Kyushu Institute of Technology
Dept. of Applied Science for Integrated
System Engineering
Graduate School of Engineering

Development and Qualification of an FPGA-Based Multi-Processor System-on-Chip On-Board Computer for LEO Satellites

By

Mohamed Mahmoud Mohamed Ibrahim

A THESIS SUBMITTED TO THE
GRADUATE SCHOOL OF ENGINEERING
AT KYUSHU INSTITUTE OF TECHNOLOGY
IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF ENGINEERING
IN APPLIED SCIENCE FOR INTEGRATED SYSTEM
ENGINEERRING

Graduate School of Engineering, Kyushu Institute of Technology
Kitakyushu, Japan
2014

Kyushu Institute of Technology
Dept. of Applied Science for Integrated System Engineering
Graduate School of Engineering

Development and Qualification of an FPGA-Based Multi-Processor System-on-Chip On-Board Computer for LEO Satellites

By

Mohamed Mahmoud Mohamed Ibrahim

A THESIS SUBMITTED TO THE
GRADUATE SCHOOL OF ENGINEERING
AT KYUSHU INSTITUTE OF TECHNOLOGY
IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF ENGINEERING
IN APPLIED SCIENCE FOR INTEGRATED SYSTEM
ENGINEERRING

Thesis Supervisor
Assoc. Prof. Kenichi Asami
Dept. of Integrated Systems Engineering
Kyushu Institute of Technology

Kyushu Institute of Technology
Dept. of Applied Science for Integrated
System Engineering
Graduate School of Engineering

Development and Qualification of an FPGA-Based Multi-Processor System-on-Chip On-Board Computer for LEO Satellites

By

Mohamed Mahmoud Mohamed Ibrahim

A THESIS SUBMITTED TO THE
GRADUATE SCHOOL OF ENGINEERING
AT KYUSHU INSTITUTE OF TECHNOLOGY
IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF ENGINEERING
IN APPLIED SCIENCE FOR INTEGRATED SYSTEM ENGINEERRING

Approved by the examiners committee:

Prof. Mengu Cho Cho 趙孟佑印

Prof. Keiichi Okuyama Keiichi Okuyama 奥山

Assoc. Prof. Kenichi Asami Kenichi Asami 浅海

Assoc. Prof. Kazuhiro Toyoda Kazuhiro Toyoda 豊田

Assoc. Prof. Motoki Miura Motoki Miura 三浦

Acknowledgment

All of the thanks go to the almighty ALLAH for giving me the strength and ability to finish this work. It is all out of his willingness and endowments upon us. May Allah reward us for the time and effort spent in producing this work. I ask for forgiveness in case you find any pitfalls in it and for prayer in case you find it beneficial.

When I joined the Egyptian Space Program (ESP), more than 10 years ago, I was deeply interested in learning more about this field of high fidelity and ultimately critical engineering, namely space engineering. I tried to do all what I could in order to learn and become a real expert in that field. During my struggle to learn, I travelled to many places, knew a lot of things and dealt with many nationalities from the west to the east of this planet. In that pervasive period of intensive education and exposure to different cultures, I learnt that all of us are alike even though cultural differences do exist among us. We are alike in our feelings, in our willingness, in our ambitions, and in our love for peace and justice.

In that journey of education, I did learn the true meaning of working for the sake of the bigger image instead of the narrow field of interests. Work for humanity, in general and without discrimination, to make the life of people better, more affordable and happier. I truly knew the value and meaning of the statement “science has no borders”. It is true because science is the resemblance of justice, truth, and equity and those meanings has no borders to cross, they are just intrinsic in good spirits and souls.

The work in this thesis would not have been possible without the support of many staff, organizations and individuals. My words are not sufficiently eloquent to express my gratitude feelings towards them. First, I would like to express my deep gratitude to the ESP and the National Authority for Remote Sensing and Space Sciences (NARSS) at Egypt for giving me the chance to work in the field of space engineering. Special thanks to Prof. Ayman El Dessouki, former chairman of NARSS, for approving and recommending my application to this fellowship through which I was granted my doctoral degree. I do appreciate the efforts and work of all of

the employees at ESP and NARSS and wish them a brilliant future. Thanks to my master degree supervisor, the late Prof. Yousri El Nahas, and to the late Prof. Matar Ali Matar, may their souls rest in peace.

I would like to thank my supervisor, Assoc. Prof. Kenichi Asami, for supervising me during my study at Kyuhsu Institute of Technology (KIT). Words cannot express my gratitude for his kindness, understanding, support and deep guidance. My appreciation and thanks to Prof. Mengu Cho, the scientist, the innovator, the teacher, the motivator and above all the mentor of our success. Deep appreciation for Dr. Werner Balogh at the United Nations Office for Outer Space Affairs (UNOOSA) and all the staff there for the extraordinary efforts in inaugurating the Doctorate in Nano-Satellite Technology (DNST) program in collaboration with KIT.

My appreciation and thanks to the professors at KIT for teaching us and giving us from their effort and time. Thanks to Prof. Keiichi Okuyama, Assoc. Prof. Kazuhiro Toyoda, Assoc. Prof. Motoki Miura and Assoc. Prof. Sozo Inoue. I do appreciate the efforts of the administrative staff at KIT for providing support and help during our study. I would like to thank Kanae Iyama, Kumiko Shirakawa, Michiko Kubota and Seiji Kawano.

Special thanks to my brothers and colleagues at KIT for making my stay in Japan more knowledgeable and much friendly. Thanks to Dr. Yasser Qudaih for all the help and support he gave to me. My great appreciation to the DNST, PNST and SEIC fellows, Amgalanbat Batsuren, Ammarin Pimnoo, Bianca Szasz, Bui Nam Duong, Chen Shiyi Danny, Hala Almubarak, Kateryna Aheieva, Mohamed Alakli, Mohamed Nori Ait, Mohamed Yehia Edris, Pauline Faure, Tejumola Taiwo and Trinh Thang Long. Thanks to my Japanese colleagues for their fruitful cooperation during my study. My thanks to Shinya Kuramoto at NEC, Yuki Seri at Hitachi for building the geometric thermal model, Yuta Okumura at Hitachi for helping with the setup of radiation testing, and Dr. Masui Hirokazu for his guidance in the thermal vacuum and radiation tests.

My mother and father supported me with every possible way. They gave me more than I deserve, all over my life. I respect them, love them, and owe them my success and all what I achieve in my life. My lord, bestow on them your mercy as they did

bring me up when I was young. Thanks to my brother and sisters for supporting me and praying for me to finish this work. My deep thanks, appreciation, love and respect for my wife. She did a lot to support me to finish this work. She struggled as much as I did during my study. Sometimes we had difficult times as much stress was over there in my experiments and tests, but she was extremely smart to help us pass all of that smoothly and peacefully. Taking care of three children while I am spending sleepless nights in the lab was a real challenging task. Thank you very much. I dedicate this work to my children Yousof, Jody and Mahmoud, my nephew Yassin and nieces Basmalla and Iten. I wish when they grow up that they would follow the way of science and engineering, May ALLAH bless them all and grant them happiness, success and health.

Thanks to Japan and to Japanese people for the generosity to give me the chance to study here. I do respect and love Japan and Japanese people so much. I wish I can help my country to reach the same level of perfection.

Abstract

Developing small satellites for scientific and commercial purposes is emerging rapidly in the last decade. The future is still expected to carry more challenging services and designs to fulfill the growing needs for space based services. Nevertheless, there exists a big challenge in developing cost effective and highly efficient small satellites yet with accepted reliability and power consumption that is adequate to the mission capabilities. This challenge mandates the use of the recent developments in digital design techniques and technologies to strike the required balance between the four basic parameters: 1) Cost, 2) Performance, 3) Reliability and 4) Power consumption. This balance becomes even more stringent and harder to reach when the satellite mass reduces significantly. Mass reduction puts strict constraints on the power system in terms of the solar panels and the batteries. That fact creates the need to miniaturize the design of the subsystems as much as possible which can be viewed as the fifth parameter in the design balance dilemma.

At Kyuhsu Institute of Technology-Japan we are investigating the use of SRAM-based Field Programmable Gate Arrays (FPGA) in building: 1) High performance, 2) Low cost, 3) Moderate power consumption and 4) Highly reliable Muti-Processor System-on-Chip (MPSoC) On-Board Computers (OBC) for future space missions and applications. This research tries to investigate how commercial grade SRAM-based FPGAs would perform in space and how to mitigate them against the space environment. Our methodology to answer that question depended on following formal design procedure for the OBC according to the space environment requirements then qualifying the design through extensive testing. We developed the MPSoC OBC with 4 complete embedded processor systems. The Inter Processor Communication (IPC) takes place through hardware First-In-First-Out (FIFO) mailboxes. One processor acts as the system master controller which monitors the operation and controls the reset and restore of the system in case of faults and the other three processors form Triple Modular Redundancy (TMR) fault tolerance architecture with each other. We used Dynamic Partial Reconfiguration (DPR) in scrubbing the configuration memory frames and correcting the faults that might exist. The system is implemented using a Virtex-5 LX50 commercial grade FPGA from Xilinx. The research also qualifies the design in the ground-simulated space environment conditions. We tested the implemented MPSoC OBC in Thermal Vacuum Chambers (TVC) at the Center of Nano-Satellite Testing (CeNT) at Kyushu Institute of Technology. Also we irradiated the design with proton accelerated beam at 65 MeV with fluxes of 10e06 and 3e06 particle/cm^2/sec at the Takasaki Advanced Radiation Research Institute (TARRI).

The TVC test results showed that the FPGA design exceeded the limits of normal operation for the commercial grade package at about 105 C°. Therefore, we mitigated the package using: 1) heat sink, 2) dynamic temperature management through operating frequency reduction from 100 MHz to 50 MHz and 3) reconfiguration to reduce the number of working processors to 2 instead of 4 by replacing the space-redundancy TMR with time-redundancy TMR during the sunlight section of the orbit. The mitigation proved to be efficient and it even reduced the temperature from 105 C° to about 66 C° when the heat sink, frequency reduction, and reconfiguration techniques were used together.

The radiation and the fault injection tests showed that mitigating the FPGA configuration frames through scrubbing are efficient when Single Bit Upsets (SBU) are recorded. Multiple Bit Upsets (MBU) are not well mitigated using the scrubbing with Single Error Correction Double Error Detection (SECDED) technique and the FPGA needs to be totally reset and reloaded when MBUs are detected in its configuration frames. However, as MBUs occurrence in space is very seldom and rare compared to SBUs, we consider that SECDED scrubbing is very efficient in decreasing the soft error rate and increasing the reliability of having error-free bitstreams. The reliability was proven to be at 0.9999 when the scrubbing rate was continuous at a period of 7.1 msec between complete scans of the FPGA bitstream. In the proton radiation tests we managed to develop a new technique to estimate the static cross section using internal scrubbing only without using external monitoring, control and scrubbing device. Fault injection was used to estimate the dynamic cross section in a cost effective alternative for estimating it through radiation test.

The research proved through detailed testing that the 65 nm commercial grade SRAM-based FPGA can be used in future space missions. The MPSoC OBC design achieved an adequate balance between the performance, power, mass, and reliability requirements. Extensive testing and applying carefully crafted mitigation techniques were the key points to verify and validate the MPSoC OBC design. In-orbit validation through a scientific demonstration mission would be the next step for the future research.

Table of Contents

List of Figures

List of Tables

Acronyms

ABFT	Architecture Based Fault Tolerance
AES	Advanced Encryption Standard
AMFT	Adaptive Middleware for Fault Tolerance
ASIC	Application Specific Integrated Circuits
AU	Astronomical Unit
BB	Black Box
BGA	Ball Grid Array
BPI	Byte Peripheral Interface
BRAM	Block Random Access Memory
BSP	Board Support Package
BSTI	Basic Space Technology Iniative
CAN	Controller Area Network
CAD	Computer Aided Design
CeNT	Center for Nano satellite Testing
CMOS	Complementary Metal- Oxide- Semiconductor
CFC	Clock Frequency Control
Cf-252	Californium-252
CLB	Configuration Logic Block
COTS	Commercial Off The Shelf
CRC	Cyclic Redundancy Check
DAQ	Data Acquisition
DCM	Digital Clock Manager
DDD	Displacement Damage Dose
DIO	Digital Input Output
DNST	Doctorate in Nano-Satellite Technologies
DPR	Dynamic Partial Reconfiguration
DTM	Dynamic Temperature Management

DUT	Device or Design Under Test
ECC	Error Correcting Code
EDAC	Error Detection And Correction
EDC	Error Detection Coding
EDK	Embedded Development Kit
ESA	European Space Agency
FAR	Frame Address Register
FDR	Frame Data Register
FIT	Failure In Time
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
HPEC	High Performance Embedded Computing
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
IDE	Integrated Development Environment
IIC	Inter-Integrated Circuit
IO	Input/output
IOB	Input Output Block
IP	Intellectual Property soft cores
IR	Infrared Radiation
ISR	Interrupt Service Routine
JAEA	Japan Atomic Energy Agency
JTAG	Joint Test Action Group
KURRI	Kyoto University Research Reactor Institute
KIT	Kyushu Institute of Technology
LMB	Local Memory Bus
LEO	Low Earth Orbit
LET	Linear Energy Transfer
LUT	Look Up Table
MBU	Multiple Bit Upset

MDM	MicroBlaze Debug Module
MOSFET	Metal-Oxide Semiconductor Field-Effect Transistor
MPI	Message Passing Interface
MPSoC	Multi-Processor System-on-Chip
MRAM	Magneto-resistive Random Access Memory
MSCP	Master System Control Processor
MTBF	Mean Time Between Failures
NASA	National Aeronautics and Space Administration
NETS	Nano-satellite Environment Test Standardization
NVP	N Version Programming
OBC	On-Board Computer
PCB	Printed Circuit Board
PLB	Processor Local Bus
PNST	Post-graduate study on Nano-Satellite Technologies
RAM	Random Access Memory
RAAN	Right Ascension of Ascending Node
RB	Recovery Block
RISC	Reduced Instruction Set Architecture
RM	Reconfigurable Module
RP	Reconfigurable Partition
RS232	Recommended Standard 232
RS422	Recommended Standard 422
SAA	South Atlantic Anomaly
SBU	Single Bit Upset
SDK	Software Development Kit
SECDED	Single Error Correction Double Error Detection
SEB	Single Event Burnout
SEE	Single Event Effect
SEFI	Single Event Function Interrupt
SEGR	Single Event Gate Rupture
SEL	Single Event Latch-up

SET	Single Event Transient
SEnT	Space Environment Testing
SEU	Single Event Upset
SFT	Software Fault Tolerance
SIHFT	Software Implemented Hardware Fault Tolerance
SMC	Space and Missile systems Center standard
SoC	System on Chip
SOI	Silicon On Insulator
SPI	Serial Peripheral Interface
SPSoC	Single Processor System-on-Chip
SRAM	Static Read Access Memory
TARRI	Takasaki Advanced Radiation Research Institute - (JAEA)
TID	Total Ionizing Dose
TMM	Thermal Mathematical Model
TO	Triple Oxide
TMR	Triple Modular Redundancy
TRL	Technology Readiness Level
TT&C	Telemetry, Tracking and Command
UART	Universal Asynchronous Receiver and Transmitter
UNOOSA	United Nations Office for Outer Space Affairs
USB	Universal Serial Bus
WDT	Watch Dog Timer
XPS	Xilinx Platform Studio

Chapter 1 : Introduction

More than 50 years passed since the launching of the first satellite into space [1]. Sputnik-1 carried a simple communication system which was used to send a beep to its ground station. The launching of the first artificial satellite marked the start of a new age. Humans started looking upward to the open skies and forward to the unlimited opportunities form them. Since that event the space age and the embarkation of the space technology had profoundly progressed and advanced concepts as well as applications were introduced [2] [3].

At the early beginnings, space was mostly a scope for contribution by a small number of countries. The technology of building satellites and launchers was a sort of a national security issue. Only several countries were able to fully build, launch and operate their own satellites. The space technology has always used to be an expensive and complicated field. Even after more than 50 years of starting the space age, we still cannot find so many countries that can afford the cost and/or the knowledge for starting a real full-cycle, from launch to operation, space industry [3].

The space race led to unleashing different applications and services that can be provided from the space-borne devices. Satellites are just one form of the devices that can be launched to space. Other devices include space probes, telescopes, and even manned space stations [4]. The applications, scientific achievements, and spin-offs that emerged from using space-borne devices and stations were tremendous and well sounded [5] [6]. Nowadays, the world enjoys real time telecommunication across the globe, earth observation constellations for disaster monitoring and resources exploration, homeland security services, terrestrial and flight navigation, high speed satellite-based internet links and media broadcasting services. These applications were used to be provided by large scale, heavy and high power satellites. However, advances in microelectronics technologies and material engineering enabled the developing of smaller and lighter satellites to provide the same services with even better efficiencies [7] [8] [9] [10]. Even the reuse of technologies developed for other space activities, such as human spaceflight systems, was investigated to develop cost effective small satellites [11].

The advancement in building satellites was associated with a paradigm shift. It was a fact that till the end of the nineties of the previous millennium, satellites and space-borne devices were used to be developed by space agencies and organizations. The facilities needed in developing, assembling, integrating and testing the space-borne devices had to meet sophisticated requirements. The technical knowledge, as well, was not available outside of these organizations. However, these conditions prevailed until the launching of university-made satellites in the eighties, such as the UoSAT-1 in 1981 [12]. Since then, a new start was remarked through a paradigm shift in the traditional concepts used in building satellites by universities.

The key point that enabled the universities to indulge in the space activities was the use of Commercial-Of-The-Shelf (COTS) components. COTS enabled building low cost satellites while maintaining reliability through design techniques. Nowadays, there are multiples of satellites that were successfully developed by universities and are already rotating in space. Universities helped in developing new techniques to shrink the satellites sizes and make them much more efficient.

International organizations and space agencies became interested in what universities were doing and offered to sponsor their programs. A remarkable example is the Basic Space Technology Initiative (BSTI) that was initiated by the United Nations Office for Outer Space Affairs (UNOOSA). The BSTI is focusing on providing space engineering education especially for developing countries which cannot afford the costs of such an expensive industry. It has a goal of defining a space engineering curriculum that can be used as a guide by academic institutes all over the world to provide standardized space engineering degrees. The BSTI also inaugurated, in collaboration with Kyushu Institute of Technology at Japan, a space education program named the Doctorate in Nano-Satellite Technologies (DNST). The program aims at hosting students from the developing countries and granting them the doctorate degree upon pursuing research in the space engineering field. The program was later promoted to handle master students as well and its name was changed to the Post-graduate study on Nano-Satellite Technologies (PNST). By the intake of 2013, 9 participants from 8 countries are already attending the program and more students would participate in the coming years as it gains more international recognition.

My research is part of the DNST program. I belong to the first batch of the students who were accepted in 2010. In the next few sections, I would discuss the outline of the research and its expected impacts. It might be necessary to point out that the major purpose of this research is to develop and qualify a low cost, high performance, high reliability, and low power on-board computer for in-orbit processing of scientific data and commands. This on-board computer would be used in developing state-of-art small spacecraft technology for future space missions by using Field Programmable Gate Arrays (FPGAs). It would start a new era in developing dynamically reconfigurable space systems to maintain the power requirements and adapt to the space environment conditions.

The research was conducted at the premises of the Kyushu Institute of Technology. Testing was conducted at CeNT, Takasaki Advanced Radiation Research Institute (TARRI) – Japan Atomic Energy Agency (JAEA), and Kyoto University Research Reactor Institute (KURRI). The results of the research recommend the developed on-board computer for in-orbit validation in a technology demonstration mission.

1.1 Research Outline

The challenge in developing small satellites is mainly concerned with keeping the satellite size and mass as minimum as possible. We should take into consideration developing satellites with an area-to-mass (A/M) ratio that would enable re-entrance to the earth atmosphere in less than 25 years. The area-to-mass ratio increment enables more drag force and hence better satellite deorbiting to reduce the problem of space debris. However, minimizing the subsystems mass, area, size and power consumption needs innovative technologies to be applied when developing the satellite subsystems. The density of the material used in developing the subsystem structure as well as the subsystem components have to be carefully selected. The reduction in the subsystem mass would affect the whole satellite. Another aspect is to reduce the power consumed by each of the satellite subsystems to reduce the surface area, and hence the mass, of the solar panels and the sizing of the batteries. This requires using innovative semiconductor manufacturing technologies and lower transistor feature sizes for the electronic components to reduce their static and dynamic power consumptions.

