows for implementations of HMAC/SHA-256 and HMAC/SHA-512 to be respectively 35% and 53% smaller than the ones employing FullTMR.

7.2.2 Frequency of Operation

The frequencies of operation of the HMAC hardware modules are listed in Table 7.4. The NoFT modules of HMAC/SHA-256 and HMAC/SHA-512 are capable of operating in 67.13MHz and 55.87MHz, respectively. By analyzing Table 7.4, it is possible to realize that the second fastest approach is FullTMR. Since FullTMR utilizes all three modules running in parallel, it suffers a minor impact on the frequency of operation. As a result, this fault tolerant scheme allows for HMAC to operate, in the average, only 4% slower NoFT.

Table 7.4: HMAC Frequency of Operation

Scheme	HMAC/SHA-256 (MHz)	HMAC/SHA-512 (MHz)
NoFT	67.13	55.87
FullTMR	64.31	54.14
TMRRegs&HCMem	44.19	37.42
HCRegs&HCMem	44.69	35.99

The utilization of a Hamming decoder between the memory and the module datapath has a big impact in the frequency of operation of TMRRegs&HCMem. When employing this fault tolerance scheme, HMAC runs in the average 34% slower than NoFT. Moreover, this means that TMRRegs&HCMem is about 31% slower than FullTMR.

Finally, the utilization of HCRegs&HCMem as a fault tolerant mechanism results in frequencies of operation similar to TMRRegs&HCMem. For example, HMAC/SHA-256 runs at 44.69MHz whereas HMAC/SHA-512 performs at 35.99MHz. In the average, this represents a frequency slowdown of 35% compared to NoFT, and 32% compared to FullTMR.

7.2.3 Throughput

Given that the HMAC processor designed in this research is based on SHA-2, its throughput ends up being a function of the hash throughput, as defined by Equation 6.1. As

traditionally considered in the literature, the cryptographic key is not taken as an input while computing throughput of HMAC. In other words, although some time is necessary to process the key, only the message is considered as input for the HMAC execution.

As described in Section 5.3.2, if a new key is used, 3 hashes are always computed (`KeyIpadHash`, `KeyOpadHash`, and `MACHash`). If a new N_k-block key is used ($K > B$), `NewKeyHash` is performed N_k times to pre-process the key. If ($K \leq B$), `NewKeyHash` is not performed, i.e. $N_k = 0$. Also, it the message is N_m blocks long, `TextHash` is executed N_m times. In sum, when a new key is used, the computation of $3 + N_m + N_k$ hash functions are needed. As a consequence, the HMAC throughput using a new key is given by

$$Throughput_{NewKey} = \frac{N_m}{3 + N_m + N_k} Throughput_{SHA-2}. \tag{7.1}$$

If the key is reused, `TextHash` is performed N_m times and `MACHash` once. As a result, the HMAC throughput is determined by

$$Throughput_{KeyReuse} = \frac{N_m}{1 + N_m} Throughput_{SHA-2}. \tag{7.2}$$

The number of clock cycles taken by the SHA-2 within HMAC modules vary with the type of fault tolerant scheme and hash function utilized. In all cases (except `HCRegs&HCMem`) a hash computation over a 1-block message takes 65 clock cycles with HMAC/SHA-256 and 81 clock cycles with HMAC/SHA-512. Notice that the number of clock cycles refer to one hash computation. In the case of `HCRegs&HCMem`, an additional clock cycle is used to compute the parity of H in the end of each hash computation. Consequently this scheme takes 66 and 82 clock cycles respectively for HMAC/SHA-256 and HMAC/SHA-512. While computing a MAC, the total number of clock cycles depends on the key and message sizes. Table 7.5 lists the number of clock cycles taken in different scenarios of hash and MAC computations.

In order to determine the throughput of each hash and MAC operation, the maximum frequency of operation of the hardware modules, as listed in Table 7.4, where utilized. Equation 6.1 was employed to determine the throughput of the hash operation, whereas Equation 7.1 was utilized to compute the HMAC throughput. Key reuse can be computed through Equation 7.2, but is not reported in this section. The HMAC modules are employed in two scenarios: 1) Hash computation of a 1-block message; 2) MAC computation of a 1-block message using a 1-block key.

Table 7.6 lists the throughput of the HMAC modules operating in the two aforementioned scenarios. When computing a hash, HMAC/SHA-256 `NoFT` reaches a throughput

Table 7.5: HMAC/SHA-2 Clock Cycles for Hash and MAC Computations

Scheme	Hash 1-Block Message		MAC, 1-Block New Key 1-Block Message	
	HMAC/ SHA-256	HMAC/ SHA-512	HMAC/ SHA-256	HMAC/ SHA-512
	(Clock Cycles)			
`NoFT`	65	81	260	324
`FullTMR`	65	81	260	324
`TMRRegs&HCMem`	65	81	260	324
`HCRegs&HCMem`	66	82	264	328

of 528.78Mbps. If a MAC of a 1-block message is computed utilizing a new key, the maximum throughput drops to a fourth of the hash one, i.e. 132.19Mbps. In the case of HMAC/SHA-512 `NoFT`, the throughput a hash computation is 706.31Mbps. Although HMAC/SHA-512 has a lower frequency of operation compared to HMAC/SHA-256, the throughput of the former is increased by the message size (1024 bits for HMAC/SHA-512 against 512 bits for HMAC/SHA-256).

In general, HMAC/SHA-512 follows the same throughput trends of HMAC/SHA-256. In other words, the computation of a MAC over a 1-block text with a new key reduces the hash throughput to one quarter. That trend also holds for all fault tolerant schemes considered in this work. The following discussions and comparisons focus on the second scenario, i.e. on the computation of a MAC using 1-block text and a 1-block key.

Table 7.6: HMAC Throughput

Scheme	Hash 1-Block Message		MAC, 1-Block New Key 1-Block Message	
	HMAC/ SHA-256	HMAC/ SHA-512	HMAC/ SHA-256	HMAC/ SHA-512
	(Mbps)			
`NoFT`	528.78	706.31	132.19	176.58
`FullTMR`	506.56	684.44	126.64	171.11
`TMRRegs&HCMem`	348.08	473.06	87.02	118.27
`HCRegs&HCMem`	346.09	449.44	86.67	112.36

In the average, the throughput of `FullTMR` is about 4% lower than `NoFT`. For instance, HMAC/SHA-256 and HMAC/SHA-512 `FullTMR` achieves 126.64 and 171.11Mbps. In con-

trast, when utilizing TMRRegs&HCMem the throughput drops to 87.02Mbps and 118.27Mbps respectively for HMAC/SHA-256 and HMAC/SHA-512. In other words, the throughput of TMRRegs&HCMem is in the average 34% lower than NoFT and 31% lower than FullTMR.

The throughput of HCRegs&HCMem is the lowest one amongst all HMAC hardware modules. HMAC/SHA-512 reaches 112.36Mbps, while HMAC/SHA-256 achieves 86.67Mbps. In the average, this represents a 35% reduction compared to NoFT, and a 33% reduction compared to FullTMR.

7.2.4 Power Consumption

Power consumption is also discussed using the two scenarios introduced in the previous section. The static power consumption of HMAC/SHA-256 and HMAC/SHA-512 are respectively 83mw and 85.3mW. As reported in Table 7.7, the dynamic power consumption of HMAC/SHA-256 NoFT while computing a hash is 119.39mW, whereas HMAC/SHA-512 demands 232.54mW. If a MAC computation is considered, HMAC/SHA-256 demands 137.09mW while HMAC/SHA-512 consumes 255.42mW. In general, HMAC/SHA-256 and HMAC/SHA-512 modules consume, respectively, 14% and 10% more power for computing a MAC than they do for a hash.

Table 7.7: HMAC Dynamic Power Consumption

Scheme	Hash 1-Block Message		MAC, 1-Block New Key 1-Block Message	
	HMAC/ SHA-256	HMAC/ SHA-512	HMAC/ SHA-256	HMAC/ SHA-512
	(mW)			
NoFT	119.39	232.54	137.09	255.42
FullTMR	333.12	659.78	378.48	730.00
TMRRegs&HCMem	186.99	312.59	203.50	336.03
HCRegs&HCMem	166.59	310.10	191.21	340.99

Again, the following discussions and comparisons focus on the computation of a MAC of a 1-block text utilizing a 1-block key. The datapath triplication in FullTMR causes this scheme to consume, in the average, 2.8 times more power than NoFT. The highest power consumption (730mW) is due to HMAC/SHA-512 employing FullTMR.

The power consumption of HMAC/SHA-256 TMRRegs&HCMem is only 1.5 times as high as the NoFT one. Moreover, HMAC/SHA-512 consumes even less power, i.e. 1.3 as much

as its non-fault tolerant version. In the case of HCRegs&HCMem, HMAC/SHA-256 consumes 191.21mW to perform a MAC computation, which represents 1.4 as much power as NoFT. Furthermore, if this fault tolerant scheme is applied to HMAC/SHA-512 the power consumption (340.99mW) is only 1.3 as high as the NoFT one.

Significant savings are obtained with TMRRegs&HCMem and HCRegs&HCMem. Surprisingly, by utilizing the TMRRegs&HCMem scheme, HMAC/SHA-256 and HMAC/SHA-512 consume respectively 46% and 54% less power than FullTMR. In the case employing HCRegs&HCMem, the power reductions in relation to FullTMR are 50% and 53% respectively for HMAC/SHA-256 and HMAC/SHA-512.

7.3 Summary

This chapter presented application of the three fault tolerance approaches first introduced in Chapter 6 into the HMAC based on SHA-2 hash functions. Again, experimental results are obtained through FPGA implementations from where a comprehensive analysis of the modules characteristics has been derived.

The traditional FullTMR approach utilizes three times as much registers and memory, implementation area and consumes twice as much power as the non-fault tolerant design. TMRRegs&HCMem, in turn, leads to similar implementation requirements, i.e. 3.1 times as much area and 3 times as many registers as in NoFT. Although the throughput provided by TMRRegs&HCMem is 31% lower than FullTMR, the power consumption of the former scheme is in the average 50% lower than the latter.

By applying the HCRegs&HCMem approach to HMAC, the reduction of memory and register requirements were exactly the same as in SHA-2, i.e. 62% in the average. However, this scheme allowed for the achievement of even bigger savings in implementation area and power consumption in the case of HMAC. Precisely, the implementation area of HCRegs&HCMem is only 50% of the FullTMR one. Moreover, the former approach consumes in the average 52% less power than the latter. In order to achieve such efficiency with HCRegs&HCMem, it was necessary to compromise its throughput, which is 33% lower than FullTMR. However, the throughput obtained with such a scheme may be sufficient for most space applications.

These results show that the utilization of information redundancy to protect registers and memory elements can lead to efficient fault tolerant implementations of HMAC. To the best of our knowledge, this is the first proposal of fault tolerant techniques for the keyed-hash message authentication code in the literature. We have shown that the proposed

scheme become even more effective than `FullTMR` as the algorithm complexity increases. In spite of that, it is necessary to conduct an SEU resistance analysis before finally arguing that `HCRegs&HCMem` can successfully replace `FullTMR` in the implementation of HMAC in space applications.

A resistance analysis for evaluating the fault tolerance mechanisms is proposed and discussed in Chapter 8. Besides, next chapter also presents a quantitative approach to compare different fault tolerant schemes.