On the other hand reliability is a big concern for all satellite developers. Design techniques as well as subsystems' components should be carefully selected in order to develop high reliability satellites that can serve for the required mission lifetime. The design techniques usually depend on redundancies whether cold or hot backups. The components are better to be selected from the space approved category that has flown on-board previous space missions. The space approved category might include space grade and non-space grade components. Space grade components are usually of low performance compared to COTS components. Depending on the mission requirements, the reliability threshold through the mission lifetime is set. Using redundancies in hot backup regime requires higher power consumption. Therefore, it is always important to design the reliability carefully to match with the power budget as well as the mass budget and size of the satellite.

We need to strike a balance between the reliability, power consumption, mass, size and performance of the space mission. The required balance needs innovative design solutions in order not to sacrifice any of the aforementioned parameters. To achieve this balance, design of digital electronic circuits should make use of three aspects: 1) Advancement in the semiconductor industry through the reduced transistor feature size, 2) Re-programmability of digital designs, 3) On-the-Fly DPR.

Moore's law suggested that the number of transistors in a chip would double every two years. The duplication of the transistors count means that their feature size would reduce in almost the same proportion. Nowadays we have FPGAs and Application Specific Integrated Circuits (ASIC) at the 28 nm scale. Chip manufacturers are working on 7 nm designs. This nano-meter scale enables high density designs to be developed. However, it sets a challenge to reduce the power consumption and the radiation immunity of the overall design. Miniaturization of satellite subsystems should be achieved through using the available nano-meter scale feature size technology. Power consumption, both dynamic and static, are reduced through applying manufacturing and design techniques. Immunity to space radiation is enhanced through fault tolerance and shielding techniques. Reducing the power consumption would reduce the operating temperature of the electronic chips which would enable them to withstand higher thermal conditions while operating in space.

Digital designs of satellite subsystems are usually implemented using components that cannot be changed after launching the satellites. The designs are fixed and whether they contain a fault or not should be fully investigated before launching. Some designs might contain a mistake that is not apparent during ground tests and might cause mission failures. Correcting the design mistakes of digital designs in-orbit used to be an impossible task. The most effort that can be done is to reprogram the code in case an unwanted software performance was noticed. This can be done only if the satellite has the feature to reprogram its code. A major advancement in digital designs that appeared in the last decade is to reprogram the hardware. Reprogramming the hardware can be either by loading alternative design modules from an on-board local memory or uploading new designs from the ground stations. Design diversity concept can thus be achieved in a much smaller area, size and mass. However, the re-programmability feature is applicable only through using Static Random Access Memory (SRAM)-based FPGAs. This type of the FPGAs has its own merits and demerits. The major merit is providing re-programmability. While the major demerit is susceptibility to design alteration while operating in space due to Single Event Upset (SEU) induced errors from the space environment radiation.

DPR is a new technique that is introduced by SRAM-based FPGA manufacturers to overcome the problem of switching off the logic whenever a reconfiguration of the FPGA is needed. The technique depends on making part of the logic as static while defining Reconfigurable Partitions (RP) that can hold different Reconfiguration Modules (RM). Whenever a reconfiguration is needed for one of the RPs with any of its associated RMs, the static logic would reload the RP from the external flash memory that holds the RMs hardware designs in the form of partial bitstreams. This technique enables loading and unloading of digital designs when needed thus it can be used to save the power consumption of the satellite. As a matter of fact, not all of the satellite subsystems need all of their components for 100% of the operation time. Some components might be switched off until they are needed to perform a specific function. Switching off using the DPR technique means simply to load a Black Box (BB) design in the RP.

DPR can also be used in developing adaptive subsystems. Adaptation can take place in response to the changes in the space environment, power and thermal operating conditions. For example loading extra redundant modules could be done to provide higher reliability in case of increased induced soft errors due to radiation. Also, unloading some of the design modules when they are not needed in the current operations might be done to save the power consumption especially if the batteries are on the critical storage limits. The thermal conditions in the sunlight and eclipse regions of the orbit vary drastically. Some modules might be unloaded to reduce the heat emissions while in the sunlight region. Heavy processing designs might defer their processing to the eclipse region by loading the processing modules there. Thus heat emissions might not introduce a problem to the operating temperature of the FPGA and other components in the satellite. This technique provides an adaptive protection through partial reconfiguration. Another approach is to use the DPR to correct the errors in the bitstream frames of the hardware in the FPGA configuration memory. As the satellite operates in space its SRAM-based FPGAs are subjected to soft errors induced through charge accumulation at the memory cells (circuit) nodes. The charge accumulation, if it crosses a certain limit, will flip the logic values stored in the internal SRAM cells of the FPGA which is known as the local configuration memory. The internal SRAM holds the design in the form of binary bitstream and if any bit is corrupted then that might lead to serious design failure issues. Therefore DPR can be used to continuously correct the induced flips through scrubbing in a read-correct-write back process.

If we consider the three illustrated aspects mentioned above, we would recommend SRAM-based FPGAs as a suitable candidate to achieve the required balance between the satellite reliability, power consumption, mass, size and performance. However there still another major issue which is the cost. If we consider space grade SRAM-based FPGAs then it would be very expensive to develop satellites at moderate budgets up to 5 M USD. However, if we use commercial grade then the cost would be reduced ultimately. In fact the price of a space grade component is about 100 times the price of the commercial grade one. Reducing the cost is a great benefit but there is not enough data about the suitability of these components when used in space. Especially if the components are of latest generations which did not fly yet on many space missions thus not enough information about their performance in space is available. Therefore, we need to provide a qualification method for such components that would reveal their suitability and provide sufficient data as well as establishes the mitigation techniques to make them fitting for space missions.

The main theme of the research is to investigate the suitability of using COTS SRAM FPGAs of the 65 nm scale in developing miniaturized High Performance Embedded Computers (HPECs). The direct application of the research is to develop an OBC suitable for small and nano-satellites. Qualification is an essential part of the research as we need to highlight the mitigation techniques needed to support COTS SRAM-based FPGAs while operating in space. In the research we investigated the concept of fault injection to simulate space radiation. Radiation testing and mitigation against soft error through DPR as well as thermal vacuum testing and mitigation through passive and active techniques were thoroughly studied. The research focuses on presenting practical achievements besides theoretical bases. Studying mitigation techniques for space-based applications needs real developments and qualifications to assess their suitability. Therefore, Space Environment Testing (SEnT) was a fundamental constituent in the researched topics. We focused on the development followed by real qualification through SEnT. Qualification is the most crucial aspect in developing working space systems and proving the validity of new concepts and technologies. This research is trying to start a new epoch in using SRAM-based FPGAs to strike the needed balance between cost, reliability, power consumption and performance in addition to flexibility in reprogramming the whole system. Some designs might require ultimate reduction in its power consumption hence reducing its total thermal power while not sacrificing its reliability. That feature was thoroughly studied in the research by investigating the time and space redundancy trade-offs when developing MPSoC and Single Processor System-on-Chip (SPSoC) using 65 nm FPGAs.

Usually satellites cannot be restored after launching. However, in some rare cases, it might be necessary to restore a satellite from its orbit for repair purposes. Nevertheless, such missions are very specific, expensive and not regular and are only performed for very peculiar reasons such as repairing a military satellite or a space telescope. Therefore, as a general rule, satellites should be fully qualified in a thorough approach as much as possible. SEnT is the real manifestation of suitability of designs to operate in space. SEnT is the main feature and differentiator of this research from other designs which are solely validated and approved through simulations.

1.2 Research Motivation

Developing space qualified low cost satellites while maintaining high performance and reliability is the main motivation of this research. Both the academia and industry are welling to achieve good results on this way. Universities and research institutes are developing new missions to help better understanding the space environment and its possible effects on space missions whether manned or unmanned. Innovative profit generating missions are being offered by space companies around the world. Developers of small satellites are struggling to make the systems smaller and better. In addition to that, some countries, especially developing countries, are welling to start making use of space for the benefit of their welfare. We wish to provide accessibility to these countries through low cost and clean technology. As miniaturizing the satellite subsystems would reduce the required number of components to be manufactured and hence reduce the emissions that accompany any manufacturing process. This means we are trying to provide access to space while supporting the concept of green technology on Earth.

Through this research we are providing a platform for In-orbit processing of the huge amounts of scientific data which is collected on-board. This would ultimately reduce the communication link bandwidths and required power on-board the satellites. This powerful processing platform is a true demonstration of the HPEC concept using COTS in space. It would enable future mission to develop further applications using the concept of evolutionary hardware. The hardware that can cure and repair itself from malfunctions while in operation in its orbit. That concept would have multitude of applications in space as well as in other fields such as robotics and medical equipment. Giving the satellites high processing capabilities while being protected against the space environment would certainly boost the research towards developing silicon brains that would enable building autonomous satellites that can take full control on the mission with minimum or no human interaction.

1.3 Research Objectives

In designing devices that would operate in space, we usually have to design the devices from one of two perspectives: 1) Either we design the device to avoid the space environment induced effects, or 2) we design the device to tolerate the space environment induced effects. The difference between avoidance and tolerance is the type of the used components and their manufacturing grade. Space grade components provide good protection against radiation induced Single Event Effects (SEE), Total Ionizing Doses (TID), and Displacement Damage Doses (DDD). Also their operating temperature range can withstand the thermal vacuum conditions in space. However, our research mainly aims at providing protection to COTS components through applying design mitigation techniques instead of using space grade ones.

The following points would summarize the main objectives of the research:

- Design a well-protected platform that would be used as a HPEC for OBC applications.
- Fully develop the system and qualify its operation and mitigation techniques in SEnT conditions.
- Use the state-of-art low cost reprogrammable COTS SRAM-based FPGAs to provide maximum performance and flexibility.
- Thorough investigation of the best techniques to mitigate the system against radiation induced transient faults as well as provisioned permanent failures.
- Thorough investigation of the best techniques to mitigate the system against the thermal vacuum conditions.
- Start a new epoch in using 65 nm COTS SRAM-based FPGAs, instead of the expensive anti-fused and space grade variants, in developing reliable high performance space devices and subsystems through mitigating all possible space environment effects on their operation.
- Make use of the compactness and re-programmability features of SRAM-based FPGAs in developing miniaturized and flexible systems as well as facilitate the way to develop future satellite-on-chips.
- Investigate the best fault tolerant approach from time and space redundancies to be applied on 65 nm COTS SRAM-based FPGAs.
- Fully characterize the suitability of using 65 nm COTS SRAM-based FPGAs in space applications by revealing the SEnT data.
- Investigate the different methods in performing SEnT and the pros and cons of each method in order to provide testing guidelines and procedures to efficiently reduce the test costs and complexity while maintaining the test quality i.e. radiation test with Californium-252 or accelerated proton beam or fault injection.

1.4 Research Methodology

The applied methodology started by thoroughly investigating and studying the theoretical concepts of fault tolerance and modern digital design approaches. The theoretical study was followed by real development and qualification through performing SEnT on the system. Qualification through SEnT is considered as a mandatory step to prove that the design can really work in space. SEnT results were used to practically demonstrate the design operability and suitability for space missions as well as the applicability of the design concepts.

FPGAs provide the capabilities to develop high speed digital designs. Nowadays, computer design packages can be used in designing complete multi-processor embedded systems inside FPGAs. Simulating functional tests as well as developing software is also available through hardware/software integrated simulators and software development environments. The complete MPSoC design and simulation was performed using Computer Aided Design (CAD) tools.

The research methodology proceeded according to the following steps:

1) Study of the different time and space redundancy approaches in fault tolerance design.
2) Study of hardware fault tolerance mitigation techniques for SRAM-based FPGAs.
3) Developing the mitigated hardware of a SPSoC to demonstrate time redundancy.
4) Developing the mitigated hardware of an MPSoC to demonstrate space redundancy.
5) Developing the software application that would be used in running the hardware during the tests as well as managing the fault tolerant features in the designs.
6) Comparison between both the SPSoC and the MPSoC to select the best fault tolerance approach based on power consumption, computation throughput, area utilization, reliability and heat emission.
7) Performing radiation tests under proton accelerated beam at the nuclear reactor to estimate the static cross section.
8) Specifying a target operation orbit for the system and using that orbit to estimate the charged particles energy spectrum and flux of the protons, electrons and

heavy ions. The SEU rate would be estimated using the static cross section and the charged particles spectrum through space radiation models.

9) Developing a fault injection bench to flexibly estimate the dynamic cross section of the system at any time and under any changes of the design without having to repeat the expensive radiation shots.
10) Performing fault injection tests to assess the mitigation against radiation induced soft errors.
11) Analysis of the system reliability in accordance to radiation induced soft errors.
12) Studying thermal vacuum mitigation techniques through active control via Dynamic Temperature Management (DTM) as well as passive control through heat sinks.
13) Performing thermal vacuum analysis of the system and running thermal vacuum tests with and without the mitigation techniques to assess their suitability.
14) Generalizing the research results to be used as concepts in different design approaches.
15) Results dissemination.

1.5 Research Originality and Contributions

The following points summarize the major research contributions and originalities:

1) Developing and qualifying a reprogrammable MPSoC using 4 embedded systems in an SRAM-based FPGA with up to 400 MIPS and power consumption of max 5 Watt, soft error reliability 99.99%, mass about 200 gm and ultra-low cost (<2000 USD). The OBC cost can be considered as about 10 % of the total mission cost as estimated in [13]. Therefore our cost reduction would contribute to about 10% of the total budget. However, if our design is used in the payload computer system then we would contribute to more reduction in the total budget as the payload forms about 21% of the mission cost. The more use of our design in other system bus subsystems, the more cost reduction we would achieve compared to the use of the space approved subsystems.
2) Using a novel technique for estimating the static cross section during radiation tests through internal scrubbing of the FPGA configuration memory instead of using the traditional external monitoring and configuration module.

3) Developing a fault injection bench for flexibility of estimating dynamic cross section in the lab through fault injections to the configuration memory bitstream.
4) Applying thermal mitigation through DTM of FPGA operating temperature through controlling the operating frequency and reconfiguring the operating hardware.
5) Clarifying the procedural details of space environment testing, radiation and thermal-vacuum, and publishing the results for the class of 65 nm SRAM-based FPGAs.
6) Clarifying the analytical details of reliability estimation procedure for induced soft errors based on the static cross section and target orbit.
7) Comparative study between suitability of time and space redundancies in developing embedded processor systems on 65 nm FPGAs.
8) One of the research publications was cited at NASA’s Small Spacecraft Technology. State of Art Report – 2014 [14].

1.6 Thesis Overview

Chapter 1 is the introduction where the research objectives and detailed problem definition are investigated as well as presenting the research contributions and originalities.

Chapter 2 presents the background related to the space environment and possible mitigation techniques of the radiation effects. It overviews the FPGA technology and refers to detailed textbooks and references for the interested reader to grasp the full detailed picture from the rich available literature. It covers the previous work as much as possible, although not exhaustive, to present the trends in developing fault tolerant and new concept OBCs.

Chapter 3 presents the details of the MPSoC design and operation concept as well as the fault tolerance features applied for mitigation against the induced soft errors. It discusses the operation of the different blocks and the concept of network-on-chip communication through mailboxes to achieve cross-voting through the score-card concept.

Chapter 4 presents the comparison between space and time redundancy in 65 nm SRAM-based FPGAs to select the best technique in terms of power consumption, resources utilization, performance and reliability. The analytical approach for estimating the soft error reliability due to induced SEUs bit flips in the configuration bitstream of the FPGA is described.

Chapter 5 presents the experimental radiation test procedure at the Takasaki nuclear reactor. The analytical estimation of the system reliability based on the configuration memory scrubbing rate and the expected SEU rates is presented.

Chapter 6 presents the experimental thermal vacuum test procedure at CeNT. The thermal analysis model and the test procedure as well as the mitigation techniques are presented.

Chapter 7 presents the results of evaluating the design performance and discusses the research contribution in developing a well mitigated MPSoC based on the results from the SEnT.

Chapter 8 presents the conclusion and future work. It discusses the idea of satellite-on-chip and its feasibility for developing future high performance and ultra-compact space missions. We call that concept the ultra-space in a reference to currently available ultra-low-size computers.

Chapter 2 : Background and Literature Review

Satellites are subjected to the harsh space environment conditions in their orbits. In this chapter we provide an overview about the radiation threats which affect the satellite electronics including the OBC. We focus on the radiation threats as they are of a probabilistic nature and a system cannot be 100% guaranteed to be radiation-hardened unless it would have been designed using space-grade components which provide radiation hardening by manufacturing. On contrary to radiation threats, thermal threats are controllable and are well determined so that their effects can be fully mitigated and verified in the design and test phases before the actual in-orbit operation. We illustrate the major functions of the OBC within the satellite to provide a sufficient background about the nature of the tasks which it handles. Fault mitigation is discussed to understand the different approaches for developing a design that can survive radiation-induced faults. As SRAM-based FPGA is used in developing the MPSoC-OBC, a simple background about its building blocks and logic components are illustrated. Previous work for developing fault tolerant OBC designs using SRAM-based FPGAs are presented. The state-of-art for small spacecraft’s OBCs in comparison to our design is discussed.

2.1 Space Environment

The space environment is mainly characterized by its specific nature of radiation, plasma, meteoroids and space debris, atmosphere, solar, geomagnetic and thermal conditions in vacuum [15] [16]. The charged particles in the space environment form a major threat for the correct operation of spacecraft electronics. Radiation testing of electronic components is performed to characterize the space radiation effects [17]. The sources of natural radiation in space are summarized in Table 2 -1 r from [18].

Table 2 -1 Main Sources of the Natural Space Radiation Environment [18]

Radiation Belts	Electrons	1 eV – 10 MeV
	Protons	1 KeV – 500 MeV
Solar Flares	Protons	1 KeV – 500 MeV
	Ions	1 to a few 10 MeV/n
Galactic Cosmic Rays	Protons and Ions	Max flux at about 300 MeV/n

The different types of hazards resulting from the space environment elements are shown in Figure 2-1 from the European Space Agency (ESA).

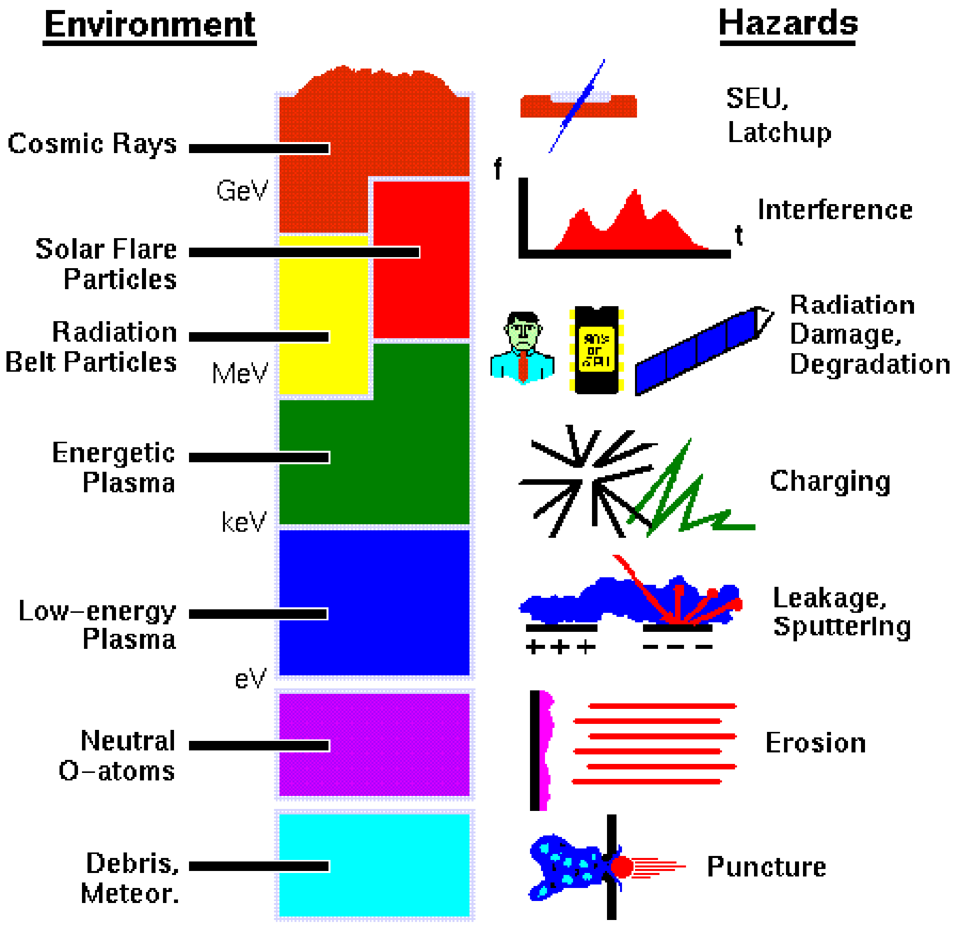


Figure 2-1. Hazards in the Space Environment. Source: European Space Agency, Space Environment and Effects Analysis Section © ESA.

Radiation has three effects on electronic components: 1) Displacement Damage Dose (DDD), 2) Total Ionizing Dose (TID) and 3) Single Event Effects (SEE). DDD and TID are accumulated effects while SEE is caused by single particles. Radiation effects on electronic components are well presented in the literature [19] [20].

DDD is an accumulative radiation effect that causes the change of the semiconductor atomic lattice due to the displacement of atoms from their locations as a result of a strike by an external charged particle such as heavy ions. It is mostly seen in the degradation of the satellite solar panels [21].

TID is another accumulative radiation effect which causes the collection of ions in the oxide of the MOSFET device leading to changes in the threshold voltage at which the device is switched on/off from its gate. Over the time the device fails to switch and it would have a permanent on or permanent off failure depending on its type as P-MOSFET or N-MOSFET [21].

SEE is characterized as an effect that is caused by a single charged particle. It has some destructive forms such as Single Event Gate Rupture (SEGR), Single Event Burnout (SEB) and Single Event Latchup (SEL) and non-destructive forms such as SEU, Single Event Transient (SET) and Single Event Function Interrupt (SEFI) [22]. Destructive and non-destructive radiation testing and the needed setups, procedures and guideline to test data interpretations are given in [22]. An interesting correspondence between the radiation sources and the types of effects they induce on electronic components is shown in Figure 2-2 referenced from [18] and was initially introduced in [23].

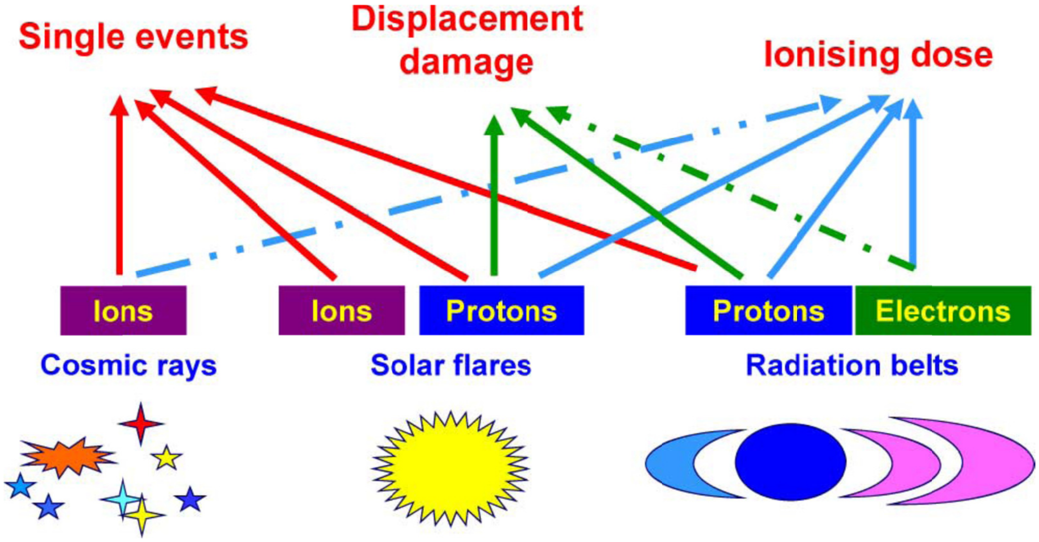


Figure 2-2. Correspondence between Radiation Sources and Effects on Components [18] [23] © 2013 IEEE.

According to Ecoffet in [18] [23], the perecntages of spacecraft anomalies due to the space enviromnet can be illustrated in Figure 2-3. 45% of the anomolies are due to radiation effects with 80% due to SEUs.

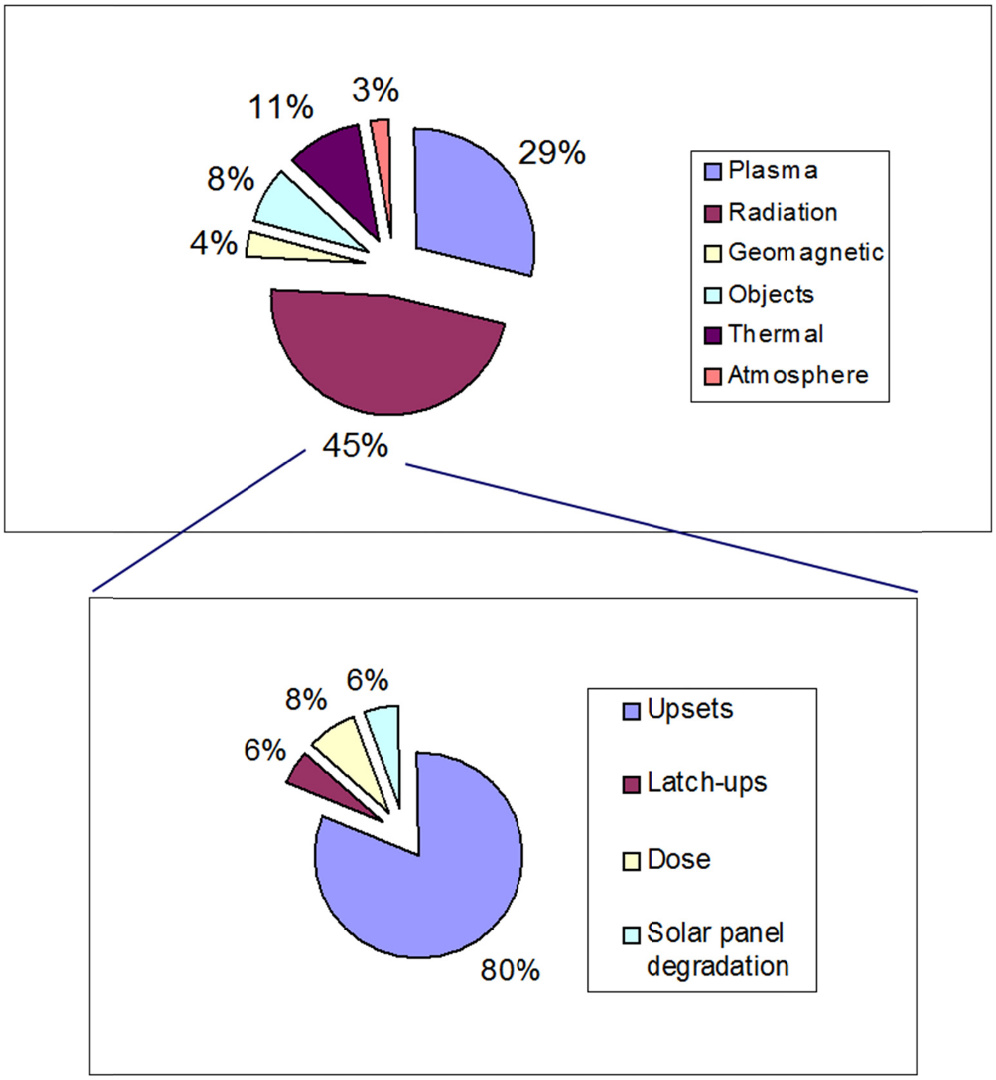


Figure 2-3. Percentages of Spacecraft Anomalies due to the Space Environment [18] [23] © 2013 IEEE.

Earth magnetic field provides a protection against charged particles in space. However, there are two regions of high radiation: 1) the polar cusps, and 2) the South Atlantic Anomaly (SAA). Satellites passing over these locations suffer higher events of SEE [24]. The trapped protons and electrons in the radiation belts can move in spiral motion along the magnetic field lines or bounce back and forth or drift along the equator as shown in Figure 2-5 referenced from ESA. An illustration of Earth's magnetosphere and radiation belts is shown in Figure 2-4 referenced from the NASA Marshall Space Flight Center.

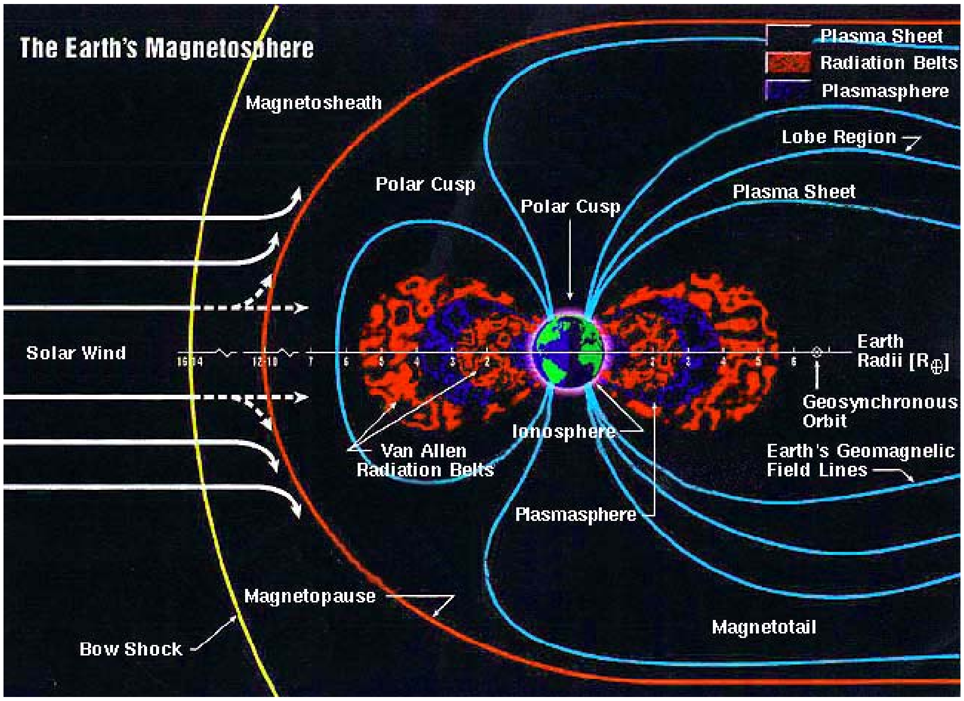


Figure 2-4. Earth Magentosphere and Radiation Belts.Source: Space Environments & Effects Program, NASA Marshall Space Flight Center © NASA.

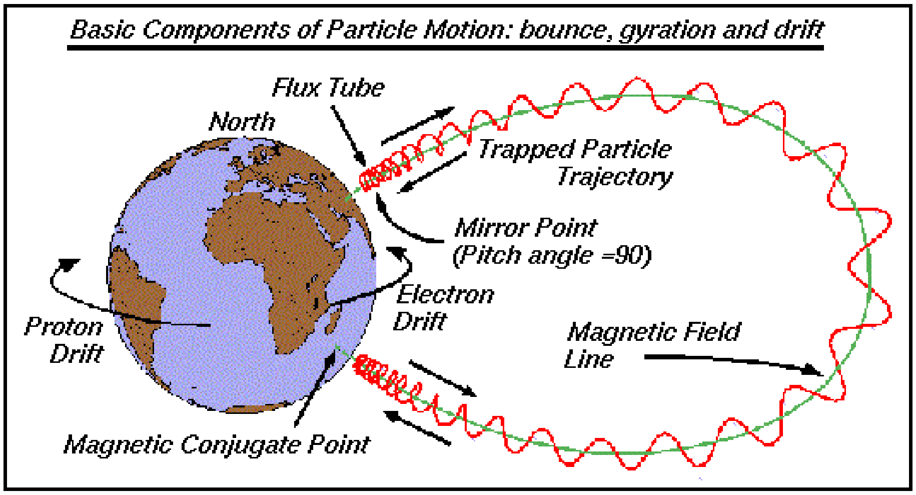


Figure 2-5. Motion of the Trapped Charged Particles. Source: European Space Agency, Space Environments and effects analysis section © ESA.

2.2 Satellite On Board Computers

Satellites are launched to carry out specific mission(s). The mission(s) is usually called the satellite payload. All satellites have a fixed set of subsystems that must exist for the satellite to operate. The power subsystem is responsible for power generation, storage, regulation and distribution. The communication subsystem is responsible for establishing a communication link in between the satellite and the ground station when the satellite is in the visibility region of the ground station. It receives the commands uploaded from the ground station and downloads the scientific data and telemetry information to it. The attitude orientation and control subsystem is responsible for maintaining the satellite orientation in reference to an inertial axis within specified margins. It handles the maneuvers which the satellite performs to orient itself to a specific position. The navigation unit is responsible for providing accurate navigation data about the satellite position in space. It can provide time steps to synchronize the satellite real time clock. The OBC is the subsystem which is responsible for receiving the commands from the communication subsystem, sorting the commands according to their execution times, switching on/off the subsystems and missions, sending the commands to the relevant subsystems and missions, receiving the telemetry from the subsystems and missions and managing the record of the scientific data and telemetry data in storage memories.

The OBC is the satellite controller which synchronizes the operation of the whole satellite. Figure 2-6 shows the typical OBC architecture. It consists of a microprocessor based system which interfaces to a program memory and storage memory. The storage memory should be of a non-volatile type with low access time. Thus the read and write access times should be low to reduce the delay in reading and writing data in order to have a match between the data generation and play back rates. Data generation comes from the scientific mission and data playback goes to the communication subsystem. The commands sent from the ground station are usually encoded in packet streams and frames. The commands encoding facilitates the retransmission in case part of the command stream is lost during transmission. The OBC receives the decoded command stream from the communication subsystem. In some architectures, a specific subsystem handles the functions of telemetry storage and encoding as well as tracking of the microwave communication signal and decoding of the received commands, it is usually known as the Telemetry, Tracking and Command subsystem (TT&C).

The OBC receives the mission data from the payload of the satellite to store it in its temporary storage memory for later playback to the communication subsystem during the communication session. It might also be possible to directly send the data from the payload to the communication subsystem without storing it in the OBC. The OBC interfaces with the satellite subsystems through the Input/Output (I/O) unit. The I/O unit transmits the interface commands and data arrays from the OBC to the satellite subsystems and receives their missions' data and or telemetry status. It handles the interface of each subsystem to match it with the OBC bus. For example it can handle the IIC interface from one side with the subsystems and connect to the parallel local bus of the OBC on the other side. Sending the commands to the satellite subsystems might be serialized over a line such as current loop interfaces. Also the switch on/off pulses to start and stop the different subsystems and missions can be done through on-board solid state relays. For these specific functions a command issuing unit is needed.

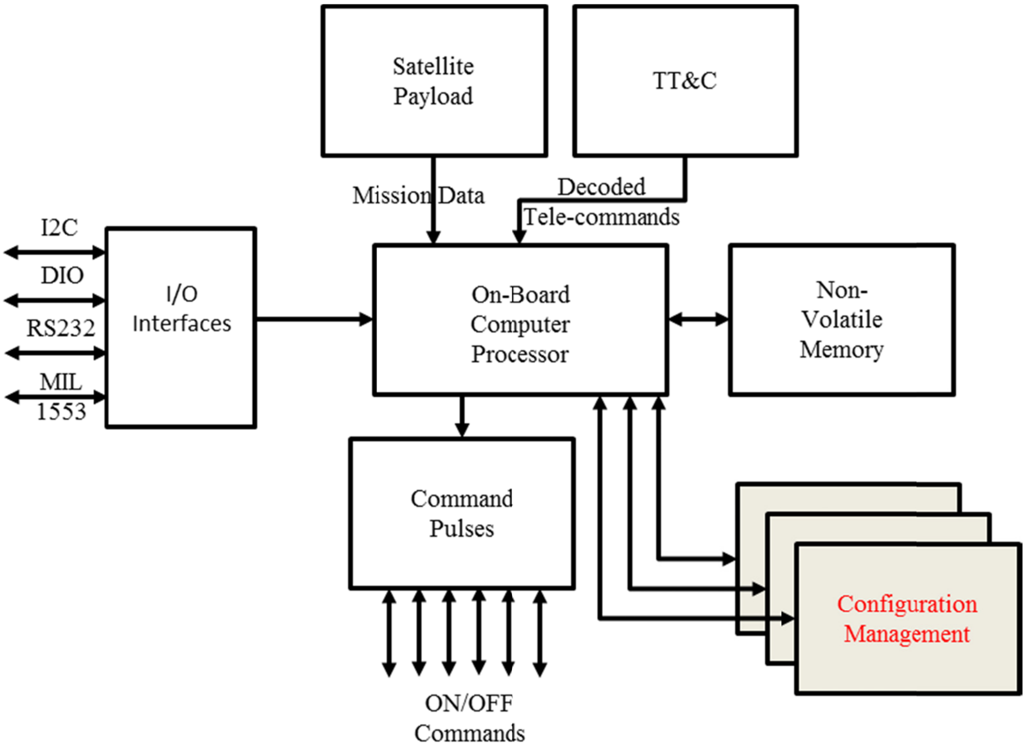


Figure 2-6. Typical OBC Block Diagram.

When building satellites, reliability is the big concern. Usually a redundancy of the units and subsystems is used. There is a specific unit in the satellite that manages the transfer between the redundant components upon detection of failures and malfunctions. It should be of high reliability and it is preferred to apply TMR on its architecture. This unit is called the configuration and failure management unit. It handles the swapping between main and reserve blocks. It can also handle the operation of the satellite when its OBC fails. It acts as the upper level control logic in emergency cases. In the next section we illustrate the major fault tolerance approaches that might be used in developing OBCs.

2.3 Radiation-Induced Fault Mitigation Approaches

Fault mitigation can be achieved either through fault avoidance or fault tolerance [25]. Fault avoidance is mainly through radiation hardening by manufacturing while fault tolerance is through radiation hardening by design [26]. Fault tolerance in its wide definition means the availability of techniques to mask the faults whenever they happen so that the system keeps on working even when faults exist. The techniques that can be used mainly focuses on adding redundancy to the system in space, time and data [27]. Space redundancy depends on adding additional computation blocks that provide redundant operation to the main working set. A consensus might be formed among the results or a switch to a backup function might be done if the main block failed to produce valid data. For time redundancy, execution is performed for the first round and the result is tested for validity. In case the result is not accepted, as it is not valid, execution repetition would take place and results are recalculated until a valid result is obtained or a system error signal takes place. Time redundancy can be executed even if the result is valid to make sure that the test module itself is working correctly. The data redundancy depends on adding additional bits known as code-words to the data to facilitate the detection of faults whenever they exist.

Usually data faults take place in the memory contents and processor registers. They might be of no effect or have severe system defects. Many techniques were developed based on the taxonomy of fault tolerance into space, time and data. Implementation of fault tolerance techniques can take place in hardware, software or both. There exists a group of techniques known as Software Implemented Hardware Fault Tolerance (SIHFT) that handles the tolerance of software to projected faults [28] [29]. It mainly covers the control flow protection and data flow protection of the running software code and emulates the hardware protection techniques using efficient implementation of software modules. Also there are pure software-based fault tolerance techniques [30]. Among the Software Fault Tolerance (SFT) techniques are the N Version Programming (NVP), Recovery Block (RB) and Check Pointing, and Architecture Based Fault Tolerance (ABFT). Techniques for protecting the SRAM-based FPGAs are developed based on scrubbing and full/partial TMR to reduce power and area utilization by applying TMR to the critical sections [31] [32] [33].

When applying a fault tolerance strategy for protecting a system against the expected faults, a hybrid methodology is usually adopted. Where a combination between time, space and data redundancy is applied and implemented partially in hardware and software together. The fault tolerance strategy is usually thought of in the form of layers where each layer focuses on the protection of a certain scope of the system design. In chapter 3 we illustrate our taxonomy of the OBC architecture alternatives. This taxonomy can be used as a basis for selecting the most fitting fault tolerant approach to develop the OBC subsystem according to the mission requirements and the available power resources. Also, it provides the analytical formulas for calculating the OBC system reliability.