Chapter 8

SEU Resistance Analysis

Resistance against SEUs is another crucial factor to be considered when designing digital hardware for space systems. This chapter introduces a method based upon probability of failure for evaluating the resistance against SEUs of the fault tolerant schemes introduced in Chapters 6 and 7. Besides, a comprehensive analysis of memory and register resistance is performed for all fault tolerant designs of SHA-2 and HMAC. In addition, a normalized analysis is also performed which takes TMR as a reference model. In summary, this chapter shows that the proposed scheme based on information redundancy provides not only implementation efficiency, but also higher resistance against SEUs than the traditional TMR approach.

8.1 Probability of Failure

The resistance analysis proposed in this chapter is based on the probability of failure of memory and registers. Such an analysis takes into account the total amount of resources available in a given device, implementation requirements and frequency of operation of each design as well as an hypothetical bit-flip rate. Since the bit-flip rate would vary with the environment in which the hardware device is operating, it is taken as literal variables that can be easily substituted for precise results. Moreover, the proposed probability of failure allows for different fault tolerant schemes to be compared. The strategy relies on the comparison with a reference model and is denoted as normalized analysis. The following sections take `FullTMR` as a reference model since this has been a traditional fault tolerance scheme employed in space applications.

It is important to mention that this probability of failure analysis was first proposed in [93] and [91]. However, the frequency of operation of the hardware modules were not taken into account in those publications. As a consequence, the accuracy of the resistance analysis ends up being compromised. That happens because faster modules, such as `FullTMR`, usually execute their computations in less time than other approaches, i.e. they leave their registers and memories exposed to bit-flips for a shorter period time. Although high performance is the main advantage of employing modular redundancy, it can be argued, though, that the robustness of each individual approach should be determined by employing the same frequency of operation. In any case, by considering different frequencies of operation, `FullTMR`, the reference model, will be situated in its best operational scenario. Thus, if other schemes can be shown to be more resistant than `FullTMR` in this context, the same conclusion can be drawn when all modules operate at the same frequency.

It is assumed that only one fault tolerant design is implemented per device and that the device resources are utilized uniformly. It is further assumed that the hardware devices are subject to the same bit-flip rate. Also, as just explained, all modules run at their maximum frequency of operation. The following literals are employed in the probability of failure equations:

Therefore, the following terms are defined:

M : Total memory resources in bits,

R : Total number of registers in bits,

m : Used memory resources in bits,

r : Used registers in bits,

ω : Bit-flip rate per memory bit per second,

ε : Bit-flip rate per register bit per second,

F : Frequency of operation (Hz),

n_S : Period of time in which bit-flips may occur expressed in number of clock cycles for a given scheme S,

l : Total number of memory bits employed by an individual encoded word,

t : Total number of register bits utilized by a trio of registers,

i : Number of clock cycles without performing error detection and correction,

g : Number of bits employed by an individual encoded register,

$P(X_1)$: Probability of the first bit-flip in X, where X can be either memory or register elements,

$P(X_2)$: Probability of the second bit-flip in X,

$P(F_M)$: Probability of a memory failure,

$P(F_R)$: Probability of a register failure.

These definitions and conditions for failure allows for the determination of the individual probability of failure of memory and register elements for each fault tolerant scheme.

In order to compute the probability of bit-flips ($P(X_1)$ and $P(X_2)$) the bit-flip rate ω (ε) is expected to cause no more than a single memory (register) bit-flip within the time frame n_S/F. Therefore, conditions $(n_S\omega)/F \leq 1$ and $(n_S\varepsilon)/F \leq 1$ must be obeyed while computing the probability of failure of each fault tolerant scheme.

8.1.1 Triple Modular Redundancy

In this scheme a failure will occur if two bit-flips happen in two different modules during the processing. In this case the voter would receive three different values from the modules and therefore would not be able to determine the correct output. Notice, however, that if two bit-flips happen in the exact same module, the voter will receive only one corrupted computation result and will be able to determine the correct output. It doesn't matter if the bit-flips happened in two registers, two memory elements, or even one bit-flip in a register and another one in a memory element.

In order to determine the memory resistance, it is assumed that the triplicated memory utilizes m bits out of the total M bits available in the device; that the device is subject to a bit-flip rate ω; that error detection and correction is performed after n clock cycles; and that the module runs at its maximum frequency of operation F. Figure 8.1(a) represents such assumptions.

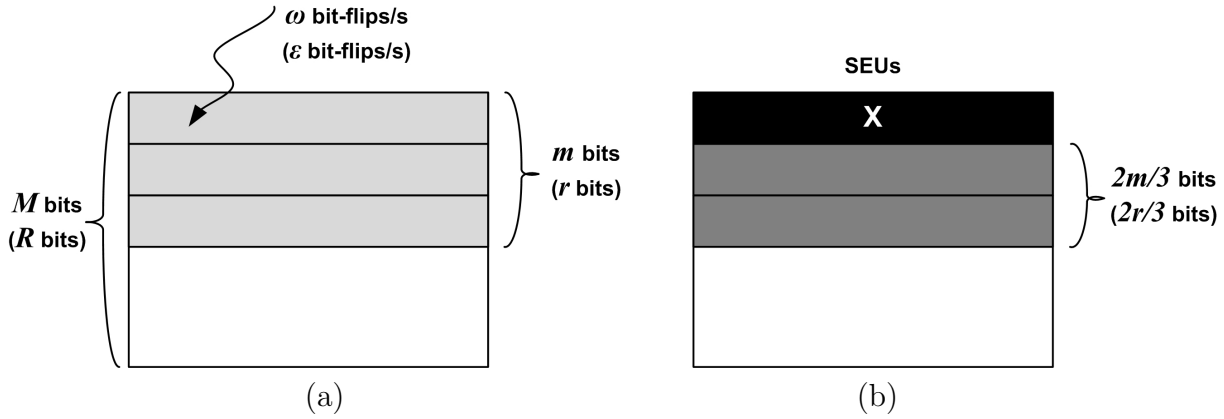


Figure 8.1: SEU Resistance Analysis for `FullTMR`'s Memory (and Registers)

Considering that the memory modules themselves do not rely on any fault tolerant mechanism, a single bit-flip in the contents of one of these memories causes the computation

of the corresponding `NoFT` module to become corrupted. The probability of the first bit-flip to happen in m memory bits during the n clock cycles taken by the processing is given by $P(M_1) = mn\omega/(MF)$. The first bit-flip is represented by an X in Figure 8.1(b). The computation of the corresponding module (represented in black) is compromised.

A second bit-flip in any other location of the other two memories ($2m/3$ bits), as represented in dark gray in Figure 8.1(b), compromises the entire `FullTMR` scheme. The probability of failure of the second bit-flip is given by $P(M_2) = 2mn\omega/(3MF)$.

Therefore, the probability of a memory failure is given by

$$P(F_M) = 2m^2n^2\omega^2/(3M^2F^2). \tag{8.1}$$

The same condition for failure applies to the registers; in order to have a register failure in `FullTMR` it is necessary to have a first bit-flip in a register of one of the modules and a second bit-flip in any of the registers of the other two modules. The condition for failure assumes that r register bits are used out of the total R, that n clock cycles are spent before error detection and correction is performed, along with a bit-flip rate ε.

The probability of the first bit-flip to happen in the r register bits is determined by $P(R_1) = rn\varepsilon/(RF)$, whereas the probability of the second register bit-flip (in the remaining $2r/3$ bits) is given by $P(R_2) = 2rn\varepsilon/(3RF)$.

As a result, the probability of register failure is

$$P(F_R) = 2r^2n^2\varepsilon^2/(3R^2F^2). \tag{8.2}$$

8.1.2 TMR for Registers and Encoded Memory

In this scheme, fault tolerance is obtained by encoding each memory word with Hamming codes. The main advantage of this approach is that each encoded word tolerates up to one bit-flip without leading to a failure. The condition for memory failure in `TMRRegs&HCMem` is to have two bit-flips happening in the same encoded word, within the n clock cycles required by the computation. It is assumed an usage of m memory bits out of the total M available together with a bit-flip rate of ω, as shown in Figure 8.2(a).

Therefore, the probability of having the first bit-flip in any of the memory bits, as represented by an X in Figure 8.2(b), is $P(M_1) = mn\omega/(MF)$.

Given that each memory position is encoded individually, the condition for failure is to have a second bit-flip exactly in the same encoded word that received the first bit-flip.

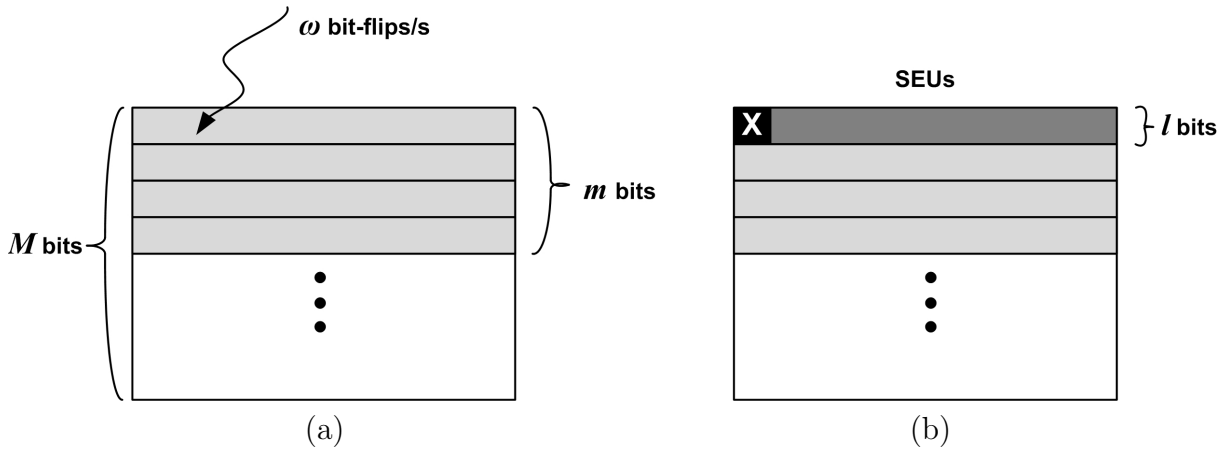


Figure 8.2: SEU Resistance Analysis for `TMRRegs&HCMem`'s Memory

Assume that each encoded memory location utilizes l bits and that the first bit-flip, represented by the black box in Figure 8.2(b), happened in one of these l bits. Then, a failure will happen if a second bit-flip happens in the remaining $(l-1)$ bits, as illustrated in dark gray in Figure 8.2(b). Thus, the probability of a second bit-flip is $P(M_2) = (l-1)n\omega/(MF)$.

The final probability of having a memory failure in `TMRRegs&HCMem` is given by

$$P(F_M) = m(l-1)n^2\omega^2/(M^2F^2). \tag{8.3}$$

Due to the employment of TMR at the register level, bit-flips happening in each trio of registers are masked out by the voters in every clock cycle. Hence, in order to cause a register failure in `TMRRegs&HCMem`, two bit-flips must occur, in the same clock cycle, in two different registers pertaining to a trio. In this case, the corresponding voter would not be able to decide for the correct result.

Consider a total register usage of r bits out of R bits available and a register bit-flip rate of ε, as illustrated in Figure 8.3(a). The probability of the first register bit-flip, that can happen in any of the r register bits, is given by $P(R_1) = r\varepsilon/(RF)$.

Assuming that a trio of registers occupies t bits and that one of these registers suffered a bit-flip, as represented by a black box with an X in Figure 8.3(b). Thus, the probability of having a second register corrupted in the trio ($2t/3$ bits), as represented in dark gray in Figure 8.3(b), is determined by $P(R_2) = 2t\varepsilon/(3RF)$.

Therefore, the final probability of register failure is

$$P(F_R) = 2tr\varepsilon^2/(3R^2F^2). \tag{8.4}$$

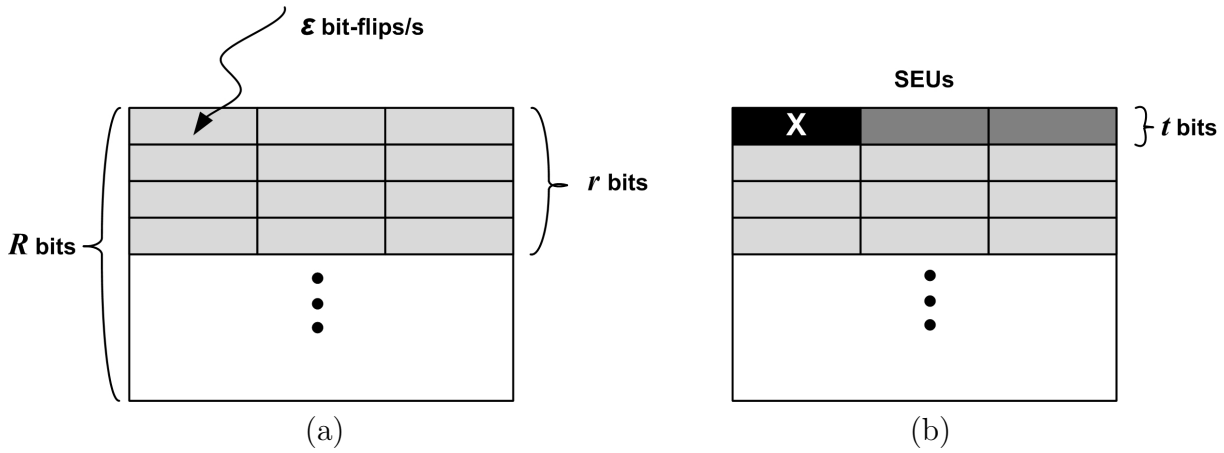


Figure 8.3: SEU Resistance Analysis for `TMRRegs&HCMem`'s Registers

8.1.3 Encoded Registers and Memory

The `HCRegs&HCMem` scheme employs Hamming codes to protect the memory, similarly to `TMRRegs&HCMem`. Therefore, the probability of having a memory failure is given by Equation 8.3, i.e. $P(F_M) = 2m^2n^2\omega^2/(3M^2F^2)$.

The register resistance analysis presented next is based on one HMAC computation with a 1-block key and 1-block text. The condition for a register failure is to have two bit-flips in the same encoded register while they are in their idle period, i.e. within the i clock cycles where no error detection and correction is not performed.

During the HMAC processing the maximum idle period is faced by the $K_0\_Opad\_Hash$. This register is written in the end of the `NewKeyHash` stage and read in the beginning of `KeyOpadHash` stage, which also loads its value onto W. Thus, the condition for failure is to have two bit-flips in the same register within that time frame.

Assume that the encoded registers employ r bits out of the R bits available and consider a bit-flip rate of ε, as illustrated in Figure 8.4(a). Then, the probability of the first bit-flip in these r register bits is $P(R_1) = ri\varepsilon/(RF)$.

Consider that a bit-flip occurred in one of the g bits of the encoded register, as shown by an X in Figure 8.4(b). A failure will occur if a second bit is flipped in the same encoded register during i clock cycles. Then, as represented in dark gray in Figure 8.4(b), the second bit-flip must happen in one of the $(g-1)$ remaining bits. Therefore, the probability of failure of the second bit-flip is given by $P(R_2) = r(g-1)i\varepsilon/(RF)$.

Thus, the probability of register failure in HCRegs&HCMem is given by

$$P(F_R) = r(g-1)\varepsilon^2 i^2/(R^2F^2). \tag{8.5}$$

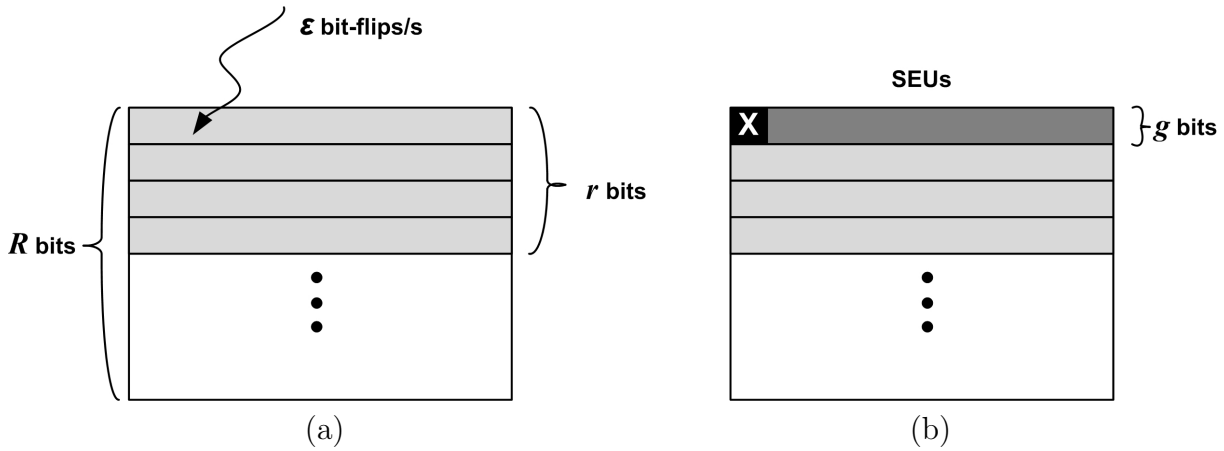


Figure 8.4: SEU Resistance Analysis for HCRegs&HCMem's Registers

8.2 Quantitative Analysis

The probability of failure of each scheme is computed by taking into account device-specific parameters such as total number of registers (R) and memory (M) bits. The device utilized for the hardware implementations is an Altera CycloneII EP2C35F672C6 FPGA, whose M and R parameters are respectively 33216 and 483840 bits. Memory (m) and register (r) requirements of each fault tolerant scheme applied to SHA-2 are shown in Tables 6.1 and 6.2, respectively. In turn, HMAC memory and register requirements are reported in Tables 7.1 and 7.2, respectively. Parameters employed in the equations defining the probability of failure are listed in Table 8.1. Bit-flip rates ε and ω are treated as literals since they are common for all designs.

8.2.1 Memory Resistance Results

The probability of memory failure can be determined by utilizing the parameters listed in Tables 6.1, 7.1 and 8.1 along with Equations 8.1 and 8.3. Table 8.2 shows the probability of memory failure, in terms of ω^2, for all fault tolerant schemes considered in this work.

Table 8.1: Resistance Analysis Parameters

Parameter	SHA-		HMAC/SHA-	
	256	512	256	512
M	33216			
R	483840			
$n_{\texttt{FullTMR}}$ (cycles)	64	80	260	324
$n_{\texttt{TMRRegs\&HCMem}}$ (cycles)	64	80	260	324
$n_{\texttt{HCRegs\&HCMem}}$ (cycles)	64	80	264	328
l (bits)	38	71	38	71
t (bits)	96	192	96	192
i (cycles)	60	76	139	171
g (bits)	38	71	38	71

Table 8.2: Probability of Memory Failure

Scheme	SHA-		HMAC/SHA-	
	256	512	256	512
	($\text{x}10^{-3}\omega^2$)			
`FullTMR`	0.10133	1.47960	1.75710	24.06252
`TMRRegs&HCMem`	0.00084	0.00845	0.01331	0.12733
`HCRegs&HCMem`	0.00088	0.00919	0.01341	0.14107

As can be observed in Table 8.2, the probability of failure of `FullTMR` SHA-256 and SHA-512 are respectively $0.10133\text{x}10^{-3}\omega^2$ and $1.4796\text{x}10^{-3}\omega^2$. In the case of applying `FullTMR` to HMAC/SHA-256 and HMAC/SHA-512 their probabilities of memory failure become respectively $1.7571\text{x}10^{-3}\omega^2$ and $24.06252\text{x}10^{-3}\omega^2$.

In turn, `TMRRegs&HCMem` and `HCRegs&HCMem` provide similar probabilities of memory failure since both use exactly the same strategy to encode the memory. For instance, `HCRegs&HCMem` SHA-256 and SHA-512 have probabilities of memory failure $0.0088\text{x}10^{-3}\omega^2$ and $0.00919\text{x}10^{-3}\omega^2$, respectively. In turn, HMAC/SHA-256 and HMAC/SHA-512 have probabilities of memory failure $0.01341\text{x}10^{-3}\omega^2$ and $0.14107\text{x}10^{-3}\omega^2$, respectively.

In general, the probability of memory failure increases as the modules get more complex. Put differently, the more memory that is employed, the higher the number of bits that can be flipped. By the same token, the slower the frequency of operation the larger the processing time frame and so the more likely to have bits flipped during operation. Thus, one interesting aspect to be analyzed is how the probability of memory failure of each fault tolerance scheme behaves as more memory bits and more clock cycles are employed by

the hardware modules. This can be accomplished through a cross-comparison among the probability of memory failures shown in Table 8.2.

The first cross-comparison evaluates the probability of memory failure of SHA-256 modules against the SHA-512 ones, as well as HMAC/SHA-256 against HMAC/SHA-512. When employing `FullTMR` the probability of memory failure of modules utilizing the SHA-512 hash function is in the average 14.2 times higher than the ones utilizing SHA-256. In the case of `HCRegs&HCMem`, the increase of probability of memory failure is 10.5 times, in the average.

The second scenario compares SHA-256 with HMAC/SHA-256, as well as SHA-512 with HMAC/SHA-512. When using `FullTMR`, HMAC modules have an average probability of memory failure 16.8 times higher than that of SHA-2 modules. By the same token, the average probability of memory failure of HMAC utilizing `HCRegs&HCMem` is 15.3 times higher than SHA-2.

Since TMR is widely used in space applications, it can be taken as a reference model to perform a normalized analysis of the probability of memory failure. As Table 8.3 shows, the utilization of Hamming codes allowed for the memory to become more than 110 times as resistant as `FullTMR`. For instance, the utilization of `HCRegs&HCMem` to implement SHA-512 made its memory 161 times more resistant than the memory of `FullTMR`. Moreover, the former fault tolerance scheme allowed for the memory of HMAC/SHA-512 to become 171 timer more resistant than the `FullTMR`. Even better results are obtained with `TMRRegs&HCMem`, however, such an approach leads to high utilization of implementation area, as discussed in Chapters 6 and 7.

Table 8.3: Normalized Memory Resistance Comparison

Scheme	SHA-256	SHA-512	HMAC/SHA-256	HMAC/SHA-512
	(Times more resistant)			
`FullTMR`	1	1	1	1
`TMRRegs&HCMem`	121	175	132	189
`HCRegs&HCMem`	116	161	131	171

8.2.2 Register Resistance Results

The probability of register failure is also determined by utilizing parameters listed in Tables 6.2, 7.2 and 8.1, together with Equations 8.2, 8.4 and 8.5. The probability of register failure, in terms of ε^2, for all fault tolerant schemes are shown in Table 8.4.