2.4 SRAM-based FPGAs for Space Applications

Typically satellites used to depend on discrete components such as microprocessors and microcontrollers to achieve the software processing tasks. Hardware designs were implemented using digital and analog components. However, with the advancement in the technology, FPGAs were introduced as programmable devices that can accommodate digital designs as well as complete embedded systems in one package. FPGAs are either anti-fused or SRAM-based. Anti-fused FPGAs can be programmed once and they provide robust performance against radiation on the expense of high incurred development costs. SRAM-based FPGAs are reprogrammable as they store the configuration data which represents the design in their local on-chip SRAM memories. The problem with such FPGAs, although they are more flexible in the design and development than anti-fused FPGAs, is the possibility of altering the design by radiation induced faults in the SRAM. Anti-fused FPGAs are usually of lower capacity in terms of the area that can hold the design when compared to SRAM-based FPGAs.

The typical contents of an SRAM-based FPGA from Xilinx are broken down in Figure 2-7 . The FPGA contains Configuration Logic Blocks (CLB) which holds flip flops for implementation of sequential logic as well as Look-Up Tables (LUT) for implementation of combinational logic. The number of slices and flip flops vary from family to another and even within the same FPGA family. Xilinx FPGAs also contain on-board Block Random Access Memory (BRAM) for storing of binary data as well as program codes, dedicated multipliers for fast multiplication and Digital Clock Managers (DCM).

The use of commercial grade SRAM-based FPGAs in space applications is an ongoing research topic which is state of the art [14]. Developing reconfigurable fast processing systems in miniaturized spaces and lower power consumptions with high reliability is possible using the technology afforded by SRAM-based FPGAs. Mitigation of the SRAM-based FPGA design is carried out through configuration memory scrubbing either through external controller or from the internal configuration access port [34]. Other techniques depend on using hot-spares configuration tiles which can be reconfigured to replace faulty tiles upon needed [35]. The first reported flight experience for an SRAM-based Virtex-4 FPGA from Xilinx showed 99.9% SEU immunity, where two FPGAs of LX and SX types where used and scrubbed externally using an anti-fused Actel RTAX FPGA [36].

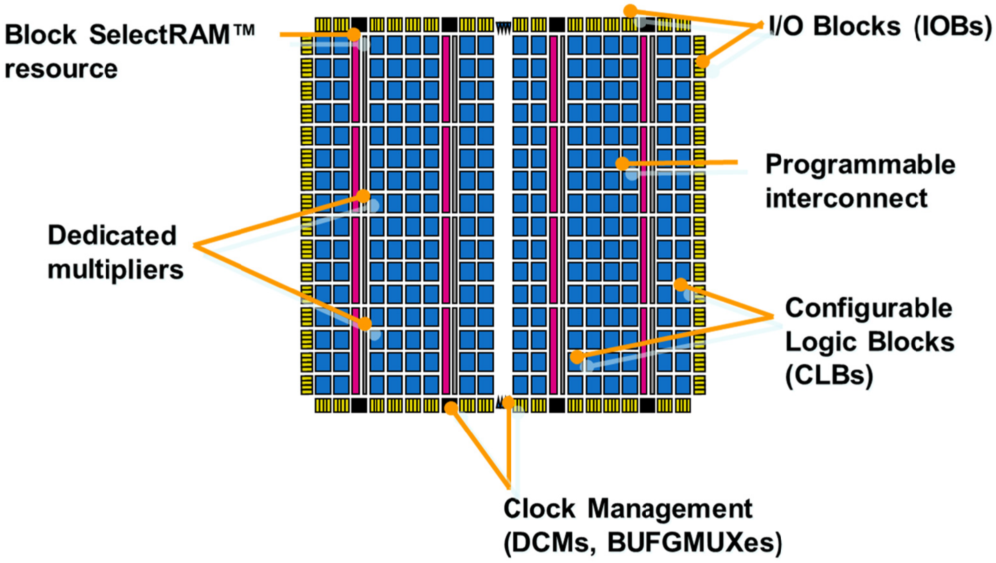

Block SelectRAM™ resource
I/O Blocks (IOBs)
Programmable interconnect
Dedicated multipliers
Configurable Logic Blocks (CLBs)
Clock Management (DCMs, BUFGMUXes)

Figure 2-7. Xilinx FPGA Typical Contents Breakdown © Xilinx.

Scrubbing is the principle fault tolerance technique used in maintaining SRAM-based FPGA's bitstream in an error free state. The bistream describes the connections between the different resources inside the FPGA to implement a specific design. It is stored in the form of 1's and 0's in the internal SRAM of the FPGA. Whenever an upset happens, such as an SEU, to the stored bits, its values might be changed accordingly from 1 to 0 or from 0 to 1. The scrubbing function, either implemented through an internal or external scrubber, is responsible for reading back the bitstream and correcting the errors found in it. The bitstream is stored in the form of binary frames of configuration bits. Each frame consists of 41 words and each word consists of 32 bits. Thus one configuration frame is 1312 bits in the Xilinx LX50 [37]. Each frame is protected by an Error Correcting Code (ECC). The ECC has the capability of SECDED. If a single error occurs to any of the bits in the frame, such as SEU, then the ECC checker circuitry can detect the error and correct it. If more than one bit error occurred, such as MBU, then the ECC checker circuitry would only detect the error and signal an error condition. The whole FPGA bitstream is protected by a readback Cyclic Redundancy Check (CRC) code that is used to check the integrity of the overall bitstream and whether it contains errors or not.

Figure 2-8 shows the configuration frame of the Virtex-5 FPGA referenced from [37] which presents the details of the configuration procedure and the possible configuration methods. It should be noticed that accessing the configuration bitstream is done through reading a complete frame through shifting its words in the Frame Data Register (FDR). In each frame 1280 bits are used as configuration bits, 16 bits are unused, 12 bits are used for ECC and 4 bits are used as miscellaneous configurations bits.

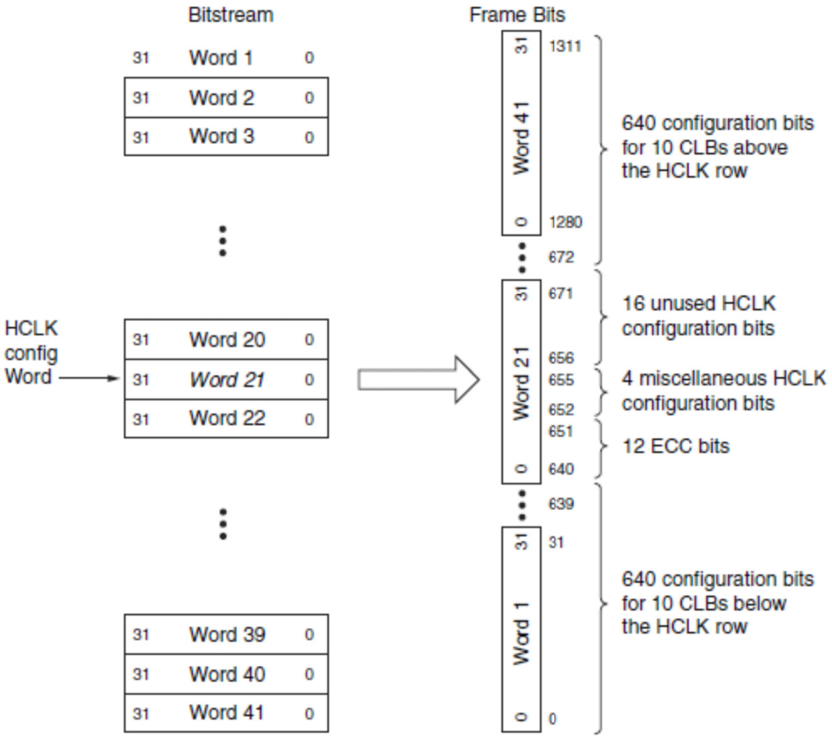


Figure 2-8. Virtex-5 Configuration Frame Format © Xilinx.

2.5 Previous Work

Developing OBCs for space missions has been concerned with the amount of functions which can be performed per unit volume and per unit power. For example, the strategy of NASA to advance the OBC technology was to develop OBCs with higher FLOPS/Watt/cm^3 and higher functions/cm^3. This strategy led the evolution from the Cassini EFC in 1992 to the Mars Path Finder MFC in 1994 till the X2000 in 2003 [38].

As space computers are expensive to develop because of the space grade components used in them, COTS were thought as a cost-effective solution as well as a higher performance one. The problems of SEUs in COTS microprocessors are investigated and fault tolerance techniques keep evolving due to the increasing threat as a result of shrinking the transistor feature sizes for higher performances and capacities [39]. However, the use of COTS should not jeopardize the safety of the mission. In high-safety missions, radiation hardened space approved processors should still be the ultimate choice such as the case of manned missions while COTS can be used for low-cost scientific missions [40].

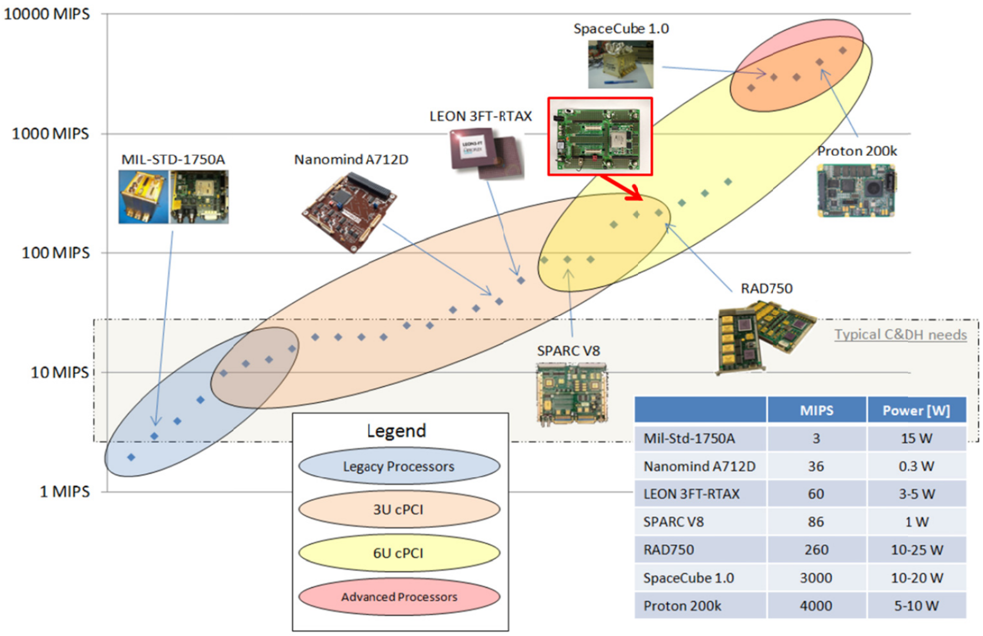
Some space agencies started, a long time ago, ambitious programs to promote the microsatellite engineering and to make space more affordable and accessible through applying advanced system engineering concepts and the use of COTS in space, such as the Myriade program by CNES [41]. Some missions already proved to fly successfully with COTS computers applying redundancy concepts [42] [43].

The next step was to work with multicore technology and introduce processors that can achieve acceptable high performance within tight mass, power and cost constraints. An example is NASA’s multicore OPERA Maestro processor which is based on the Tilera TILE 64 architecture [44]. NASA objective is to get high performance within error rate equivalent to the currently working uniprocessor systems.

The concept of distributed OBC is introduced through Adaptive Middleware for Fault Tolerance (AMFT) design. Tasks can be located among the OBC processing elements, which are located at different positions, in case one of the processing elements was hit by debris as an example [45].

The use of redundant parallel processing FPGAs interfaced to processing nodes to implement a fast computational platform was utilized in the design of the X-Sat microsatellite [46]. Some architectures use a mix between COTS microprocessors and radiation hardened FPGAs with reconfigurable processing resources for achieving high performance in communication satellites such as the Iridium constellation [47]. Satellite sensor networks based on distributed nodes computing are investigated in [48] [49]. NASA New Millennium Program, COTS processors with (10x-100x) the processing power of radiation hardened processors is investigated in [50] [51] [52] [53]. A reconfigurable OBC architecture using FPGAs for next generation space missions is under development by DLR [54].

The state-of-art in OBCs is to develop systems using FPGAs rather than microprocessors and ASICs for higher logic capacities and achieving the required performance, cost, and power constraints [14]. Figure 2-9 referenced from [14] shows the processor throughputs in comparison to their power consumption. Our design is targeting the 400 MIPS at a power consumption of less than 5 Watt maximum. The RAD750 achieves 260 MIPS with higher power consumption, almost 2X our target. Also, the price of such computers, although official quotation is not available to us, might be up to 500X our design cost. It has been stated in [54] that the unit price of QML-V grade microcontrollers and FPGAs vary between US$ 20,000 and US$ 70,000 while their COTS counterparts are less than US$ 100. The price of our system is targeting less than US$ 2000. The main difference between our design and all of the reviewed work is that it presents an integrated cycle to design a low cost COTS-based MPSoC OBC using SRAM-based FPGA and thoroughly qualify it. We collected the pieces of the puzzle together in this research.



	MIPS	Power [W]
Mil-Std-1750A	3	15 W
Nanomind A712D	36	0.3 W
LEON 3FT-RTAX	60	3-5 W
SPARC V8	86	1 W
RAD750	260	10-25 W
SpaceCube 1.0	3000	10-20 W
Proton 200k	4000	5-10 W

Figure 2-9. Satellite Processing Trends [14] © NASA – Ames Research Center.

The worst case MIPS/Watt for each computer is shown in Figure 2-10. Kyutech OBC power consumption is estimated as 4 Watt using heat sink only as a thermal mitigation technique as shown later in Table 7-7.

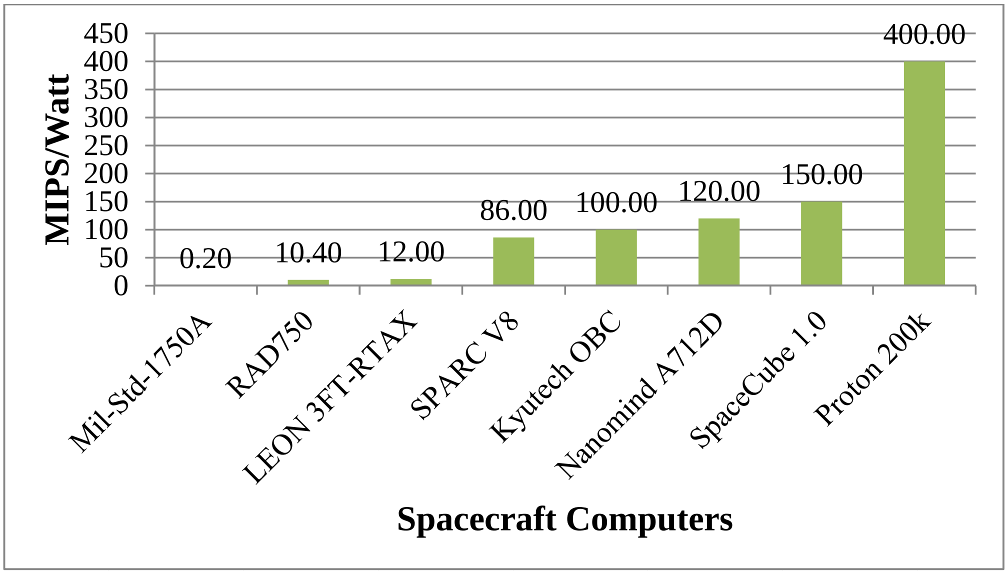


Figure 2-10. Spacecraft Computers Comparison based on Performance/Power Consumption.

The MIPS/Watt was estimated for the FPGA only without adding the power consumption of the external MRAMs. This gives meaningful comparison with other processors, which does not include external memories, such as the SPARC V8.

It would be more realistic to compare our design, Kyutech OBC, with computers which have closer MIPS. The RAD750 has 260 MIPS at worst power consumption of 25 Watt. If we consider that our design has 400 MIPS, total maximum throughput of the 4 embedded processors, then the MIPS/Watt for Kyutech OBC is far much better than the RAD750. We might increase the estimated power consumption of Kyutech OBC by counting for the external MRAMs. For example we consider 4 memories, one for each processor, each of which is 2 Mbytes and consumes about 0.6 Watt, and then we would have 68.97 MIPS/Watt which is still much better than the RAD750.

The MIPS/US$ can be estimated based on an assumption of the price of the space grade computers which is about 500 K USD. Kyutech OBC total price is less than 2,000 USD. Therefore the MIPS/US$ of Kyutech OBC is about 0.2 while for the RAD750 is about 0.00052. However, we have to state that the RAD750 is well designed and tested in about 28 space missions including the Mars Reconnaissance Orbiter mission [55]. RAD750 uses the IBM PowerPC 750 in its radiation hardened version. It is mostly suitable for missions requiring high level of reliability against radiation induced faults and accumulated doses. Our comparison with the RAD750 tries to present an estimation of how good our system is. However, our Kyutech OBC is designed for missions that can accommodate some sort of risk as we use COTS components. Manned missions which need ultimate reliability should use space grade and space approved components.

The innovation in this research is that it provides complete mitigation and systematic analysis as well as ground testing of the radiation and thermal conditions to develop a working system that can be used in real space missions. We judge that our design passed Technology Readiness Level (TRL) 6, as defined by NASA, which is concerned with testing the functionality of the engineering model/prototype in the relevant environment [13]. It would be raised to 8 after successfully demonstrating the operation in real flight conditions.

In the next chapter we introduce the MPSoC OBC design in details and illustrate the architecture alternatives for developing fault tolerant OBC systems using SRAM-based FPGA as well as the different scrubbing methodologies that can be applied.

Chapter 3 Multi-Processor System-on-Chip On-Board Computer Design

In this chapter we present the design of the MPSoC On-Board Computer. The MPSoC can act as the avionics system for small satellites as it can handle OBC tasks as well as central data processing for other subsystems. The architecture alternatives to develop a fault tolerant system are investigated. Having selected an adequate architecture, we then introduce the concept of using dynamic partial reconfiguration for developing fault tolerant MPSoC using SRAM-based FPGAs. The implementation of the MPSoC using Xilinx tools such as the Embedded Development Kit (EDk) and ISE Design Suite is thoroughly illustrated. Some sections of this chapter were part of the author publications at [56] [57] [58].

3.1 OBC Architecture Alternatives

Typically, an OBC consists of a processor system with necessary peripherals needed for an embedded system to operate. Special Input/Output interface circuitry might be added to match with the interface requirements of the system but still the major parts of any embedded system are of common type as shown in Figure 3-1. Embedded system building blocks typically comprise microprocessors/microcontrollers, reset modules, timers, memories either volatile or non-volatile, interrupt controllers, bus controllers/arbiters, Input/Output interface logic, watch dog timers and programmable logic.

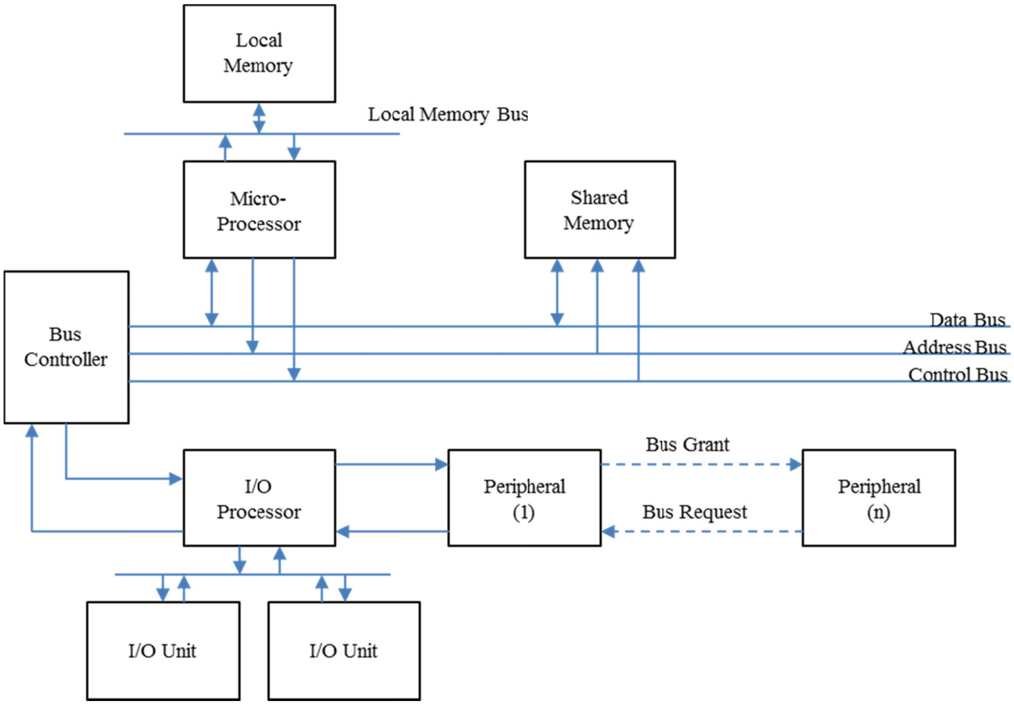


Figure 3-1. General Embedded OBC System Architecture.