Table 8.4: Probability of Register Failure

Scheme	SHA-256	SHA-512	HMAC/SHA-256	HMAC/SHA-512
	($x10^{-3}\varepsilon^2$)			
`FullTMR`	4.24695	41.51328	372.82598	3267.60837
`TMRRegs&HCMem`	0.00008	0.00050	0.00018	0.00102
`HCRegs&HCMem`	0.37898	3.16266	0.81187	6.66147

The utilization of `FullTMR` leads to probabilities of register failure $4.24695x10^{-3}\varepsilon^2$ and $41.51328x10^{-3}\varepsilon^2$, respectively, for SHA-256 and SHA-512. As the register usage and processing time increase, so does the probability of register failure. For instance, the probabilities of register failure in HMAC/SHA-256 and HMAC/SHA-512 are respectively $372.82598x10^{-3}\varepsilon^2$ and $3267.60837x10^{-3}\varepsilon^2$.

If `TMRRegs&HCMem` is utilized, the probability of register failure becomes extremely low for both SHA-2 and HMAC/SHA-2 modules. Such a low probability of register failure results from error detection and correction being performed in every clock cycle. However, as discussed before, implementation area is the main drawback of this approach.

Besides benefits in implementation area and power consumption, the proposed approach `HCRegs&HCMem` also provides higher register resistance. The probability of register failure of the SHA-256 module is $0.37898x10^{-3}\varepsilon^2$, whereas it is $3.16266x10^{-3}\varepsilon^2$ for SHA-512. With regard to HMAC/SHA-256 and HMAC/SHA-512 modules, their respective probabilities of register failure are $0.81187x10^{-3}\varepsilon^2$ and $6.66147x10^{-3}\varepsilon^2$.

Again, it is interesting to realize a cross-comparison between SHA-256 and SHA-512 modules, as well as between HMAC/SHA-256 and HMAC/SHA-512. When `FullTMR` is employed, the probability of register failure of modules using SHA-512 is in the average 9.3 times higher than the ones utilizing SHA-256. In turn, such a probability of register failure is increased, in the average, 8.3 times when `HCRegs&HCMem` is utilized.

Surprisingly, if SHA-256 is compared with HMAC/SHA-256 an average increase of 83.3 times in the probability of register failure is observed when employing `FullTMR`. The same results are obtained when comparing SHA-512 with HMAC/SHA-512. In the case of `HCRegs&HCMem`, the probability of register failure is increased, in the average, only 2.1 times. This interesting result shows the robustness of the proposed fault tolerant scheme. In other words, `FullTMR` faces a higher increase in the probability of failure as the register requirements increases, whereas `HCRegs&HCMem` keeps such a probability of register failure at much lower levels in the same conditions.

A normalized register resistance comparison is listed in Table 8.5. `TMRRegs&HCMem` leads to extremely high resistance against SEUs, which can achieve $3x10^6$ in the case of HMAC/SHA-512. This robustness comes at the price of higher requirements in implementation area.

On the other hand, `HCRegs&HCMem` allows for reduced implementation requirements and increased resistance against SEUs compared to the previous approaches. For instance, when SHA-256 and SHA-512 employs this approach their registers became 11 and 13 times more resistant than the registers of `FullTMR`. Even better results are obtained as the register requirements increase. Precisely, `HCRegs&HCMem` makes the registers of HMAC/SHA-256 and HMAC/SHA-512 respectively 459 and 491 times more resistant than when employing `FullTMR`.

Table 8.5: Normalized Register Resistance Comparison

Scheme	SHA-		HMAC/SHA-	
	256	512	256	512
	(Times more resistant)			
`FullTMR`	1	1	1	1
`TMRRegs&HCMem`	$50x10^3$	$82x10^3$	$2x10^6$	$3x10^6$
`HCRegs&HCMem`	11	13	459	491

It is important to mention that `HCRegs&HCMem` allows for a more uniform execution of error detection and correction, causing the approach to be less dependent on the register size and number of algorithm iterations, as shown in the previous analyses. Besides, this approach represents a first proposal of a mechanisms for both error detection and correction for SHA-2 and HMAC.

8.3 Summary

This chapter introduces a resistance analysis against SEUs based on the probability of failure of memory and registers. By taking into account device properties and implementation results, it was possible to perform a quantitative analysis for each of the fault tolerant techniques proposed in the previous chapters. For the sake of comparison, `FullTMR` was taken as a reference model therefore allowing for a normalized resistance analysis to be performed as well as for a direct comparison with the traditional scheme used in space.

The benefits of using an encoded memory is twofold. Besides utilizing less memory bits, the memory became more resistant against SEUs. For instance, the least resistance

using of encoded memories is due HCRegs&HCMem SHA-256, but in this case the memory is 116 times more resistant than that of FullTMR. The highest resistance is obtained with TMRRegs&HCMem HMAC/SHA-512, whose memory is 189 times more resistant than FullTMR.

Regarding register resistance, TMRRegs&HCMem offers the highest level of protection against SEUs. However, the downside of this scheme is its demand in terms of implementation area, which is higher than FullTMR. On the other hand, HCRegs&HCMem offers a high protection against SEUs, with reduced area and power requirements in comparison with FullTMR. For example, HCRegs&HCMem allows for the register resistance of SHA-512 to be 13 times higher than the one obtained through FullTMR. The register resistance provided by HCRegs&HCMem becomes even higher as the module complexity and number of register increase. For instance, this scheme allows for the register of HMAC/SHA-512 to become 491 times more resistant than FullTMR.

These results show that information redundancy not only leads to efficiency in implementation requirements, but also offers higher levels of protection to memory and registers elements. Therefore, HCRegs&HCMem is an appropriate fault tolerant mechanism to successfully replace FullTMR in the implementation of SHA-2 and HMAC/SHA-2 cryptographic primitives in FPGAs.

Next chapter provides the reader with a discussion on the fault tolerant schemes for hardware implementation of SHA-2 and HMAC. Moreover, it includes a comprehensive comparison in terms of implementation requirements of the proposed mechanisms with TMR which has been traditionally used in space.

Chapter 9

Discussions, Conclusions and Future Work

This chapter presents discussions on the trusted platform for spacecraft recovery as well as on the fault tolerant schemes for SHA-2 and HMAC proposed in this research. Comparisons with related work is performed whenever possible, although not many similar work could be found in the literature that could be directly compared to this research. Besides, conclusions are presented which aims at answering the open questions on space security that were posed in the introduction.

9.1 Trusted Platform

Traditional trusted platform modules [43, 20, 45, 110] have relied on a set of cryptographic primitives that are well suited for general-purpose desktops and laptops. Even so, there are multiple issues in the employment of such TPMs in space. First, these TPMs should be made fault tolerant in order to cope with the harsh environment found in space. Although TPMs can provide a wide variety of cryptographic functions, it cannot be reutilized as a recovery platform as currently needed in space applications.

The trusted platform proposed in this research has been designed with fault tolerance in mind, so that it can be employed in space systems. A set of fault tolerance mechanisms have been devised, such as encoding of memory elements and data being processed within the platform, as well as an automated secret synchronization mechanism. Furthermore,

the proposed platform was devised to be able to recover the computational platform in case of major failures and attacks.

Furthermore, the standard TPM has employed RSA to perform encryption and signatures, as well as on SHA-1 as the cryptographic hash algorithm. It should be noticed that the scientific community working with space security has no formal position on the utilization of asymmetric primitives in space due to the computational requirements and key management infrastructure demanded by asymmetric primitives. Symmetric encryption primitives, in turn, have been considered by the Consultative Committee for Space Data Systems (CCSDS) to be adopted as a space standard [28, 29]. Besides RSA, standard TPM specification also relies on SHA-1 algorithms as cryptographic hash functions. However, space missions usually face a long time frame from the time of their conception to the moment they are actually deployed to operate in space. Such a time frame is usually a couple of years and may even reach decades. Thus, the utilization of older and potentially weaker cryptographic algorithms seems not to be the best choice for mission concepts targeting years, or even decades, of operations in space. Specially if it is taken into consideration the difficulty in updating a computation platform that is already deployed into space. Therefore, whenever possible, newer and more secure algorithms should be utilized if such a platform is intended to operate in space for long periods of time. Hence, SHA-2 algorithms are considered in this research, as well as message authentication codes based upon HMAC/SHA-2.

In [143] a scheme is proposed to generate keys based on features and properties directly associated with the actual spacecraft. Unfortunately this work does not provide any details on how such an on-board key generation should be performed. Additionally, it assumes that an attacker never has access to such pieces of information. On the contrary, if an attacker gains information about the spacecraft subsystems and features, he/she could break the entire system by generating identical cryptographic keys by him/herself.

The trusted recovery proposed in this research not only shows how a trusted reset can be performed, but also how to securely recover cryptographic keys stored within the trusted modules. Moreover, even if an attacker has direct access to the trusted modules, it cannot perform any better than conducting an exhaustive search. However, such an attack has been shown to be infeasible. The proposed approach achieves both security and implementation simplicity so that it can be efficiently employed in emergency commanding during contingency situations.

The approach utilized in the system recovery are based on the challenge-response protocol [106, 167] commonly used ground-based network settings. Moreover, this protocol associated with the one-time secrets have similar functionality to one-time-passwords [106].

However, differently from the challenge-response protocol proposed in [167], the protocol utilized in this research does not rely on HMAC, but on `xor` functions. From another perspective, the random number generated by the spacecraft, along with the `xor` function and the one-time secret sent by the control center resembles an one-time pad system. The difference from ground-based systems is that the proposed mechanism relies on an on-board trusted platform capable of storing one-time secrets that cannot be discovered by unauthentic users.

The challenge-response protocol together with communications delays between the spacecraft and the ground station are used in favor of providing higher security to the scheme. On the one hand, a valid control center has to execute the protocol only once, which results in relatively fast recoveries. On the other hand, a potential attacker is forced to spend considerable amounts of time while going though the protocol every time he/she tries to compromise the system. Is has been shown that an exhaustive search would take $8.82\text{x}10^{66}$ years to be performed against a LEO satellite, independently of the computational power of the attacker. Therefore, a high security level is achieved by the trusted module approach.

The challenge-response protocol imposes crescent recovery delays that grow proportionally to the distance between the ground station and the spacecraft. Unlike ground-based networks, where communication can be performed very quickly, a 3-way communication with the spacecraft orbiting Mars, for instance, would take about 39 minutes. Considering the mission concept adopted, this delay may not be acceptable by ground controllers. It is important to emphasize that the utilization of such a protocol is not mandatory. Thus, mission planners and operators have the option of utilizing an one-way communication with the spacecraft to send one-time secrets. Notice, however, that in this case an attacker can perform an exhaustive search by sending to spacecraft one one-time secrets after the other. The only limitation is the processing time within the trusted platform, which is below a hundred nanoseconds. Therefore, if such one-way communication approach is to be utilized, the size of the one-time secrets would have to be increased accordingly, so that an exhaustive search attack is still infeasible. Consequently, more on-board storage will be necessary to implement the secrets table.

The proposed trusted platform can be used to thwart all capabilities of an attacker as considered in the threat model presented in Section 4.3.1. An attacker can indeed listen to all RF signals employed in the recovery protocol, such as the random r (and r') as well as $r \oplus s$ (and $r' \oplus n$). Therefore, it is trivial for an attacker to compute s (and n). However, any replay attack is thwarted by the one-time use of secrets, which are destroyed by the trusted module right after their utilization. Besides, although an attacker can build a message in a valid data format and send it to the spacecraft, he/she can perform no

better than guess a valid one-time secret. The attacker can launch an exhaustive search attack, but this approach has been shown to be infeasible.

If an attacker happens to gain control of the computation platform by either exploring a security bug or breaking a cryptographic mechanism, it is still possible for operators to regain control of the spacecraft. This is possible even when such an intruder has reconfigured the untrusted computational platform. The trusted configuration relies on a secure bus path to check the integrity of SDRAM and the FPGA configuration, which cannot be altered by an attacker. Besides, given a direct hardware connection with the communication modules there is no means for an attacker to disrupt the communication with the control center. Therefore, if any on-board component is tampered, the control center will necessarily be informed. As a consequence, procedures can be initiated to bring the computational platform to a safe state.

Ground-based network settings can usually rely on abundant implementation resources, power supply, computational power, and high bandwidth. This is not the scenario encountered by spaceborne systems, which require very constrained implementations. For instance, the FPGA (Altera CycloneII EP2C35F672C6 [38]) utilized to implement the trusted modules has a total of 33,216LEs and 483,840 RAM bits. In respect to implementation area the TRM and TKM modules utilize about 5% and 7%, respectively, of the total LEs available in this FPGA. Storage for one-time secrets of TRM and TKM have utilized respectively 14% and 21% of the FPGA's memory elements. Besides, the maximum dynamic power consumption is less than 41mW. The relatively low power consumption is due to simple operations involved with the recovery, which are basically memory accesses, additions, and a few logical operations. furthermore, low bandwidth is required by the recovery mechanisms given that each communication with the spacecraft does not take more than the secret size utilized, independently of a reset or key recovery being performed.

9.2 Fault Tolerant SHA-2 and HMAC

This section highlights the most important improvements brought by the proposed fault tolerant schemes based on information redundancy. Due to the lack of similar work on error detection and correction, the most appropriate strategy to compare the proposed approach with is TMR.

As listed in Tables 6.1 and 7.1, the employment of Hamming codes to encode the memory allows for 60% savings in memory requirements for both SHA-256 and HMAC/SHA-256 in comparison with `FullTMR`. In the case of SHA-512 and HMAC/SHA-512 the savings

reached 63%. Likewise, HCRegs&HCMem allows for the same savings (in %) in relation to FullTMR through the utilization of encoded registers, as shown in Tables 6.2 and 7.2.

Certain trends in the implementation area of the fault tolerant modules can be observed in Figure 9.1. As expected, FullTMR occupies about three times as much area as NoFT. Understandably, TMRRegs&HCMem is slightly bigger than FullTMR due to the number of voters employed. The best area efficiency is achieved with HCRegs&HCMem. This approach utilizes in the average only 58% of the area of FullTMR.

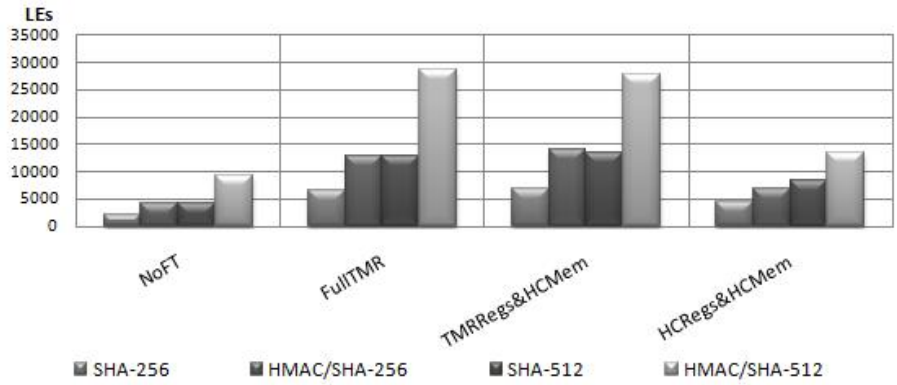


Figure 9.1: SHA-2 and HMAC Graphical Comparison of Implementation Area

Undeniably, the fault tolerant mechanism provides the highest frequencies of operation is FullTMR. That reflects on the throughput, which tends to be very close to the ones obtained by NoFT, as can be observed in Figure 9.2. On the other hand, the average throughput of HCRegs&HCMem represents a reduction of 35% when compared to FullTMR.

The HCRegs&HCMem scheme can provide an average power saving of 50% in comparison with FullTMR, as graphically shown in Figure 9.3. These results represent a considerable economy in the power consumption compared to the FullTMR traditionally employed in space applications.

It is interesting to notice that, although TMRRegs&HCMem utilizes slightly more area than FullTMR, the dynamic power consumption of the former is much lower than the latter. In fact, the dynamic power consumption of TMRRegs&HCMem modules are similar to the HCRegs&HCMem ones. That is explained by the fact that TMRRegs&HCMem employs LEs exclusively to implement registers. Therefore, there is not much signal triggering caused by the look-up tables of those LEs. As a consequence, their dynamic power consumption end up being very low.

With regard to memory and register resistance, it can be observed that TMRRegs&HCMem

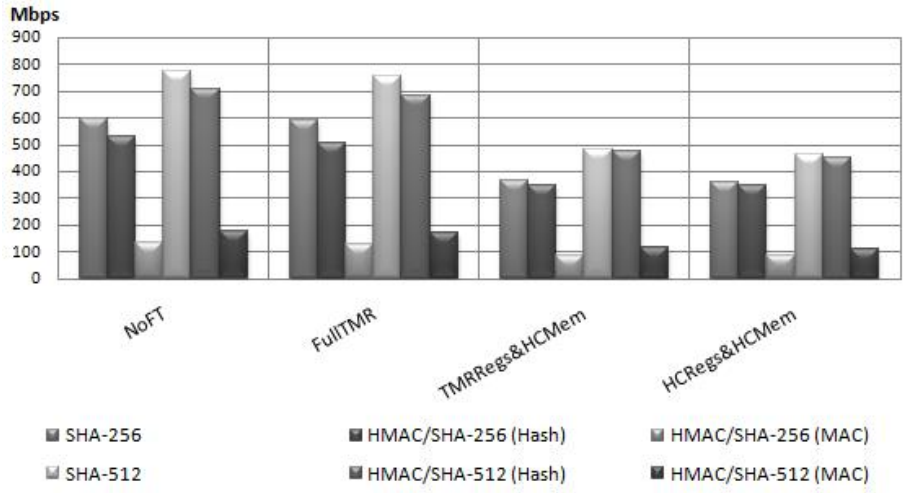


Figure 9.2: SHA-2 and HMAC Graphical Comparison of Throughput

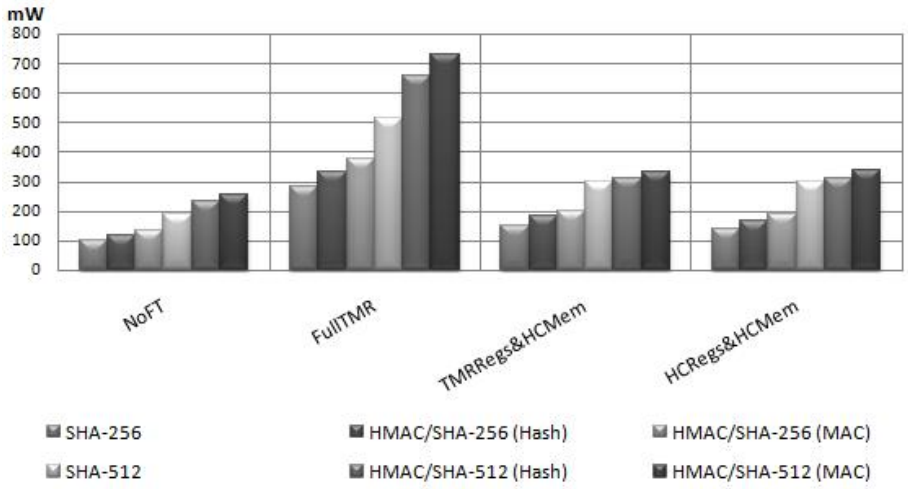


Figure 9.3: SHA-2 and HMAC Graphical Comparison of Dynamic Power Consumption

provides very high levels of protection against SEUs. However that comes at the cost of utilizing more area than `FullTMR`. Also, the former approach provides lower throughput than the latter one. Consequently, `TMRRegs&HCMem` does not result in a good trade-off in terms of implementation parameters and resistance against SEUs.

In turn, the `HCRegs&HCMem` approach presents low probability of failure. Precisely, the memory of SHA-512 and HMAC/SHA-512 using this fault tolerant scheme became respectively 161 and 171 times more resistant than the `FullTMR` approach. Even higher

resistance is obtained by the registers. Specifically, the registers of HMAC/SHA-512 employing `HCRegs&HCMem` became 491 times more resistant than when using `FullTMR`.

Another interesting aspect to be analyzed is the normalized resistance against SEUs of each fault tolerant schemes when they are applied to different designs, and how much more efficient they are in comparison with `FullTMR`. If the normalized memory resistance of HMAC/SHA-2 is compared with the SHA-2 ones, it is possible to notice that the memory resistance of SHA-2 is higher than HMAC/SHA-2. Although both algorithms use the same memory encoding scheme, HMAC requires much more iterations to perform its computations than SHA-2. Consequently, the memory in HMAC is susceptible to SEUs for a longer period of time therefore reducing its resistance to bit-flips. Therefore, HMAC faces in higher probability of memory failure compared to SHA-2.

On the other hand, HMAC achieves higher normalized register resistance than SHA-2. This comparison shows that, as the register requirements and processing time increase, the `HCRegs&HCMem` scheme becomes more effective against SEUs compared to `FullTMR`. The reason for such higher protection is that all registers are kept encoded all the time and that error detection and correction is performed more often than `FullTMR`.

Regarding fault attacks, the proposed fault tolerant scheme may protect against some attack models. As previously mentioned, the Hamming codes utilized in the hardware implementation of SHA-2 and HMAC are capable of detecting and correcting single bit-flips, and detecting two bit-flips. The following discussion considers that an attacker is capable of performing a fault injection, which can be targeted into an individual bit or larger portions of data. It is also assumed that such an attacker can set or reset the target storage element. Besides, the following analysis is valid for both the trusted platform and the fault tolerance scheme proposed in this research, given that both of them utilize Hamming codes to protect the data being processed within the hardware module.

Memory elements are protected against single bit-flips and are assumed not to be reconfigured during the life time of the platform. As a result, if a bit happen to be tampered by an attacker, it will remain in that state indefinitely. Therefore, an attacker is capable of compromising the correct functioning of the memory by altering the state of a second bit in such a way that the Hamming decoder would not be capable of correcting it. In this case, the Hamming decoder would be capable, though, of detecting that two bit-flips have happened. Moreover, since the considered attack model assumes that injected faults can target multiple individual bits, an attacker can manipulate data bits and, in the sequence, correct associated parity bits (used by the Hamming decoder to correct errors). In this case, the maliciously encoded data would deceive the Hamming decoder, that would take it as valid, while in fact it has been tampered by the attacker.