Before thinking of the fault tolerance techniques that can be applied to protect any embedded OBC system, the design should be analyzed to determine the error prone areas at first. Also a careful study and expectation of the type of faults and rate of upsets is necessary to select components and techniques that can mitigate and/or mask the effect of such faults. Table 3-1 presents a summary of the target areas to be protected in an embedded OBC system together with the most common hardware and software fault tolerance protection techniques that can be applied.

Table 3-1. Scope Protected and the Fault Tolerance Protection Techniques.

Scope	**Hardware Technique**	**Software Technique**
Processor Registers	Processor redundancy with cross voting	Re-execution in extra time
	Watch dog processor	Data and control flow protection
	Dual processors with comparator	RB, NVP, Check pointing
Memory	Error Detection and Correction (EDAC)	Software EDAC
	Multiple blocks	Memory Scrubbing
FPGA configuration	Scheduled repair	-
	Emergency repair	
	Running repair	
Input/Output	Minority voters	Repetition of execution
Reset and Clock	Redundancy	-

We present a new taxonomy for classifying redundancy while implementing satellite OBCs. The areas that are most prone to SEUs are those carrying data that can be corrupted. The system memory carrying the program code, program related variables and data in addition to the processor registers are the areas of the highest risk to be affected by the SEUs. When an FPGA system is used, then if it was an SRAM-based system, the configuration data should be protected. The techniques used for protecting the FPGA configuration data depend on reading back the configuration data and detecting if the data contained an error or not. The corrected data would be written back in case an error was detected. A complete reconfiguration every period of time can be initiated to overcome the effect of accumulated errors and as a precautionary action.

Processors and memories in an embedded system can be replicated or used as single modules. If a processor is used alone without repetition we call it Single Processor (SP), while if a processor is replicated we call it Multiple Processor (MP). The same taxonomy is valid for memory, either Single Memory (SM) or Multiple Memory

(MM). Combining these architectural possibilities we obtain four architectures: SPSM, SPMM, MPSM, and MPMM as shown in Figure 3-2. We call that taxonomy, Abu Zaid's Taxonomy of fault tolerant processor-memory interfaces [56]. Abu Zaid is the author's family name.

The architectures vary in the number of used blocks and their connection schemes. The number of used blocks proportionally affects the system complexity. If the number of used blocks increases then the complexity increases and vice versa. If some of the components are redundant then the overall system reliability would increase. Another dimension might be added to the proposed taxonomy which is the use of EDAC. It can be either implemented in hardware logic or in software routine. The system then can be classified as (SP,SM,E), (SP,SM,NE), (SP,MM,E), (SP,MM,NE), (MP,SM,E), (MP,SM,NE), (MP, MM,E), and (MP,MM,NE). Where the (E) denotes the existence of EDAC and (NE) denotes the absence of the EDAC.

The system data reliability can be estimated by estimating the reliability of each of the processor and memory modules. In our design we use an FPGA to implement the processor system(s). The FPGA that is used is the Xilinx XC5VLX50, which contains a configuration array size of 392,124 words, each word consists of 32 bits. Thus the (full) configuration data size is around 11.97 Mbit.

We would give a numerical example and plot of the OBC system reliability, based on our proposed fault tolerance taxonomy of the processor-memory interface. The calculations would make use of the estimated Failure-In-Time (FIT) for the Virtex-5 FPGAs which is 165 FIT/Mbit. The FIT corresponds to the number of expected failures per 10^9 hours. Thus the Mean Time Between Failures (MTBF) is about 57.814 years, in case the FPGA is fully configured. The FIT figure is estimated per Mbit of configuration data therefore the total FIT for an FPGA should be calculated based on the actual number of slices occupied by a design (the design size in configuration bits). If we calculate for the full configuration data size which is about 11.97 Mbit for the XC5VLX50, then the FIT value would be in the order of 1975 FIT. Thus the failure rate would be estimated as (λ_P= 0.017297 failures/year). We select the memory to be used in the design as a Magneto-resistive Random Access Memory (MRAM). The EVERSPIN MR4A16B memory is a good example as it can retain data up to 20 years. Thus its failure rate is estimated to be ($\lambda_m = 0.05$ failures/year).

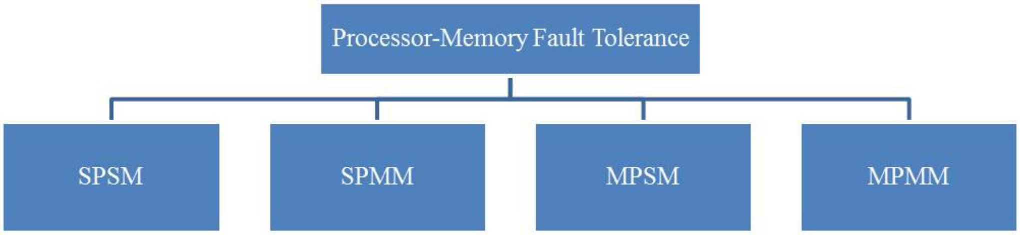


Figure 3-2. Abu Zaid Taxonomy of Fault Tolerant Processor-Memory Interfaces [56].

The system can have different architectures as indicated in Table 3-2. The reliability calculation formula is indicated for the different cases. The architectures make use of redundancies to provide fault tolerant performance. Nevertheless, the redundancies were limited to a practical number of modules that can be included due to size, area and power constraints that must be considered when designing satellite systems. Maximum of 3 redundant units can exist at any architecture. The use of the TMR scheme is well known in designing reliable systems. The reliability formulas apply a (2 out of 3) reliability calculation. If EDAC is used it would certainly have an effect on the overall reliability. It would raise the reliability values of the proposed architectures.

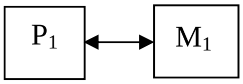
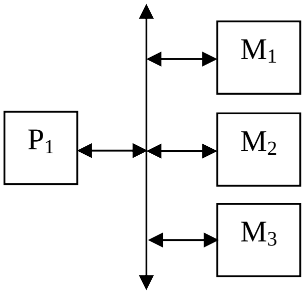
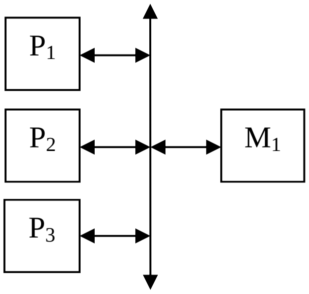
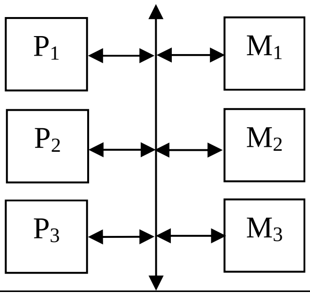
TMR systems use voters to produce the final output value. The voters themselves have reliability calculations related to them. If the voter and EDAC are implemented inside the FPGA then their reliabilities can be estimated by first calculating their expected FIT then estimating the MTBF from it. The MTBF would be used to calculate the failures per year which is the failure rate for that design portion. To calculate the FIT of a design or even part of it, then the number of configuration bits occupied by that design would be multiplied by the FIT/Mb for that type of FPGA technology. The FIT for the concerned design would then be estimated. Therefore if we would like to include the EDAC and voter effect on reliability then we have to estimate their own reliabilities and then include them in the reliability calculation formula according to their connection in the design structure.

In assessing the reliability of each of the proposed architectures we have to differentiate between assessing the reliability to understand SEU susceptibility and assessing the reliability related to the device lifetime. The SEU susceptibility would indicate the vulnerability of the design towards SEUs and its ability to withstand faults induced due to them. On the other side, device reliability is an issue that would indicate the device ability to function properly during its planned lifetime. In order to improve the system data reliability or trustworthy, variety of techniques can be used.

TMR is one of the techniques that is popular and well established. It is used to improve data reliability through 2 out of 3 voting. On the other side, building a system that would live longer in terms of operational hours and robustness in the different operating conditions can be achieved through manufacturing and design approaches. Manufacturing should focus on selecting durable materials of characteristics suitable to the expected operating conditions. While the design should focus on defining architectures that consider redundancy on different levels such as modules, subsystems and even complete systems.

Redundancy can be active where the redundant units operate simultaneously with the main units or it can be inactive where the redundant units are only activated when the main units halt or malfunction. Switching between redundant units can be done through a system level controller. Systems can have their redundant units running at the same time where the main units are running. Other systems might have the redundant units switched off and they are only switched on when an error in the main set takes place such as switching from main OBC to reserve OBC.

Table 3-2. Architecture Alternatives for Processor-Memory Interfaces.

Fault Tolerance Type	**Architecture Processor-Memory**	**Reliability Formula**
SPSM		$\mathbf{R} = \boldsymbol{R_p R_m}$ **(3-1)**
SPMM		$\mathbf{R} = \boldsymbol{R_p\,(3R_m^2 - 2R_m^3)}$ **(3-2)**
MPSM		$\mathbf{R} = \boldsymbol{R_m\,(3R_p^2 - 2R_p^3)}$ **(3-3)**
MPMM		$\mathbf{R} = \boldsymbol{(3R_p^2 - 2R_p^3) \times (3R_m^2 - 2R_m^3)}$ **(3-4)**

When memories are interfaced to more than one processor on the same bus it is said to be shared. If memories are interfaced to processors on separate buses they are said to be non-shared or distributed. Shared memory systems are known as tightly coupled systems while non-shared memory systems are known as loosely coupled systems. In the case of single processor the memories are only accessible to that processor so they are always considered as shared. It is meaningless to think of a non-shared memory system in case of single processor. Therefore, in the architecture presented in Table 3-2, the MPSM is a shared memory system, it can be implemented as a non-shared memory system as shown in Figure 3-3. Multi-Processors will have 2 out of 3 voting on their results, while each of them has got its own separate memory interface.

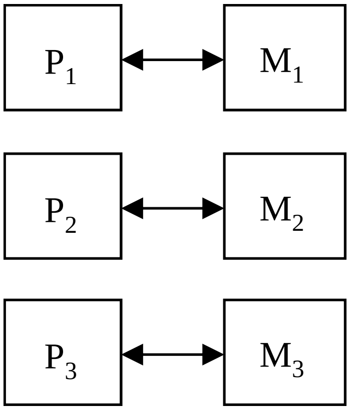


Figure 3-3. MPSM with non-shared Memory.

The same concept applies in the case of the MPMM where in Table 3-2, it is shown as a shared memory system, while in Figure 3-4 it is shown as a non-shared memory system. Each processor will have its own interface with multiple memory units. Voting (2 out of 3) will take place in two levels, among the memory units attached to each processor then among the processors themselves.

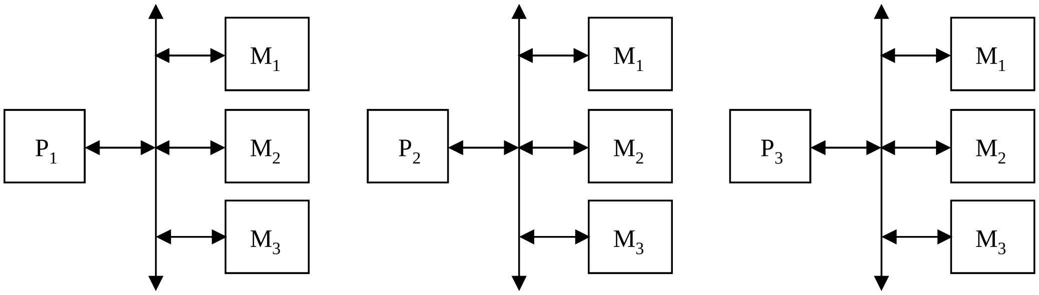


Figure 3-4. MPMM non-shared memory.

It is necessary to notice the different architectures and their effects on the reliabilities formulas of the system. The following graphs show the system soft error reliability for selected architectures using a very pessimistic assumption. We used the FIT value for a fully configured FPGA as if it is the FIT value for any single processor system from those indicated in the architectures. In fact this is a very tough approximation, as the real FIT for the processor system will be much smaller due to the fact that it occupies only a portion of the FPGA configuration bitstream.

When analyzing reliability, it would be necessary to consider that some designs might not only have redundant processors inside one FPGA but also they might have complete redundant FPGA devices. The FPGAs might be of different types and thus of different FIT values. The FPGAs digital designs, which might have several embedded systems, can still form a (2 out of 3) redundancy by communicating data in between them. Voting would then be carried out inside each FPGA among the data generated internally and the data received from the other two external FPGAs. In the case of the processor modules inside the same FPGA, data exchange is easier as it takes place through mailboxes via a sort of Message Passing Interface (MPI).

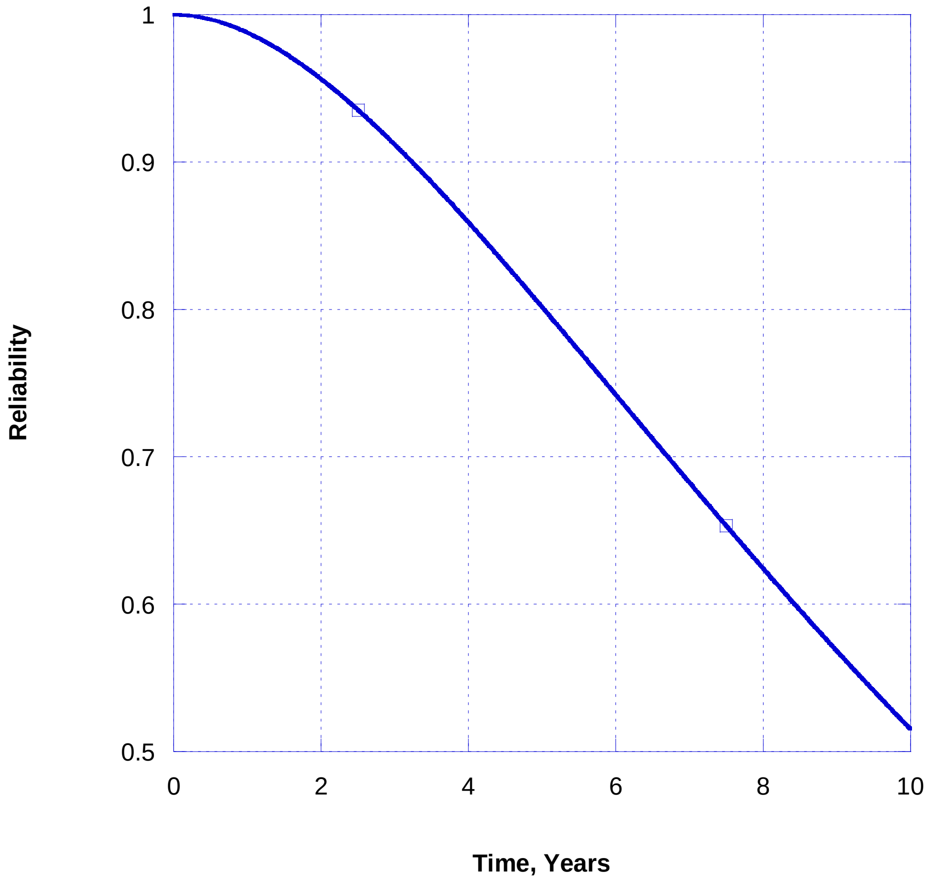


Figure 3-5. MPSM non-shared Memory Reliability Curve.

Figure 3-5 shows the soft error reliability of the MPSM non-shared memory model. It is calculated from Eq. (3-5):

$$\mathrm{R} = 3\mathrm{R}_{\mathrm{ps}}^{2} - 2\mathrm{R}_{\mathrm{ps}}^{3}$$

$$\text{where: } \mathrm{R}_{\mathrm{PS}} = \mathrm{R}_{\mathrm{p}}\mathrm{R}_{\mathrm{m}}.$$

(3-5)

R_{PS} is the processor system reliability which consists of both the processor and the memory, R_{p} is the processor reliability, and R_{m} is the memory reliability. The reliability of the MPMM architecture with non-shared memory is shown in

Figure 3-6. It can be calculated by estimating the reliability of each processor system at first then estimating the reliability among the processor systems. The processor system in this case means the processing set which consists of 1 processor and 3 memories. The overall system consists of 3 processor systems (processing sets). The reliability of each processing set is the multiplication of the processor reliability and the (2 out of 3) memory reliability. Calculations are shown in Eq. (3-6):

$$\mathrm{R} = R_{ps}^{3} + 3R_{ps}^{2}\left(1 - R_{ps}\right)$$

$$\text{Where: } R_{ps} = R_{p}\,(R_{m}^{3} + 3R_{m}^{2}(1 - R_{m})).$$

(3-6)

The MPMM architecture with non-shared memory shows higher reliability values than the MPSM architecture with non-shared memory for the same service period. However, the MPMM consumes high power relative to the MPSM. A compromise should be made between the gain in reliability and the losses in consumed power.

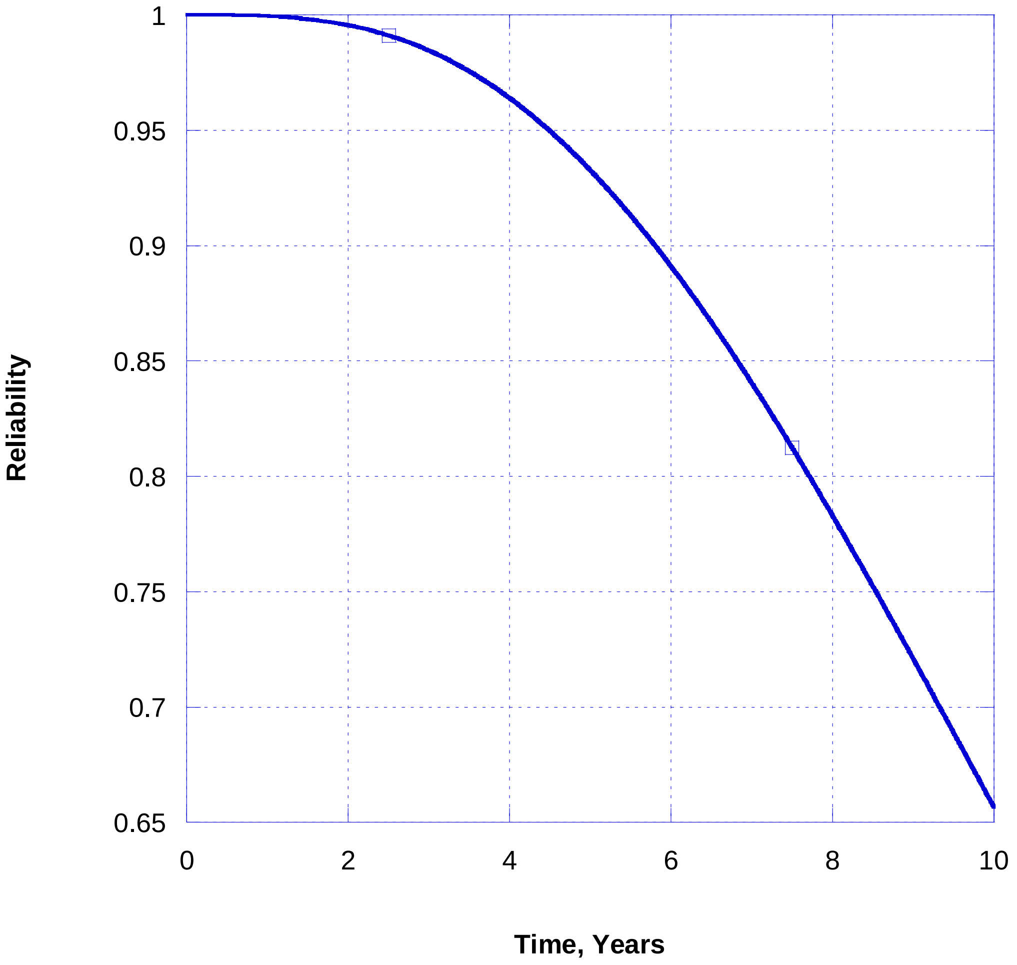


Figure 3-6. MPMM non-shared Memory Reliability Curve.