Notice, however, that it has been assumed that no error detection and correction is performed between two fault injections. Refreshing the encoded the memory, i.e. decoding and re-encoding its data, from time to time would allow for bit-flips to be corrected. However, it would continue to be exposed to fault injection between two refresh cycles. Depending on the refresh rate, higher protection against fault injection can be obtained. Although this method has not been considered in this research, it should be implemented in two scenarios where: 1) fault injections are expected; 2) long periods of exposure to radiation are expected.

In the case of registers, error detection and correction is performed very frequently. The maximum period of time that the data becomes exposed to intentional or unintentional bit-flips is during the encoded data idle period, as defined in Chapters 6 and 7. In order to compromise the data stored in a register, an attacker would have to precisely inject faults within a pre-determined time frame, which may last less than a few micro seconds. For instance, the data idle period for SHA-512 is 76 clock cycles, while for HMAC/SHA-512 it reaches 171 clock cycles. If an attacker is able to inject faults in the desired positions as well as correct the associated parity bits in such a short amount of time, he/she can successfully mislead the Hamming decoder. On the contrary, the Hamming decoder can correct single bit-flips and detect double bit-flips.

All in all, independently of the storage element (memory or register) the success of attacks based on injection will depend on the time frame considered for error detection and correction. It is important to remember that the proposed cryptographic mechanisms are intended to operate in space, where physical access to the devices is extremely limited (if not impossible). The main risk of suffering attacks, however, is in the pre-launch phase while systems are still easily accessible on the ground.

9.3 Conclusions

The first problem addressed by this research is the recovery of spacecrafts from failures and attacks. The approach utilized relies on the employment of trusted modules tailored to space missions needs. Such a platform relies on a trusted integrity check of the computational platform as well as a direct connection with the communication modules, which is resilient against the interference of attackers.

Trusted reset modules have been proposed to serve as a stepping stone in the recovery of the spacecraft. Similarly, trusted key recovery modules are proposed to issue temporary cryptographic keys so that a secured channel can be established with a control center after a

platform recovery is performed. The trusted modules employ a scheme similar to one-time-passwords to issue trusted resets, which is used in conjunction with a challenge-response protocol. Exhaustive search attacks are the only known attack that can be performed against the trusted modules. However, it has been shown that such an attack would take up to $8.82 \text{x} 10^{66}$ years to be performed against a LEO satellite. Therefore, it is infeasible for an attacker to break the system through exhaustive search. In spite of that, a trusted reset or key recovery can be performed in less than $80ns$ by an authentic control center.

Additionally, the trusted modules include fault tolerance mechanisms such as encoded memory and automated secret re-synchronization so that it can cope with radiation-induced faults. Moreover, the simplicity of the trusted modules design favors efficient hardware implementations. For instance, an FPGA implementation of trusted key recovery module demands less than 2400LEs, requires less than 100KB of on-board storage, and consumes less than 41mW. Besides, even though the challenge-response protocol contributes to increased system security and decreased implementation requirements, its utilization is not mandatory. Different missions concepts can discard the challenge-response protocol to achieve even faster system recovery, with the drawback of utilizing larger one-time secrets and therefore more on-board storage.

The second main problem addressed by this research is the achievement of efficient fault tolerance for cryptographic primitives. Since the trusted modules demands integrity checks, SHA-2 algorithms were adopted as the cryptographic hash functions. The proposed fault tolerant scheme has been denoted as `HCRegs&HCMem` and employs information redundancy to protect memory and registers. In order to evaluate how `HCRegs&HCMem` would behave as the implementation requirements increases, the application of such a technique to HMAC was also investigated. Besides, SHA-2 and HMAC algorithms can also be employed to provide secured communications with spacecraft. Efficiency in the fault tolerance mechanism is achieved by taking advantage of SHA-2 and HMAC features so that encoding and decoding of information is minimized without compromising reliability. Furthermore, in order to determine the resistance against SEUs achieved by each approach, an evaluation metrics based on probability of failure was proposed so that it was possible to compare different fault tolerant mechanisms.

Experimental results based on FPGA implementations have shown that `HCRegs&HCMem` provides considerable savings in comparison with the traditional TMR. The proposed approach allowed for a reduction of up to 63% in memory and register requirements in comparison with TMR. Furthermore, average area savings in the order of 42% were achieved. In the case of HMAC/SHA-512, the area savings reached 53% in comparison with TMR. Moreover, the average power consumption could be reduced in 50%, with a maximum saving of 53% in the case of HMAC/SHA-512. The drawback of this approach is that its

throughput is, in the average, 39% lower than the ones provided by TMR. However, the achieved throughput may be high enough for most space applications.

In spite of lower implementation requirements, `HCRegs&HCMem` also provides higher resistance against SEUs than TMR. For instance, the memory of SHA-512 and HMAC/SHA-512 became respectively 161 and 171 times more resistant. By the same token, the registers of SHA-512 is 13 times more resistance against SEUs. Even higher resistance is achieved in HMAC/SHA-512, whose registers became 491 times more resistant than TMR. This also shows that `HCRegs&HCMem` can provide higher level protection with low implementation requirements as the algorithm complexity increases. Furthermore, it can successfully replace TMR in FPGA implementations.

These results show that the utilization of information redundancy to protect registers and memory elements can lead to efficient fault tolerant implementations of HMAC. To the best of our knowledge, this is the first proposal of fault tolerant techniques capable of performing error detection and correction for SHA-2 and HMAC in the literature.

The provision of secure recovery mechanisms as well as efficient fault tolerant cryptographic primitives can be considered as the first step towards achieving higher levels of security in space applications. Therefore, the results of this research aims at supporting the incorporation of on-board security mechanisms in order to meet the increasing security requirements of modern space missions.

9.4 Future Work

Future work includes several research threads aiming at implementation-efficient and fault tolerant security mechanisms for space applications. The first goal is to investigate the utilization of trusted hardware modules to support on-board key derivation. Key derivation based on SHS and AES would be a first priority, given the widespread use of these algorithms in secure communications. Besides, some platforms may already have an AES module available, which would make it possible to reuse it for key derivation. Another topic of research is the evaluation of the proposed fault tolerant mechanisms applied to AES as well as to to SHA-3 as soon as the algorithm competition [136] is complete.

Due to the lack of a recommendation for key management targeting space applications, this would be another topic of research. Therefore, it would be necessary to analyze a key hierarchy consisting of multiple levels, where the lower level would session or traffic keys, the next level up would be the key encryption keys (KEKs), with potential utilization of additional levels for re-keying KEKs. Besides, this would include the definition of key

lengths and corresponding crypto periods and scopes, i.e. keys required for authentication and encryption of both command, tracking, telemetry as well as payload data.

Given the severe constraints in on-board computational power, limited bandwidth and intermittent communications, public-key cryptography has not been traditionally considered to support key management in space applications. However, it would be important to conduct a feasibility study of asymmetric primitives targeting the space segment so that the space community can better understand what to use and what not to use for key management in the space setting.

Another important aspect in future space missions, specially in deep space missions, is spacecraft autonomy. This feature is also important in LEO/GEO during contingency situations. Therefore, it would be crucial to investigate how rovers, satellites and probes could collaborate to perform key agreement, assuming minimum or none interaction with a control center. Secret sharing schemes [150] would be considered candidates to perform key agreement based on virtual conferences.

Finally, yet another research topic includes the improvement of the trusted platform to permit secure reconfiguration of algorithms and on-board secrets/keying material. Post-launch reconfiguration not only allows for higher flexibility of the on-board key management system. It also plays an important role in allowing for the extension of space missions lifetimes, which has become very common in the last couple of years.

References

[1] Virtex-E 1.8V field programmable gate arrays. Data Sheet DS022-1 (v2.3), Xilinx Incorporated, July 2002.

[2] APEX 20K programmable logic device family. Data Sheet DS-APEX20K-5.1, Altera Corporation, March 2004.

[3] Virtex-II platform FPGAs. Data Sheet DS031 (v3.5), Xilinx Incorporated, November 2007.

[4] About.com. Mars Odyssey Orbiter in safe mode, Shuttle delay and Mars Odyssey Orbiter. Online, December 2006. http://space.about.com/b/2006/12/08/mars-odyssey-orbiter-in-safe-mode-shuttle-delay-and-mars-odyssey-orbiter.htm.

[5] P. Adell and G. Allen. Assessing and mitigating radiation effects in Xilinx FPGAs. JPL Publication 08-9 2/08, Jet Propulsion Laboratory, California Institute of Technology, 2008.

[6] Infineon Technologies AG. Infineon technologies ag home page. Online, February 2011. http://www.infineon.com.

[7] Russian Space Agency. GLONASS project portal. Online, February 2011. www.glonass-ianc.rsa.ru.

[8] I. Ahmad and A. Das. Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs. *Computers and Electrical Engineering*, 31(6):345–360, 2005.

[9] I. Ahmad and A. Das. Analysis and detection of errors in implementation of SHA-512 algorithms on FPGAs. *The Computer Journal*, 50(6):728–738, 2007.

[10] Altera Corporation. *DE2 Development and Education Board User Manual*, 1.4 edition, 2006.

[11] Altera Corporation. *Quartus II Version 9.0 Handbook*, March 2009.

[12] M. Arslan and F. Alagoz. Security issues and performance study of key management techniques over satellite links. In *11th Intenational Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*, pages 122–128, 2006.

[13] Atmel Corporation. *Rad Hard Reprogrammable FPGA ATF280E*, 2007. 7750AAERO07/07.

[14] S. Bain. The increasing threat to satellite communications. *Online Journal of Space Communication*, 6, November 2003.

[15] R. Banu and T. Vladimirova. Fault-tolerant encryption for space applications. *IEEE Transactions on Aerospace and Electronic Systems*, 45(1):266 –279, January 2009.

[16] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370 –382, February 2006.

[17] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Transactions on Computers*, 52(4):492–505, April 2003.

[18] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 513–525. Springer-Verlag, 1997.

[19] J. Blöemer and J. Seifert. Fault based cryptanalysis of the advanced encryption standard. Cryptology ePrint Archive, Report 2002/075, 2002. http://eprint.iacr.org.

[20] H. Brandl and T. Rosteck. *Technology, Implementation and Application of the Trusted Computing Group Standard (TCG).* Infineon Technologies AG, September 2004.

[21] C. Carmichael, M. Caffrey, and A. Salazar. *Correcting Single-Event Upsets Through Virtex Partial Configuration.* Xilinx Incorporated, June 2000. XAPP216.

[22] C. Carmichael, E. Fuller, P. Blain, and M. Caffrey. SEU mitigation techniques for Virtex FPGAs in space applications. In *MAPLD Proceedings*, 1999.

[23] CCSDS. The application of CCSDS protocols to secure systems. Informational Report CCSDS 350.0-G-2, CCSDS, January 2006.

[24] CCSDS. Security threats against space missions. Informational Report CCSDS 350.1-G-1, CCSDS, October 2006.

[25] CCSDS. Recommended practice for a key management scheme. Draft Recommended Practice CCSDS 000.0-R-0, CCSDS, September 2007. Red Book.

[26] CCSDS. Recommended practice for authentication. Draft Recommended Practice CCSDS 000.0-R-0, CCSDS, March 2007. Red Book.

[27] CCSDS. Authentication / integrity algorithm issues survey. Informational Report CCSDS 350.1-G-1, CCSDS, March 2008. Green Book.

[28] CCSDS. Encryption algorithm trade survey. Informational Report CCSDS 350.2-G-1, CCSDS, March 2008. Green Book.

[29] CCSDS. Symmetric encryption. Draft Recommended Practice CCSDS 353.0-R-1, CCSDS, March 2008. Red Book.

[30] CCSDS. CCSDS home page. Online, February 2010. www.ccsds.org.

[31] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Improving SHA-2 hardware implementations. In *Proceedings of the Cryptographic Hardware and Embedded Systems (CHES 2006)*, pages 298–310, 2006.

[32] Actel Corporation. Actel launches flash-based FPGAs into space. Online, September 2008. http://www.actel.com/company/press/2008/9/15/1.

[33] Actel Corporation. *Radiation-Tolerant ProASIC3 Low Power Space-Flight Flash FPGAs with FlashFreeze Technology*. Actel Corporation, 1 edition, November 2009.

[34] Actel Corporation. *RTAX-S/SL and RTAX-DSP Radiation-Tolerant FPGAs*. Actel Corporation, August 2010.

[35] Actel Corporation. *RTSX-SU RadTolerant FPGAs*. Actel Corporation, October 2010.

[36] Actel Corporation. Actel corporation home page. Online, February 2011. www.actel.com.

[37] Altera Corporation. Altera HardCopy IV ASICs for military applications. Online, September 2008. www.altera.com/literature/po/ss-hciv_military.pdf.

[38] Altera Corporation. *Cyclone II Device Handbook*, February 2008. CII5V1-3.3.

[39] Altera Corporation. *Error Detection and Recovery Using CRC in Altera FPGA Devices*. Altera Corporation, July 2008. AN 357.

[40] Altera Corporation. *Nios II Processor Reference Handbook*. Altera Corporation, December 2010. NII5V1-10.1.

[41] Altera Corporation. Altera corporation home page. Online, February 2011. www.altera.com.

[42] Altera Corporation. Altera's enhanced cots initiative overview. Online, February 2011. http://www.altera.com/end-markets/military-aerospace/overview/mil-overview.html.

[43] Atmel Corporation. *Trusted Platform Module – Atmel AT97SC3204 LPC Interface*. Atmel Corporation, 2010. 5294BSTPM9/10.

[44] Atmel Corporation. Atmel corporation home page. Online, February 2011. www.atmel.com.

[45] Broadband Corporation. High-performance security processor – BCM5825 product brief, 2006.

[46] Broadcom Corporation. Broadcom corporation home page. Online, February 2011. http://www.broadcom.com.

[47] Lattice Semiconductor Corporation. Lattice semiconductor corporation home page. Online, February 2011. www.latticesemi.com.

[48] QuickLogic Corporation. QuickLogic corporation home page. Online, February 2011. www.quicklogic.com.

[49] China Daily. Falun Gong hijacks HK satellite. Online, November 2004. www.chinadaily.com.cn/english/doc/2004-11/22/content_393776.htm.

[50] A. Dominguez-Oviedo and A. Hasan. Error detection and fault tolerance in ECSM using input randomization. *IEEE Transactions on Dependable and Secure Computing*, 6(3):175 –187, 2009.

[51] P. Dusart, G. Letourneux, and O. Vivolo. Differential fault analysis on A.E.S. In *Applied Cryptography and Network Security*, volume 2846 of *Lecture Notes in Computer Science*, pages 293–306. Springer Berlin / Heidelberg, 2003.

[52] ESA. Galileo - the European programme for global navigation services. Online, http://esamultimedia.esa.int/docs/galileo/GalileoE3web_copy.pdf, January 2005.

[53] ESA. ESA portal. Online, February 2011. www.esa.int.

[54] ESA. Galileo project portal. Online, February 2011. www.esa.int/esaNA/galileo.html.

[55] ESA. Rosetta project home page. Online, February 2011. www.esa.int/esaMI/Rosetta/index.html.

[56] S. Fleming. Hacker infiltrates military satellite. Online: The Register, March 1999. http://www.theregister.com/1999/03/01/ hacker_infiltrates_military_satellite/.

[57] International Organization for Standardization. Information technology – trusted platform module – part 1: Overview. International Standard ISO/IEC 11889-1:2009(E), International Organization for Standardization, May 2009.

[58] International Organization for Standardization. Information technology – trusted platform module – part 2: Design principles. International Standard ISO/IEC 11889-2:2009(E), International Organization for Standardization, May 2009.

[59] International Organization for Standardization. Information technology – trusted platform module – part 3: Structures. International Standard ISO/IEC 11889-3:2009(E), International Organization for Standardization, May 2009.

[60] International Organization for Standardization. Information technology – trusted platform module – part 4: Commands. International Standard ISO/IEC 11889-4:2009(E), International Organization for Standardization, May 2009.

[61] International Organization for Standardization. International organization for standardization home page. Online, February 2011. http://www.iso.org.

[62] FPGA and Programmable Logic Journal. All is not SRAM - a survey of flash, anti-fuse, and EE programmable logic. Online, February 2004. http://www.fpgajournal.com/articles/sram.htm.

[63] FPGA and Programmable Logic Journal. FPGAs in space. Online, August 2004. http://www.techfocusmedia.net/archives/articles/20040803_space/.

[64] FPGA and Structured ASIC Journal. Space FPGAs get a boost. Online, June 2007. http://www.techfocusmedia.net/archives/articles/20070626_space/.

[65] Aeroflex Gaisler. *LEON3 Multiprocessing CPU Core*. Aeroflex Gaisler, February 2010.

[66] Aeroflex Gaisler. Aeroflex Gaisler home page. Online, February 2011. http://www.gaisler.com.

[67] Aeroflex Gaisler. *LEON3-FT SPARC V8 Processor*. Aeroflex Gaisler, 1.2 edition, January 2011. http://www.gaisler.com/doc/leon3ft-rtax-ag.pdf.

[68] C. Giraud. DFA on AES. In *Advanced Encryption Standard AES*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer Berlin / Heidelberg, 2005.

[69] GlobalSecurity.org. SCORE (signal communication by orbiting relay equipment). Online, September 2008. www.globalsecurity.org/space/systems/score.htm.

[70] G. Gordon and W. Morgan. *Principles of Communications Satellites*. John Wiley & Sons, Incorporated, 1993.

[71] P. Graham, M. Caffrey, J. Zimmerman, P. Sundararajan, E. Johnson, and C. Patterson. Consequences and categories of SRAM FPGA configuration SEUs. In *MAPLD Proceedings*, 2003.

[72] The Landfield Group. Hackers seize UK military satellite. Online, March 1999. http://www.landfield.com/isn/mail-archive/1999/Mar/0001.html.

[73] Trusted Computing Group. Trusted computing group home page. Online, February 2011. http://www.trustedcomputinggroup.org.

[74] S. Habinc. Suitability of reprogrammable FPGAs in space applications. Feasibility Report FPGA-002-01, Gaisler Research, September 2002. Version 0.4.

[75] R. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, April 1950.

[76] M. Hsueh, T. Tsai, and R. Iyer. Fault injection techniques and tools. *Computer*, 30(4):75 –82, April 1997.

[77] Aeroflex Colorado Springs Incorporated. *UT6325 RadTol Eclipse FPGA*. Aeroflex Colorado Springs Incorporated, September 2008.

[78] Free Software Foundation Incorporated. GNU General Public License. Online, June 2007. http://www.gnu.org/copyleft/gpl.html.

[79] Microchip Technology Incorporated. *PIC16F8X 18-pin Flash/EEPROM 8-Bit Microcontrollers*. Microchip Technology Incorporated, 1998. DS30430C.

[80] Xilinx Incorporated. Virtex-5 EasyPath FPGAs: Cost reduction for highly integrated FPGA platforms. Online, September 2008. www.xilinx.com/publications/prod_mktg/pn2019.pdf.

[81] Xilinx Incorporated. *PicoBlaze 8-bit Embedded Microcontroller User Guide for Spartan-3, Spartan-6, Virtex-5, and Virtex-6 FPGAs*. Xilinx Incorporated, January 2010. UG129 (v2.0).

[82] Xilinx Incorporated. *Radiation-Hardened, Space-Grade Virtex-5QV Device Overview*. Xilinx Incorporated, August 2010. DS192(v1.1).

[83] Xilinx Incorporated. *Space-Grade Virtex-4QV Family Overview*. Xilinx Incorporated, April 2010. DS653(v2.0).

[84] Xilinx Incorporated. Xilinx incorporated home page. Online, February 2011. www.xilinx.com.

[85] Infowars.com. Falun Gong denies sending China pirate signals. Online, March 2005. www.infowars.com/articles/world/china_falun_gong_denies_pirate_signals.htm.

[86] I. Ingemarsson and C. Wong. Encryption and authentication in on-board processing satellite communication systems. *IEEE Transactions on Communications*, 29(11):1684–1687, November 1981.

[87] B. Johnson. *Design and Analysis of Fault Tolerant Digital Systems*. Addison-Wesley, 1989.

[88] M. Juliato and C. Gebotys. An approach for recovering satellites and their cryptographic capabilities in the presence of SEUs and attacks. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2008)*, pages 101–108, June 2008.

[89] M. Juliato and C. Gebotys. SEU-resistant SHA-256 design for security in satellites. In *Proceedings of the 10th International Workshop on Signal Processing for Space Communications (SPSC 2008)*, October 2008.

[90] M. Juliato and C. Gebotys. Tailoring a reconfigurable platform to SHA-256 and HMAC through custom instructions and peripherals. In *Proceedings of the 2009 International Conference on Reconfigurable Computing and FPGAs (Reconfig'09)*, pages 195–200, December 2009.

[91] M. Juliato and C. Gebotys. An efficient fault-tolerance technique for the keyed-hash message authentication code. In *Proceedings of the 2010 IEEE Aerospace Conference (Aeroconf'10)*, pages 1–17, March 2010.

[92] M. Juliato and C. Gebotys. FPGA implementation of an HMAC processor based on the SHA-2 family of hash functions. CACR Technical Report CACR 2011-10, University of Waterloo, April 2011. http://www.cacr.math.uwaterloo.ca/.

[93] M. Juliato, C. Gebotys, and R. Elbaz. Efficient fault tolerant SHA-2 hash functions for space applications. In *Proceedings of the 2009 IEEE Aerospace Conference (Aeroconf'09)*, pages 1–16, March 2009.

[94] E. Khan, M. El-Kharashi, F. Gebali, and M. Abd-El-Barr. Design and performance analysis of a reconfigurable, unified HMAC-hash unit. *IEEE Transactions on Circuits and Systems I*, 54(12):2683–2695, December 2007.

[95] C. Kim and J. Quisquater. Faults, injection methods, and fault attacks. *Design Test of Computers, IEEE*, 24(6):544 –545, November-December 2007.

[96] M. Kim, Y. Kim, J. Ryou, and S. Jun. Efficient implementation of the keyed-hash message authentication code based on SHA-1 algorithm for mobile trusted computing. In *Proceedings of the 4th International Conference on Autonomic and Trusted Computing (ATC 2007)*, pages 410–419, 2007.

[97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, IETF Network Working Group, February 1997. http://www.ietf.org/rfc/rfc2104.txt.

[98] Jet Propulsion Lab. Status report: Engineers assessing cassini spacecraft. Online, November 2010. http://saturn.jpl.nasa.gov/news/newsreleases/newsrelease20101104/.