Two other schemes are considered in our architecture evaluation for the OBC design. The MPMM with shared memory and the MPSM with shared memory as shown in Table 3-2. The reliability curves for both schemes are shown in Figure 3-7 and Figure 3-8 respectively.

It is interesting to notice that the power consumption profile of the MPMM with non-shared memory would be the highest as it contains more components, at the same time it is of the highest data reliability among the Multi-Processor schemes. The lowest power consumption profile would be the MPSM with shared memory which would be of the lowest reliability as well. Both the MPMM with shared memory and the MPSM with non-shared memory are almost of the same power consumption profile and close reliability values. In our selection of the architecture we have to consider the mission reliability requirements, service conditions and satellite power budget. In our MPSoC design, the MPMM scheme is not suitable for the mission power budget. We selected the MPSM with non-shared memory as it has a moderate power consumption and reliability while less complexity in managing memory interface than the MPMM with shared memory.

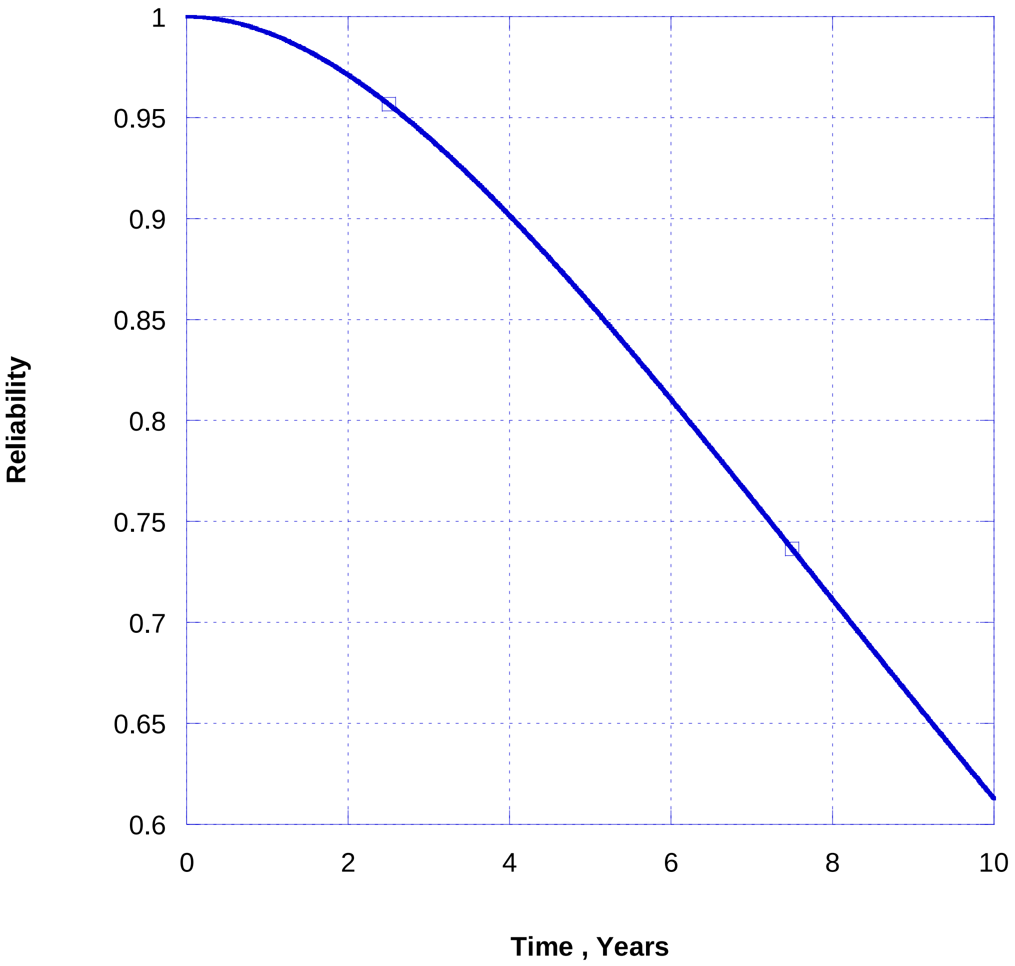


Figure 3-7. MPMM shared Memory Reliability Curve.

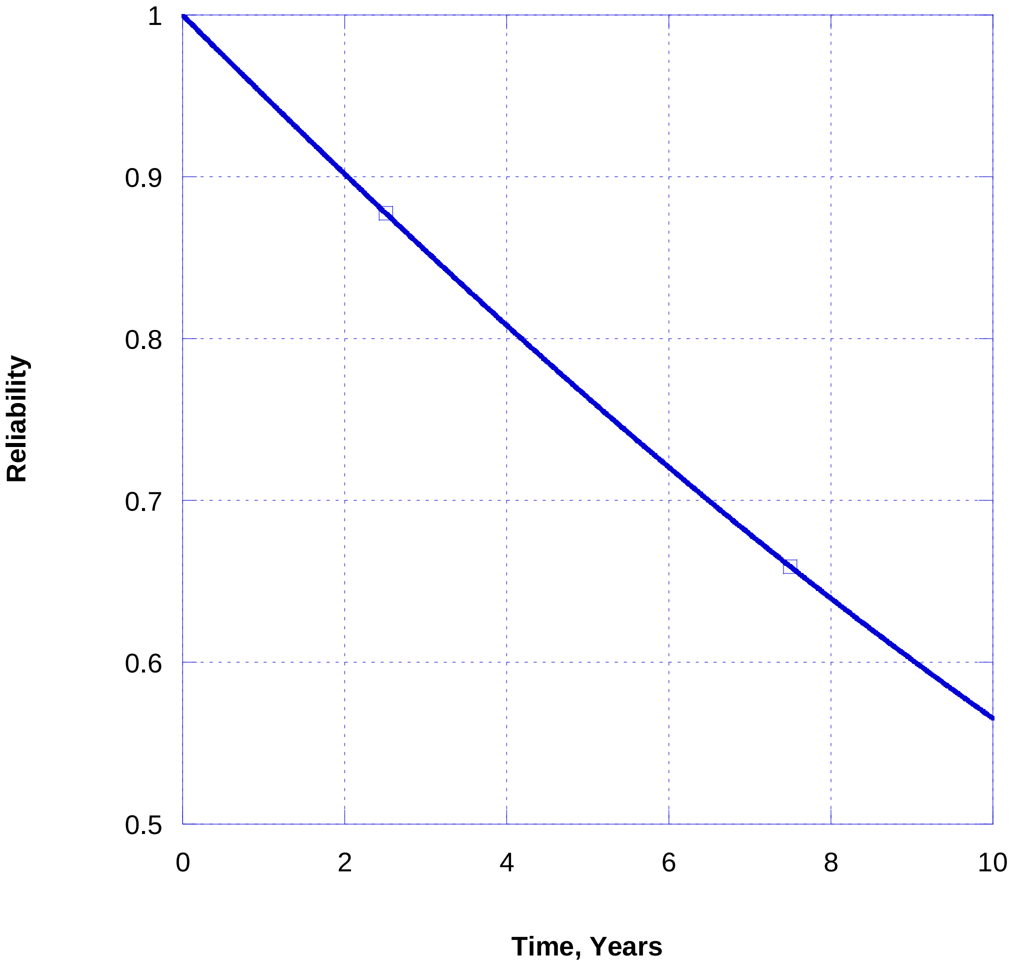


Figure 3-8. MPSM shared Memory Reliability Curve.

By analyzing the MP curves from Figure 3-5 to Figure 3-8, we can estimate the degradation in soft error reliability over the mission service period for the Multi-Processor architectures. Therefore, for example, if a satellite mission was planned to serve for 4 years in space then from the MP reliability curves we can estimate the soft error reliability values as: 1) MPMM with non-shared memory about 96%, 2) MPMM with shared memory about 90%, 3) MPSM with non-shared memory about 86%, and 4) MPSM with shared memory about 81%. These results are summarized in Figure 3-9.

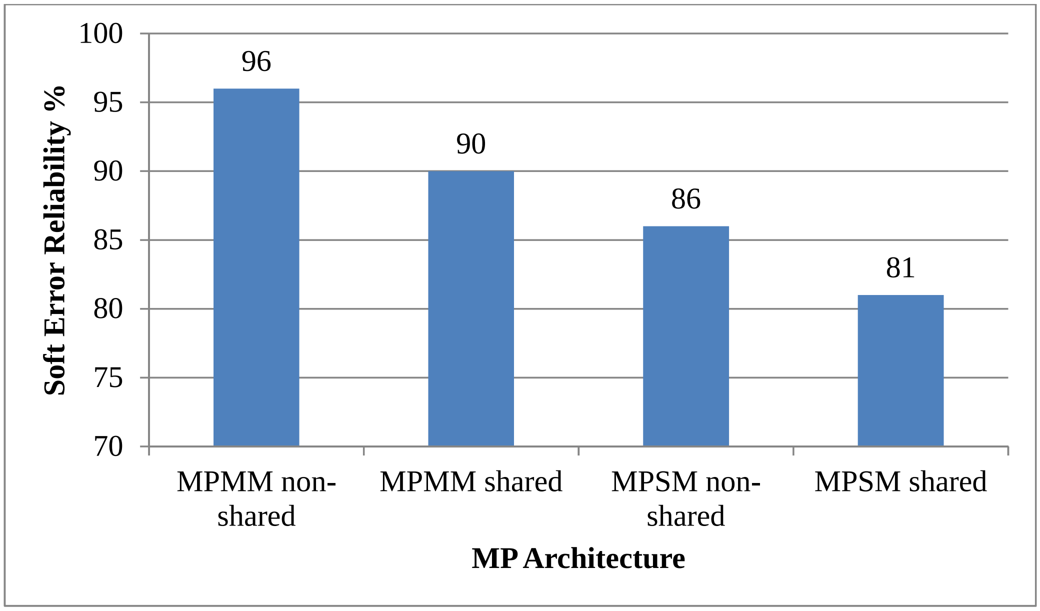


Figure 3-9. Soft Error Reliability for MP Architectures.

In the abovementioned analysis, we focused on analyzing the MP cases as we are implementing an MPSoC. However, in case of implementing an SP system then it should be straight forward to estimate the soft error reliability from the formulas in Table 3-2. Next we present the use of the dynamic partial reconfiguration concept to scrub the configuration frames in the SRAM-based FPGA and provide fault tolerance against SEU induced faults.

3.2 FPGA Reconfiguration Strategies

Several strategies have been proposed to deal with the reconfiguration of FPGA bitstream in case of induced errors. The typical form of errors that might take place in the bit stream is the SEU, it is known as a flip in the configuration data. Thus the value of binary stored information will be negated, (1) changes to (0) and (0) changes to (1). The SEU is one form of what is globally known as SEE which includes other forms of faults caused due to single events of radiation effects on electronic components. The effect can be either in a single bit known as SBU or it can take place in multiple bits known as MBU.

Reconfiguration of the bit stream is meant to overcome the effects of changing the bitstream values stored in the FPGA SRAM. The bitstream is stored in the form of configuration frames. Each frame contains a group of configuration bits, for Virtex 5 the frame consists of 1312 bits. The configuration bits are grouped into words of size 32 bits. Thus each frame consists of 41 words, each of which consists of 32 bits that is a total of 1312 bits.

The frames contain the bits that are related to the CLB, BRAM, Input/Output Blocks (IOB) and other configuration data for FPGA interconnects routing and modules. Each frame contains 12 bits for ECC such as Hamming code. It can be used for SECDED. The configuration time for the FPGA is based on the configuration clock rate and the configuration array size. The array size consists of words and the total bit stream can be viewed roughly as the total number of words times 32. Each word of 32 bits would take one clock cycle to be programmed into the FPGA. Thus an FPGA such as the XC5VLX50 would approximately take 7.42 mS to be fully configured at a configuration clock of 50 MHz.

A group of registers are used to manage the access to the configuration data through a sequence of control commands. The configuration data can be read or written as one complete frame which is the smallest unit to be accessed for configuration. The frame data can be accessed by setting the Frame Address Register (FAR) which denotes the type of the block being accessed and the FPGA top or bottom half, details are described in [37].

The Xilinx XC5VLX50 has two versions, the LX50T and the LX50. Both versions are of identical technology except for some extra logic slices and hardwired modules in the LX50T. In our research we performed the design on both FPGAs and we used them interchangeably. The LX50T contains a total of 12,547,968 configuration bits but about 11.37 Mbit are the actual available bits for configuration. The Virtex 5 FPGAs can load an encrypted bit stream using Advanced Encryption Standard (AES) with a key length of 256 bit. The FPGA contains an AES decryption module that can be used only to decrypt the encrypted bitstream. The decryption module cannot be used in the design being loaded in the FPGA. It is dedicated only to the bitstream decryption. The FPGA configuration can take place through several modes that can be controlled through configuration pins. The modes are either in Master or Slave operation. In the Master operation, the configuration clock is driven from an internal oscillator. While in the slave operation the clock is an input. The configuration data can be loaded in parallel or serial streams. Other modes are available such as the JTAG/Boundary-scan mode and the SelectMap interface mode.

The sequence of configuring the FPGA proceeds in two phases, the setup phase and the bitstream loading phase. In the setup phase the FPGA clears the internal configuration memory just after power-up, it then detects the configuration mode to be used such as Master or Slave and parallel or serial transfer. In the bitstream loading four stages are executed, synchronization, device ID check, loading of configuration data, and the CRC check. A 32 bit CRC is used to denote any errors in the configuration data loaded to the FPGA. Details of the Xilinx XC5VLX50 characteristics and operation can be found in user guides, manuals and data sheet.

The configuration stream can be read-back after being loaded to the FPGA internal SRAM. The read-back can take place through the external SlelectMap interface or the Internal Configuration Access Port (ICAP) primitive. In case errors exist in the bit stream due to SEU, the read-back technique can be used for mitigation. There are three main strategies when dealing with the SEU problem in the configuration data:

1) Scheduled maintenance
2) Emergency maintenance
3) Running repairs

3.2.1 Scheduled Maintenance

The scheduled maintenance depends on triggering a complete reconfiguration for the FPGA every specified period of time. It does not try to detect whether bitstream errors exist due to SEU or not. In that technique a wash out is performed through rewriting the whole configuration data once more thus allowing to wipe the transient errors formed due to SEUs, if any. That strategy can be used when the time between successive reconfigurations is smaller than the mean time between bit upsets.

The main advantage of the scheduled maintenance is that it performs a complete washout which removes all accumulated configuration errors whether SBUs or MBUs. Scheduled maintenance can take place by reading data from an external flash memory. That flash memory is different from the original platform configuration flash. However, it holds an exact bitstream copy, golden image, identical to the one that was initially loaded to the FPGA from the platform configuration flash. The scheduled maintenance procedure is shown in Figure 3-10. A configuration timer would trigger timeouts to the configuration controller which would read the configuration frames from an external flash memory and write it back to the FPGA through the internal ICAP.

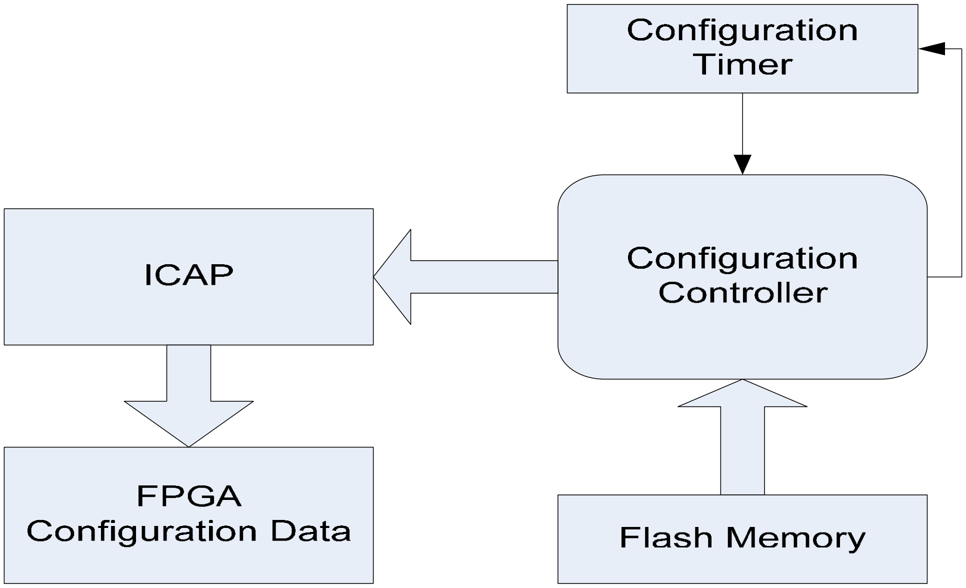


Figure 3-10. Scheduled Maintenance – Reload from External Flash.

Another approach is to trigger a complete reboot for the FPGA system to reload the whole configuration stream from the platform configuration flash as in Figure 3-11.

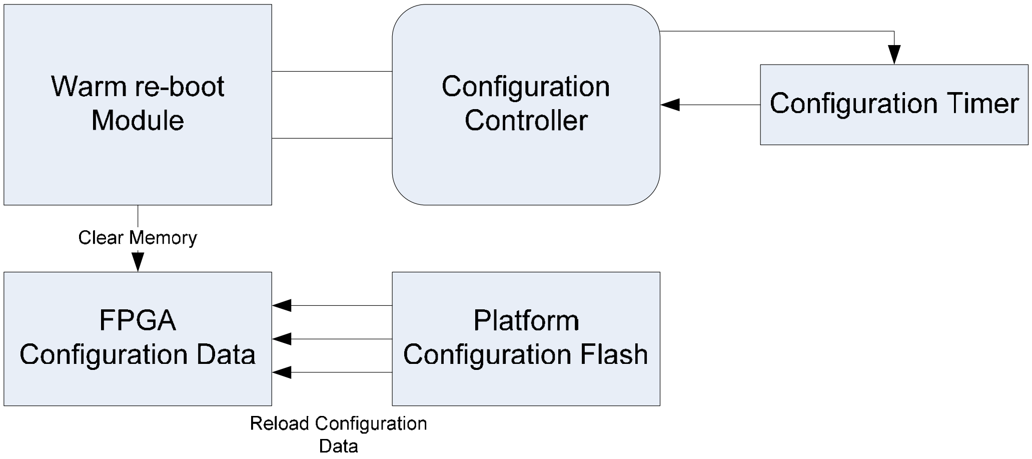


Figure 3-11. Scheduled Maintenance – Reload from Platform Configuration Flash.

3.2.2 Emergency Maintenance

Emergency maintenance as shown in Figure 3-12 depends on being more responsive than the scheduled maintenance. Any SEU in the configuration frames would be detected by performing a scan on the whole configuration frames. The read-back of the frames is done by a dedicated logic which runs independent of the FPGA design. This technique makes use of the read-back CRC logic primitive available for Virtex 5. The read-back CRC logic reads the configuration frames through the ICAP primitive and the CRC values for the read frames are calculated through the frame_ECC primitive. The CRC syndrome is used to locate the error in in the frame bitstream, if any.

SBU can be detected and located for correction by simply flipping their values. Double bit errors can be only detected and a complete reconfiguration would be initiated. In case an MBU exists then the frame_ECC cannot identify their existence reliably and the whole bitstream would be invalid. The bitstream integrity can be judged through the bitstream CRC value while the individual configuration frames integrity is judged through the frame ECC value. In case of uncorrectable errors in the bitstrem then a full reconfiguration might be needed if the errors lead to serious design failures.