[99] B. LaMeres and C. Gauer. Dynamic reconfigurable computing architecture for aerospace applications. In *Proceedings of the 2009 IEEE Aerospace Conference (Aeroconf'09)*, March 2009.

[100] B. LaMeres and C. Gauer. Spatial avoidance of hardware faults using FPGA partial reconfiguration of tile-based soft processors. In *Proceedings of the 2010 IEEE Aerospace Conference (Aeroconf'10)*, March 2010.

[101] F. Lima, L. Carro, and R. Reis. Designing fault tolerant systems into SRAM-based FPGAs. In *Proceedings of the Design Automation Conference 2003*, pages 650–655, June 2003.

[102] F. Lima, L. Carro, and R. Reis. *Fault-Tolerance Techniques for SRAM-Based FPGAs*. Springer, June 2006.

[103] Discover Magazine. A signal event: on the track of Capt. Midnight - home box office's transmission interrupted. Online, July 1986. http://findarticles.com/p/articles/mi_m1511/is_v7/ai_4293600.

[104] M. McLoone and J. McCanny. Efficient single-chip implementation of SHA-384 and SHA-512. *2002 IEEE International Conference on Field-Programmable Technology (FPT'02)*, pages 311–314, 2002.

[105] M. McLoone and J. McCanny. A single-chip IPSec cryptographic processor. pages 133–138, October 2002.

[106] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[107] G. Messenger and M. Ash. *Single Event Phenomena*. Springer, 2006.

[108] H. Michail, A. Kakarountas, A. Milidonis, and C. Goutis. Efficient implementation of the keyed-hash message authentication code (HMAC) using the SHA-1 hash function. In *Proceedings of the 11th IEEE International Conference on Electronics, Circuits and Systems*, pages 567–570, 2004.

[109] H. Michail, A. Kakarountas, G. Selimis, and C. Goutis. Optimizing SHA-1 hash function for high throughput with a partial unrolling study. In *PATMOS 2005*, pages 591–600, 2005.

[110] ST Microelectronics. Trusted platform module (TPM) with complete TCG software solution, 2004.

[111] ST Microelectronics. ST microelectronics home page. Online, February 2011. http://www.st.com.

[112] A. Moradi, M. Shalmani, and M. Salmasizadeh. A generalized method of differential fault attack against AES cryptosystem. In *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 91–100. Springer Berlin / Heidelberg, 2006.

[113] D. Morrill. Hack a satellite while it is in orbit. Online, April 2007. http://it.toolbox.com/blogs/managing-infosec/hack-a-satellite-while-it-is-in-orbit-15690.

[114] NASA. First FPGA in space. Online, January 2006. http://klabs.org/home_page/first_fpga/index.htm.

[115] NASA. Artificial satellites. Online, 2007. www.nasa.gov/worldbook/artificial_satellites_worldbook.html.

[116] NASA. Mars Reconnaissance Orbiter mission status. Online, March 2007. www.nasa.gov/mission_pages/MRO/news/mrof-20070322.html.

[117] NASA. Sputnik and the dawn of the space age. Online, October 2007. http://history.nasa.gov/sputnik/.

[118] NASA. Mars reconnaissance orbiter mission status report. Online, November 2009. http://www.nasa.gov/mission_pages/MRO/news/mro-20091208.html.

[119] NASA. Cassini-Huygens mission home page. Online, February 2011. www.nasa.gov/mission_pages/cassini/main/index.html.

[120] NASA. Hubble Space Telescope portal. Online, February 2011. hwww.nasa.gov/mission_pages/hubble.

[121] NASA. Radiation effects & analysis group. Online, February 2011. http://radhome.gsfc.nasa.gov.

[122] NASA. Solar system exploration webpage. Online, February 2011. http://solarsystem.nasa.gov/planets/index.cfm.

[123] NASA. Space Shuttle portal. Online, February 2011. www.nasa.gov/mission_pages/shuttle.

[124] BBC News. HK probes Falun Gong 'hacking'. Online, November 2004. http://news.bbc.co.uk/2/hi/asia-pacific/4034209.stm.

[125] BBC News. Galileo price rises 1.9 billion euros. Online, January 2011. http://www.bbc.co.uk/news/science-environment-12220537.

[126] Telegraph Newspaper. British hackers attack MoD satellite, March 1999. http://www.telegraph.co.uk/connected/main.jhtml?xml=/connected/1999/03/04/ecnhack04.xml.

[127] NIST. Computer data authentication. Federal Information Processing Standards Publication FIPS PUB 113, NIST, May 1985.

[128] NIST. Advanced encryption standard (AES). Federal Information Processing Standards Publication FIPS PUB 197, NIST, November 2001.

[129] NIST. The keyed-hash message authentication code (HMAC). Federal Information Processing Standards Publication FIPS PUB 198, March 2002.

[130] NIST. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. NIST Special Publication SP 800-38C, NIST, May 2004.

[131] NIST. Recommendation for block cipher modes of operation: The CMAC mode for authentication. NIST Special Publication SP 800-38B, NIST, May 2005.

[132] NIST. The keyed-hash message authentication code (HMAC). Federal Information Processing Standards Publication FIPS PUB 198-1, NIST, July 2008.

[133] NIST. Secure hash standard (SHS). Federal Information Processing Standards Publication FIPS PUB 180-3, NIST, October 2008.

[134] NIST. Digital signature standard (DSS). Federal Information Processing Standards Publication FIPS PUB 186-3, June 2009.

[135] NIST. Recommendation for applications using approved hash algorithms. NIST Special Publication SP 800-107, NIST, February 2009.

[136] NIST. Cryptographic hash algorithm competition. Online, February 2011. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[137] Union of Concerned Scientists (UCS). UCS satellite database. Online, February 2011. http://www.ucsusa.org/nuclear_weapons_and_global_security/space_weapons/technical_issues/ucs-satellite-database.html.

[138] Embassy of the People's Republic of China in the United Stated of America. Chinese satellite TV hijacked by Falun Gong cult. Online, October 2003. http://www.china-embassy.org/eng/zt/ppflg/t36611.htm.

[139] JPL Electronic Parts Engineering Office. JPL electronic parts engineering office home page. Online, February 2011. http://parts.jpl.nasa.gov/.

[140] United States General Accounting Office. Critical infrastructure protection: Commercial satellite security should be more fully addressed. Technical Report GAO-02-781, United States General Accounting Office, 2002.

[141] Council on Foreign Relations. Liberation Tigers of Tamil Eelam. Online, July 2008. www.cfr.org/publication/9242.

[142] Asia Times Online. Falungong sabotages chinese satellite TV. Online, March 2005. www.atimes.com/atimes/China/GC26Ad04.html.

[143] E. Papoutsis, G. Howells, A. Hopkins, and K. McDonald-Maier. Key generation for secure inter-satellite communication. In *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pages 671–681. IEEE Computer Society, 2007.

[144] J. Patel and L. Fung. Concurrent error detection in ALUs by recomputing with shifted operands. *IEEE Transactions on Computers*, C-31(7):589–595, July 1982.

[145] K. Poulsen. Satellites at risk of hacks. Online: SecurityFocus, October 2002. http://www.securityfocus.com/news/942.

[146] A. Roy-Chowdhury, J. Baras, M. Hadjitheodosiou, and S. Papademetriou. Security issues in hybrid networks with a satellite component. *IEEE Wireless Communications*, 12(6):50–61, 2005.

[147] J. Schmidt, M. Hutter, and T. Plos. Optical fault attacks on AES: A threat in violet. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on*, pages 13 –22, September 2009.

[148] ScienceDaily. Orbiter puts itself into standby safe mode. Online, July 2010. http://www.sciencedaily.com/releases/2010/07/100720215455.htm.

[149] G. Selimis, N. Sklavos, and O. Koufopavlou. VLSI implementation of the keyed-hash message authentication code for the wireless application protocol. In *Proceedings of 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS'03)*, pages 24–27, December 2003.

[150] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[151] N. Sklavos and O. Koufopavlou. Implementation of the SHA-2 hash family standard using FPGAs. *The Journal of Supercomputing*, 31(3):227–248, 2005.

[152] S. Skorobogatov and R. Anderson. Optical fault induction attacks. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, pages 2–12. Springer-Verlag, 2003.

[153] Slashdot. Sri Lankan terrorists hack satellite. Online, April 2007. http://it.slashdot.org/article.pl?sid=07/04/13/068222.

[154] The Planetary Society. Mars explorations rovers update: Spirit investigates new outcrop methuselah as opportunity roves into a ripple and gets stuck. Online, April 2005. www.planetary.org/news/2005/0428_Mars_Explorations_Rovers_Update_Spirit.html.

[155] The Planetary Society. Cassini went into safe mode, but everything is OK. Online, September 2007. www.planetary.org/blog/article/00001129/.

[156] The Planetary Society. Mars Odyssey Orbiter is in safe mode. Online, December 2007. www.planetary.org/blog/article/00000791/.

[157] The Planetary Society. Dawn journal: Software updates. Online, April 2008. www.planetary.org/blog/article/00001404/.

[158] The Planetary Society. Report on Phoenix sol 9 activities: Ready to get samples; but Odyssey is in safe mode. Online, June 2008. www.planetary.org/blog/article/00001495/.

[159] Space.com. Nasa's Comet Probe in safe-mode, but healthy, after launch. Online, January 2005. www.space.com/missionlaunches/delta2_dpimpact_050112.html.

[160] Space.com. Hubble telescope's main camera stops working. Online, June 2006. www.space.com/news/060623_hubble_acs.html.

[161] Space.com. Cassini in safe mode after Saturn flight. Online, September 2007. www.space.com/scienceastronomy/070912_ap_cassini_safe.html.

[162] Space.com. Mars Orbiter back at full strength. Online, September 2007. www.space.com/scienceastronomy/070921_marsodyssey_caves.html.

[163] Space.com. Mars Orbiter in safe mode after glitch. Online, September 2007. www.space.com/missionlaunches/070918_mo_glitch.html.

[164] Space.com. Space computer hiccup sidelines mars spacecraft. Online, 2010 September. http://www.space.com/9145-space-computer-hiccup-sidelines-mars-spacecraft.html.

[165] Spaceref.com. Kepler goes into safe mode. Online, December 2010. http://www.spaceref.com/news/viewsr.html?pid=35606.

[166] SPARC International Incorporated. *The SPARC Architecture Manual*, version 8 edition. Revision SAV080SI9380.

[167] D. Stinson. *Cryptography Theory and Practice*. CRC Press, 3rd edition, 2005.

[168] M. Tafazoli. A study of on-orbit spacecraft failures. *Acta Astronautica*, 64(2-3):195–205, 2009.

[169] K. Ting, S. Yuen, K. Lee, and P. Leong. An FPGA based SHA-256 processor. In *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications (FPL '02)*, pages 577–585. Springer-Verlag, 2002.

[170] M. Wang, C. Su, C. Huang, and C. Wu. An HMAC processor with integrated SHA-1 and MD5 algorithms. In *Proceedings of the 2004 Conference on Asia South Pacific Design Automation (ASP-DAC'04)*, pages 456–458, 2004.

[171] I. Yiakoumis, M. Papadonikolakis, and H. Michail. Efficient small-sized implementation of the keyed-hash message authentication code. In *Proceedings of the IEEE Eurocon Conference 2005 Computer as a Tool*, volume 2, pages 1875–1878, November 2005.