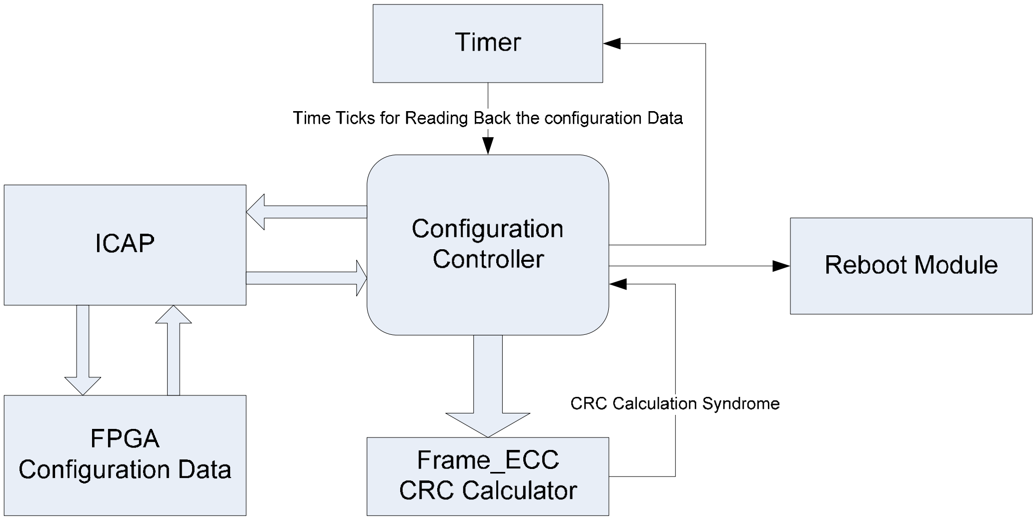


Figure 3-12. Emergency Maintenance.

The scan time of the total configuration frames depend on the FPGA configuration array size and the scan clock rate. The clock can be an external clock or the FPGA can make use of its internal oscillator. The use of the internal oscillator is meant to avoid the external circuitry and connections that might be a source of erroneous operation due to design, routing, wiring mistakes or even components wear out. The internal ICAP can operate at a maximum frequency of 100 MHz. The SBU, once detected in any frame, would be corrected by flipping the faulty bit, while if an MBU is detected then it would not be corrected. The correction takes place by writing back the correct frame through the ICAP interface. The control signals of the frame_ECC shows the interpretation of the error condition using the syndrome valid, ECC error, CRC error and syndrome signals.

3.2.3 Running Repairs

In case errors cannot be permitted for the maximum detection time, when the scheduled scans would run, then the running repairs strategy can be used. The running repairs depend on the availability of redundant modules that would run either simultaneously or consecutively to provide redundant results as shown in Figure 3-13 and Figure 3-14.

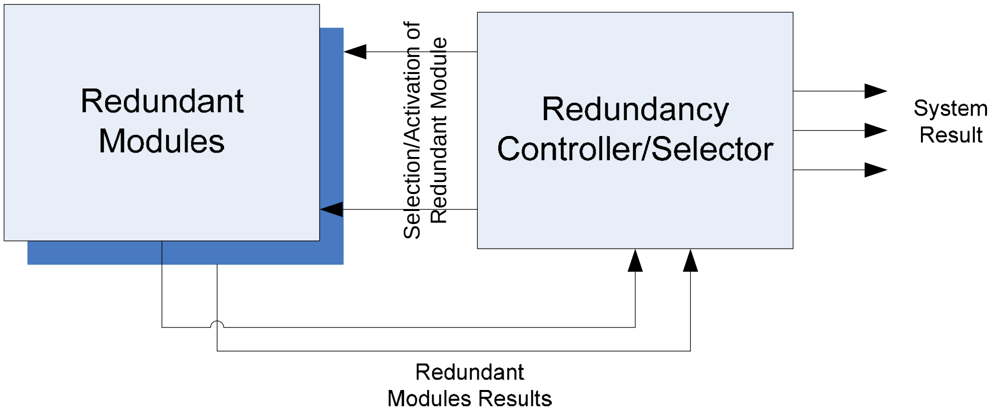

Redundant Modules
Selection/Activation of Redundant Module
Redundancy Controller/Selector
System Result
Redundant Modules Results

Figure 3-13. Running Repairs – Selection of an Active Module.

The time between the error occurrence in configuration bits and the detection of that error can be sufficient for fault propagation and hence the initiation of a system wide failure. The application and the design details would define whether the system is susceptible to enter any of its failure modes due to delayed detection of bit upsets or not. If the system should be repaired immediately, in order not to have a system wide failure, and if the repair should be executed without resetting the system, in order not to lose critical data, then redundant modules are the perfect choice. The use of the redundant modules would allow overcoming the problem that might happen in the configuration bits of any of the individual modules. As it is very unlikely to have bit upsets taking place at the redundant modules configuration frames at the same design locations on the same time.

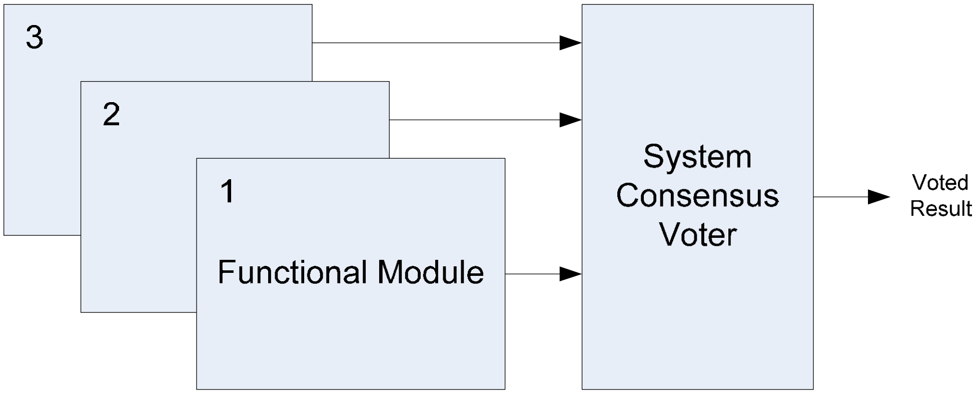


Figure 3-14. Running Repairs – Active Modules with Voting.

In real avionics systems, combination of the aforementioned strategies is typically implemented. Scheduled maintenance takes place regularly over larger time intervals. In standby modes emergency maintenance through read-back CRC can be used to avoid error accumulation. While during critical operation of the system, where any disturbance to the system or halt to reconfigure is not allowed, the running repairs strategy can be used. Xilinx recommends using a hybrid mitigation strategy and it ensures that SEU induced errors can be actively mitigated through configuration frames scrubbing. In the next section we present the details of the MPSoC design and implementation.

3.3 MPSoC OBC System Design

The MPSoC technology is used to develop high performance embedded systems in a miniaturized form. Our design for the MPSoC OBC was conducted using Xilinx Embedded Development Kit (EDK) software package. We designed the hardware using the EDK - Xilinx Platform Studio (XPS) while we designed the software using the EDK - Software Development Kit (SDK). Also, we used the Xilinx ISE design suite to design a top level schematic to integrate the MPSoC design from EDK with the SEU controller IP core. The top level schematic was developed to achieve design fault tolerance through mitigating the FPGA configuration frames via internal scrubbing.

EDK provides variety of IP cores that can be used to develop complete embedded systems with different architectures to fit for the specific design requirements. The block diagram of the OBC implemented in the Xilinx Virtex5 LX50 FPGA is shown in Figure 3-15. The system consists of 4 MicroBlaze soft IP core processors. Each processor is interfaced to a BRAM where the initial boot loader, system initialization program code, and application code are stored. The BRAM size is 32 Kbytes and can be increased up to 64 Kbytes or more in larger size FPGAs. The LX50 is pin compatible with larger size FPGAs such as the LX85. This gives the advantage of replacing the FPGA without changing the design to get larger resources when needed. All the microprocessors are connected to the MicroBlaze Debug Module (MDM) IP core for software debugging. The debugging interface with the external software Integrated Development Environment (IDE) is of JTAG type. The Xilinx USB/JTAG cable is used to connect to the system during debugging and configuration flash programming.

The configuration bitstream generated by the Xilinx EDK package, using the ISE Design Suite tools to compile, synthesize, map, place and route the design, is stored in the configuration flash memory. The configuration flash memory might have Serial Peripheral Interface (SPI) or Byte Peripheral Interface (BPI) with the FPGA. Whenever the FPGA is restarted, its bitstream will be reloaded from the configuration flash memory.

Each processor has got its own Processor Local Bus (PLB). The processor uses the PLB to communicate with its peripherals which are connected directly as slave terminals on the bus. The processor is the master terminal on the PLB. The arbitration between the slave peripherals, requesting the bus, is granted through the PLB bus arbiter. The arbitration policy is configurable. For example it can be set to daisy chain or priority based arbitration.

The 4 MicroBlaze processors communicate together through FIFO mailboxes. The size of each mailbox is 512 bytes. Each mailbox is bidirectional and can have asynchronous read/write operations. The mailboxes connect between the processors through the PLB bus of each processor. There are 6 mailboxes connecting all of the processors together.

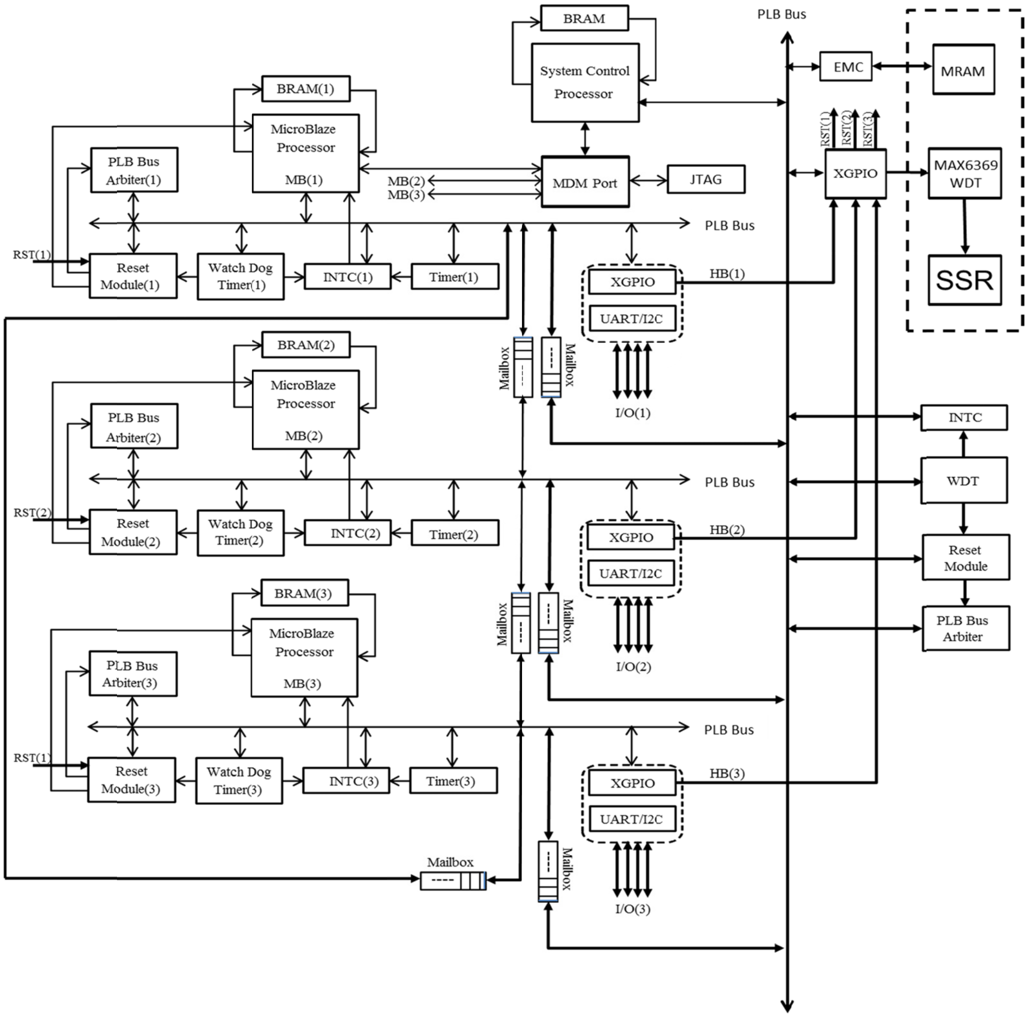


Figure 3-15. MPSoC OBC Block Diagram.

Three MicroBlaze processors are considered as application processors which run the user code. These are the processors labeled MB1, MB2, and MB3. The fourth processor is labeled as the Master System Control Processor (MSCP), This is the system master which handles the monitoring function on the operation of other processors. Each processor of the three user processors generate a heartbeat signal based on an interrupt received by a Watchdog Timer (WDT) connected to its PLB.

When the WDT times out, it sends a signal to an Interrupt Controller (INTC) to wake up the processor in concern. The WDT timer interval is set at the design time and can be calculated from Eq. (3-7).

$$\mathrm{WDT}_{\mathrm{Interval}} = 2^{\mathrm{n}} \times \mathrm{Clock\_Period}$$

Where the *Clock_Period* is:

$$\mathrm{Clock}_{\mathrm{Period}} = \frac{1}{\mathrm{System\_Clock\_Frequency}}$$

(3-7)

The WDT interval can be controlled by setting the value of (n) at design time. The MSCP implements software WDTs to monitor the three user processors. These software-based WDTs at the MSCP are reset based on receiving the heart beat signals labeled HB(1), HB(2) and HB(3). These signals are sent through the parallel General Purpose Input/Output IP core (XGPIO) of each MicroBlaze processor to the XGPIO of the MSCP. In order to provide different paths to report the heart beat signals, as they are crucial to the operation of the system, a backup path is provided. The MicroBlaze processors send an Acknowledge (ACK) response based on a request issued by the MSCP as shown in Figure 3-16. The MSCP regularly invokes the slave processors using the (Ping) request. The slave processors should respond to the (Ping) request by issuing an (Ack) response. The response should be received by the MSCP within a specified time interval. If the time interval exceeds the limits set without receiving the (Ack) response, the MSCP resets the slave processor by sending a reset signal to the slave processor reset module.

The MSCP itself reports its heart beat signal to an external WDT, The MAX6369. The external WDT sends a reset signal to the Crydom MPDCD Solid State Relay (SSR). The reset signal is an active low signal. It is pulled up and kept high via a pull-up resistance connected to the supply voltage. Whenever the MSCP fails to reset the external WDT (MAX6369), the reset signal will be set to low and thus disconnecting the power supply from the FPGA board leading to power recycling.

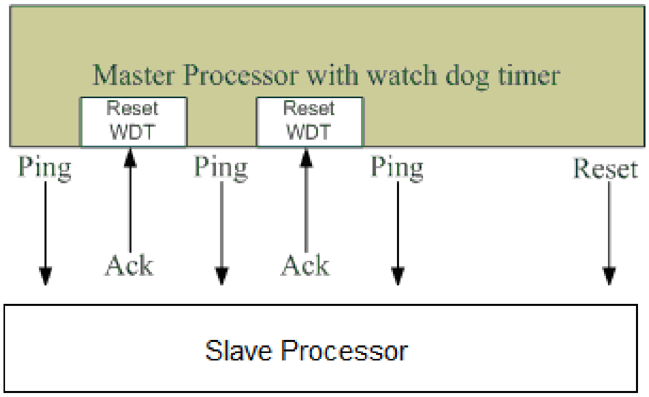


Figure 3-16. Ping/Ack Watchdog Timer **[59]**.

The XC5VLX50 FPGA has an internal thermal diode that can be used to read the die temperature. Reading the die temperature can be done through the system-monitor IP core or through an external temperature measurement IC over IIC interface as shown in Figure 3-17 [60] [61]. The measured temperature is used to facilitate DTM to control the operating temperature of the FPGA as will be described in chapter 6. When using the MAX6655 IC, It is important to notice that the standby pin should be pulled up high otherwise there will be no temperature readings.

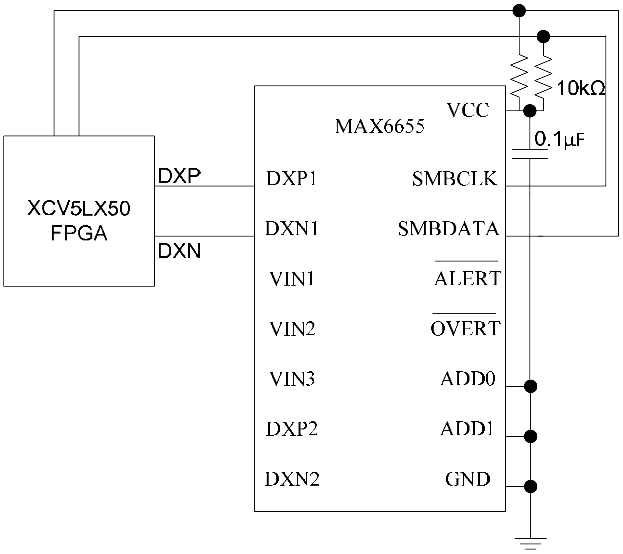


Figure 3-17. Interface with Temperature Measurement IC.

The MSCP can reset any of the slave processors when it does not receive the heart beat signal from it. The slave processors can also reset themselves through their local watchdog timers as a backup to the reset by the MSCP. Whenever any of the slave processors is reset a (RESET) message is sent through the mailbox after boot-up of the slave processor to the MSCP. The slave processor which was just reset would copy the current state of the other processors from the MSCP which saves a copy regularly of the slave processors essential data. The code written on each processor is recommended to be in a state machine form where the state code, input vector, output vector and variables' values can be stored in one task stack and exchanged with the MSCP to store it as a system restoration point. To store the recovery data, the MSCP has an external MRAM which acts as a flash memory and SRAM at the same time. The interface with the external MRAM is carried out through the External Memory Controller IP core (EMC).

The processors exchange data with each other to make sure that their calculations are correct. They make use of the TMR with feedback concept. Figure 3-18 shows the concept of data exchange among the processor nodes in the MPSoC OBC to execute TMR with feedback. Whenever a slave processor generates a data vector, it exchanges it with the other user slave processors, to execute voting according to Eq. (3-8) .

$$\mathrm{Data}_{\mathrm{vote}} = \mathrm{Data}_1 \,\&\, \mathrm{Data}_2 + \mathrm{Data}_2 \,\&\, \mathrm{Data}_3 + \mathrm{Data}_1 \,\&\, \mathrm{Data}_3$$

(3-8)

The voting is bitwise so that the logic operations of the (AND) and (OR) in Eq. (3-8) are carried out on the corresponding bit locations in the three (Data) vectors.

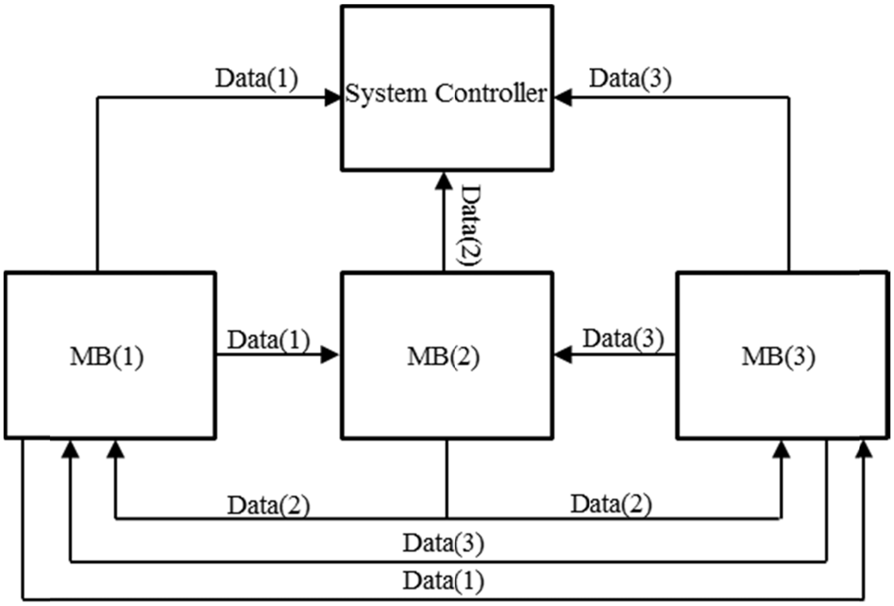


Figure 3-18. TMR with Feedback among the Data of the MPSoC Nodes.

The copies of the source data which were voted at each slave processor are sent to the MSCP to vote on it. The MSCP saves the voting result in its MRAM as a restoration point for slave processor(s) recovery in case of failures. The voting process is carried out through software code at each slave processor as well as the master processor. Thus the voter software function is located within the code running on each of the system processors. In general, the voting process is preferred to be implemented on the data generated by any TMR-based processing logic as shown in Figure 3-19.

The voted data vector is used instead of the locally generated data vector to guarantee correct operations based on results consensus. In space redundancy the consensus comes from voting on the redundant units. In time redundancy, the consensus comes from voting on the results of repeated executions [27] [29]. The data stored at the MSCP MRAM is used as an RB to roll backward the system in case of failures [30]. Creating a restoration point at the MSCP and updating it regularly is called check-pointing [27]. A score-card can be kept for each slave processor at the MSCP. The score card counts the numbers at which each slave processor generated data that had consensus with at least one of the other two processors. The processors are classified as being trustworthy according to their scores. The processor with the highest score records the highest trustworthy.

Therefore, in case of absence of consensus among the three processors, the data from the processor with the highest trustworthy can be used by all of the processors. The best way in case of absence of consensus is to roll back the system via the MSCP.

The process of check-pointing was studied thoroughly to define the basis of theoretical analysis for estimating the optimum check-pointing rate [27]. In our MPSoC design, check-pointing depends on the user applications running on the slave processors. Check-pointing rate changes according to the type of the processes running on each of the slave processors as well as the rate of generating critical new data from these processes. Therefore, the check-pointing is an application dependent process that needs to be optimized based on the specifics of each application.

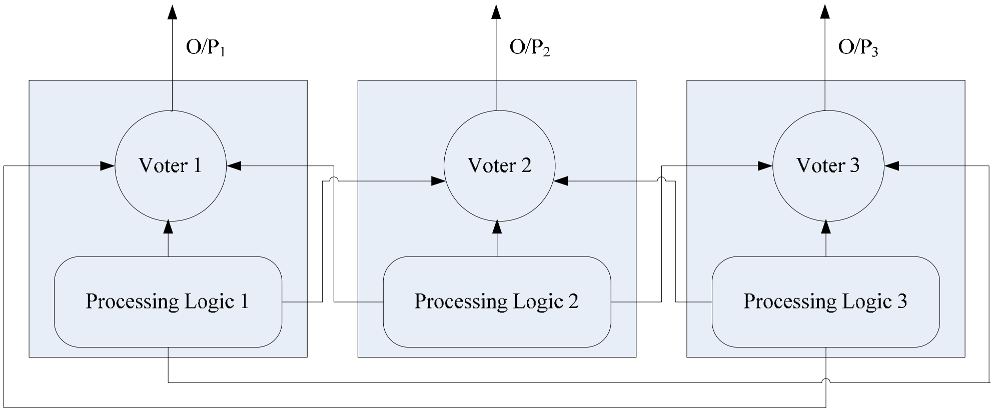


Figure 3-19. Distributed Voting Function at Redundant Processors **[59]**.

The configuration memory of the SRAM based FPGA is scrubbed continuously using the SEU controller IP core provided by Xilinx [17][18]. The SEU controller can be used to inject faults in the configuration bitstream for the purpose of testing as well as scrubbing for correcting SBUs. It cannot handle the correction of MBUs in a single configuration frame.

The configuration frame of the Xilinx Virtex5 LX50 consists of 1312 bits of configuration bitstream [37]. The soft IP core PicoBlaze microcontroller is used within the SEU controller IP core to handle the reading of the configuration frames through the Internal ICAP IP core. The ICAP can directly access the configuration frames SRAM in the FPGA by reading and writing to them. The read frames are sent to an ECC decoder for error checking and correction. The ECC calculates the error syndrome to detect the location of the error bit. Once the error location is detected, it can be corrected by toggling its value. Figure 3-20, shows the major functional blocks used in detecting and correcting the SBUs in the FPGA configuration frames. The SEU controller IP core operates at 50 MHz external clock frequency as recommended by Xilinx.

The frame ECC has the capability of SECDED. In case of double errors detected in any configuration frame, the SEU controller generates an uncorrectable error notification and it automatically stops the Auto Correction Mode (ACM). When MBUs are detected, the FPGA can be left without resetting if the bits were not critical. The critical bit information can be analyzed at design time using Xilinx tools. It is important to notice that not all bits are critical to the design. According to Xilinx less than 20% and typically less than 10% of the upsets are considered as critical and might lead to design failure.

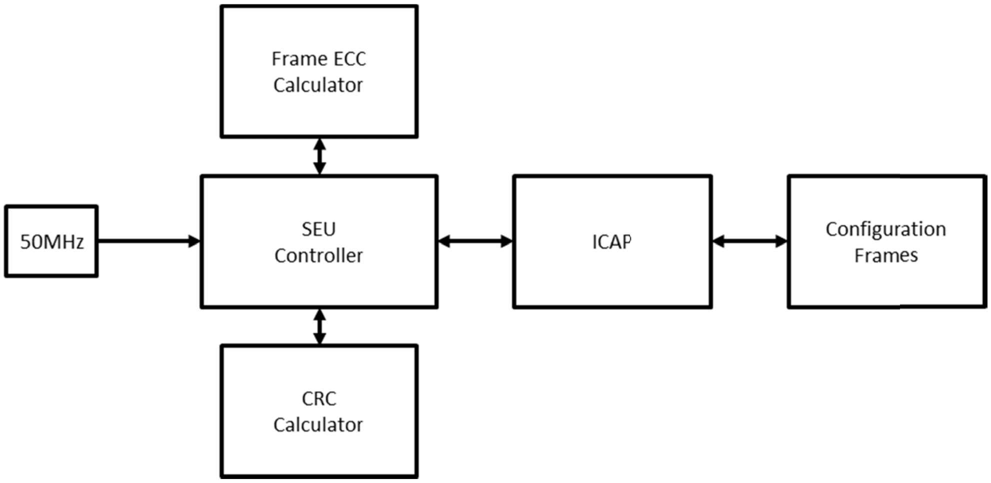


Figure 3-20. Functional Blocks for Correcting SBUs.

The whole bitstream stored in the SRAM of the FPGA is protected by the CRC code which generates an error syndrome whenever the bitstream contains an error. This is necessary to check the integrity of the bitstream as a whole and to know if the errors were corrected whenever they are detected or not. Figure 3-21 shows the scrubbing algorithm to correct SBUs.

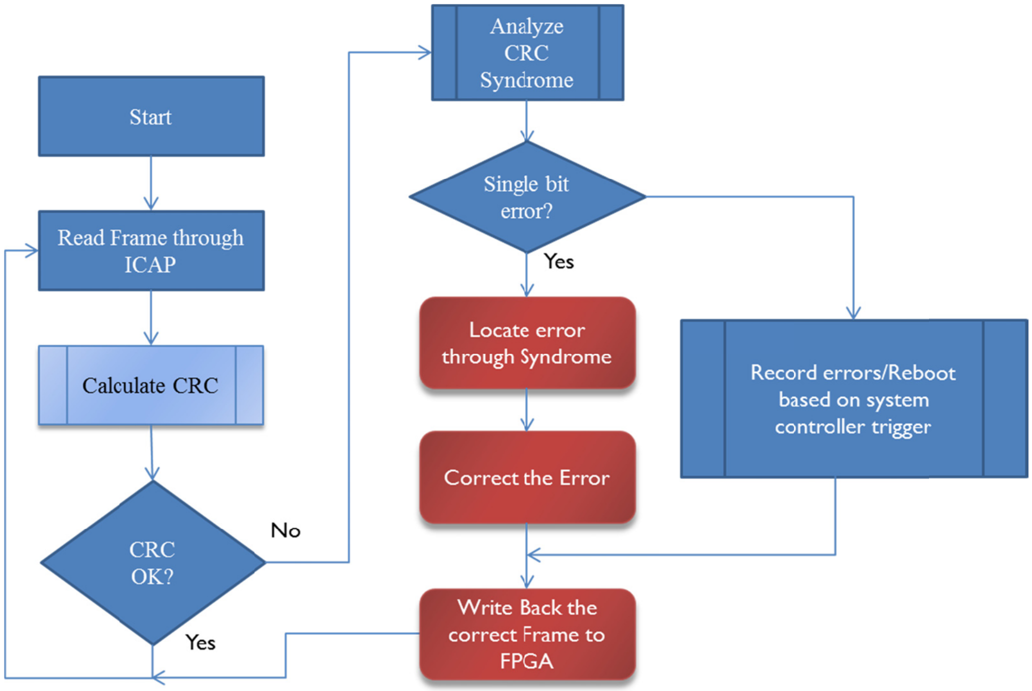

Start
Read Frame through ICAP
Calculate CRC
CRC OK?
No
Yes
Analyze CRC Syndrome
Single bit error?
Yes
Locate error through Syndrome
Correct the Error
Write Back the correct Frame to FPGA
Record errors/Reboot based on system controller trigger

Figure 3-21. Scrubbing Algorithm.

The system resources are estimated based on the Xilinx EDK tool as shown in Table 3-3. The Occupation of the slices in the FPGA is up to 98%. Memory usage is up to 70%. More than half of the LUTs which implement the combinational logic functions are used as well as the registers inside the logic slices.

Table 3-3. FPGA used resources in the MPSoC design.

Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	16,362	28,800	56%
Number of Slice LUTs	18,763	28,800	65%
Number of Occupied Slices	7,076	7,200	98%
Number of BRAM/FIFO	42	60	70%
Total Memory Used (KB)	1,512	2,160	70%

The power consumption, for example when using LX50T, is estimated for the MPSoC design by the XPower analysis tool from Xilinx as shown in Table 3-4. A considerable amount of the power is dissipated in the form of leakage in the 65 nm technology devices from Xilinx. The dynamic power used in the logic gates, signals, BRAMs and IOs is quite few. The second source of high power consumption is the clocks generated in the FPGA then the Phase Locked Loop (PLL) circuitry. The leakage, clocks and PLLs consume about 85% of the total consumed power. Each processor runs at 100 MHz, while if the clock is reduced to smaller value the dynamic power consumption would reduce as well. The total power consumption of the OBC system would be higher than the shown hereinafter as it would include other components rather than the FPGA. Also it would change according to the operating temperature and FPGA size.

Table 3-4. Power Estimation of the MPSoC design.

On-Chip	Multi-core Power(W)
Clocks	0.347
Logic	0.007
Signals	0.014
BRAMs	0.096
DSPs	0.000
PLLs	0.263
DCMs	0.068
IOs	0.002
Leakage	0.460
Total	1.256

The schematic output of the EDK package is shown in Figure 3-22. Four processors are represented in the schematic with the four large rectangles. The rest of components are the peripherals connected to each of the processors. The design in EDK is implemented through 3 steps: 1) Selecting the IP cores to be used from the IP catalog, 2) Connecting the IP cores to the PLB interface, 3) Connecting the IP cores through their ports interface to other peripherals, and 4) Defining the address map of the IP cores in the system. The compilation of the system to generate the bitstream files are done through the Xilinx ISE Design Suite tools.

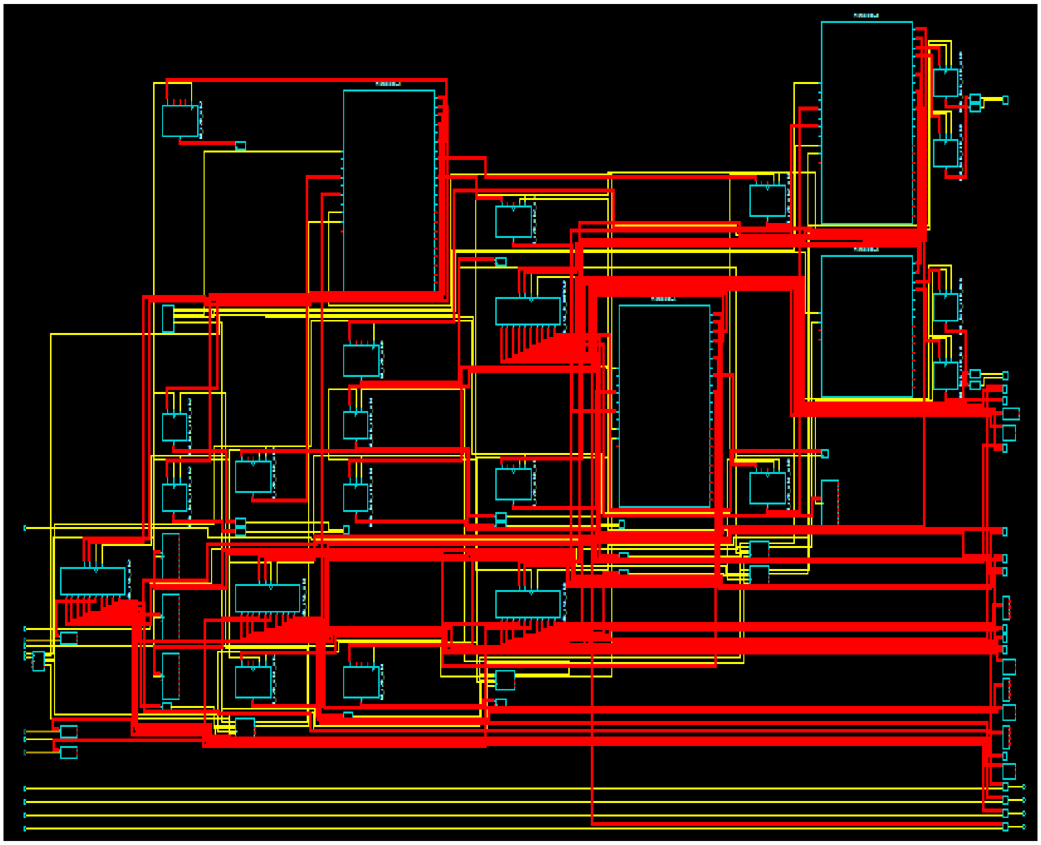


Figure 3-22. Schematic of the MPSoC Design in the XC5VLX50-1FFG676C FPGA.

The hardware bitsteram is exported to the SDK tool for software development. Software is developed for each processor using its Board Support Package (BSP) and software libraries. The BSP contains the device drivers of the peripherals used in the design. The executable files are either programmed to the BRAM or the external memory depending on their sizes. The file generated after the hardware design, which is called the (system.bit), is used by the SDK together with BRAM memory map file (system.bmm) and the executable code files (*.elf) to generate the download file (download.bit) file. Programing of the FPGA can be done through SDK, EDK, ISE or Impact.

The technical specifications of the MPSoC OBC design are shown in Table 3-5. The interfaces can be changed according to the satellite system bus requirements. Typically RS232, IIC and GPIO are used. Other interfaces include Controller Area Network (CAN), MIL-1553 bus and Spacewire. Application memory can be extended however the power consumption would increase as well. We use the MPSM architecture therefore we recommend an external separate MRAM for each of the slave processors. They can use an internal BRAM for booting-up, such as using it for the boot strap purpose. The MSCP can use the internal BRAM as it does not execute large code however it needs an external MRAM for storing the recovery data. The overall power consumption should be less than 5 Watt to be adequate for use in small as well as nano-satellites with mass up to 50 kg and 100 Watt of generated power. Collected scientific data from the satellite payload can be stored via the OBC in the storage memory. The memory size is customizable according to the satellite mission. The power consumed by the storage memory should be considered when estimating the total power consumption of the OBC system. The OBC operating temperature is defined at the interface, attachment point, between the OBC and the satellite chassis. This temperature is used as a basis in the thermal vacuum testing in chapter 6. It should be respected in the thermal design of the whole satellite.

Table 3-5. FPGA Based MPSoC OBC Specifications.

Weight	< 0.5 kg
Power	Total < 5 Watt FPGA 1.256 Watt, MRAM 0.6 Watt x 4, Others 0.3 Watt
Size	10 cm x 10 cm x 0.5 cm
Interfaces	RS232, IIC, GPIO
Application Memory	Int. 32 Kbytes, Ext. MRAM 2 Mbytes
Throughput	100 MIPS x 4 cores
Storage Memory for Scientific Data	Customizable
Number of Processor Cores	4 cores
FPGA Type	SRAM Based
Operating Temperature	-24˚C to +60˚C
Target Orbits	Low Earth Orbit

The total throughput is concerned with the total number of operating cores. The MicroBlaze processor uses Reduced Instruction Set Computer (RISC) architecture. The throughput of RISC processors is almost equivalent to its operating frequency. MicroBlaze processor operating at 100 MHz would have an approximate 100 MIPS throughput. If the processor cores are 4 then the total throughput would be 100 MIPS. This throughput is reached if the processors work cooperatively through load balancing to share the tasks execution. If the processors work individually to achieve

TMR based fault tolerance among three of them and a master control as the fourth processor then the effective throughput would be 100 MIPS. The target orbit was selected to be LEO due to the fact that we are designing the system using commercial grade FPGA. Our radiation analysis and tests in chapter 5 are based on the simulations for a satellite at LEO. Higher altitude orbits require different radiation analysis and higher level of protection against SEEs as well as accumulated doses. These orbits might require a different grade FPGA, typically space grade.

The floor plan of the FPGA as generated by PlanAhead tool from Xilinx is shown in Figure 3-23. The design is using 98% of the FPGA resources. This percentage of utilization would be considered good if there is no intention to put additional logic to implement extra functions in the same package. In case a larger package is needed then the LX85 FPGA can be used to hold the extra design functions but take into consideration that the static and dynamic power consumptions would increase as well. The LX85 contains about double the resources of the LX50.

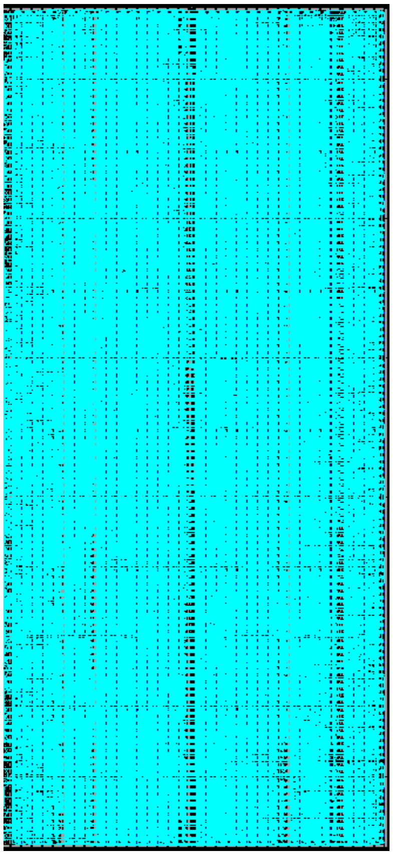


Figure 3-23. FPGA Floor Plan of the MPSoC OBC.

The design Microprocessor Hardware Specification file (MHS) is presented at Appendix (A) and the top level schematic at Appendix (B).

In this chapter we presented the design of the MPSoC OBC. We introduced the taxonomy of processor-memory interface which we call it Abu Zaid Taxonomy of fault tolerant processor-memory interfaces. It was cited at the small spacecraft technology state-of-art report by NASA [14]. We selected the MPSM with non-shared memories architecture and carried out the design based on it.

In the next chapter we introduce a detailed analysis and comparison between the time and space redundancy approaches which can be used in single and multi-processor systems.

Chapter 4 : Space and Time Redundancy Trade-offs

Several techniques can be used to implement fault tolerant OBCs. The time redundancy technique uses a single processor and executes the code for three times to compare between the results of execution to check for consensus. The space redundancy technique uses three hardware units which work in parallel at the same time and finds consensus among their results. It is obvious that time redundancy would have longer time in execution than space redundancy. However, it should have smaller power consumption and smaller area utilization. On the other hand, reliability of the FPGA bitstream against induced soft errors, both in the case of time and space redundancy, depends on the scrubbing rate.

In order to understand the tradeoffs in using space and time redundancy we had to carry out a comparative analysis based on a benchmark. We selected the bubble sort to be used as the benchmark for the performance. As the main function of the OBC is to sort the commands according to their execution times for further execution or dispatching to other subsystems, a sorting algorithm would fit as a benchmark. Bubble sort technique would have execution times that vary based on the effort exerted in sorting the data. Therefore, by generating random vectors of unsorted data we guarantee to have random execution times. The execution times are then grouped to get the statistical distribution.

It is important to notice that our MPSoC design uses space redundancy between the slave processors to implement TMR with feedback. However, in the master processor, time redundancy is used as there is only one instance of the MSCP.

The test which we conducted was comparing the single processor time redundancy system and the triple processors space redundancy system from four perspectives:

1) Performance in executing the bubble sort algorithm.

2) Power consumption both static and dynamic.

3) FPGA area utilization.

4) Soft error reliability.

The Xilinx Virtex-5 XC5VLX50T FPGA in the Gensys kit from digilent was used in the space and time redundancy evaluation as the development kit had multiples of interface options. However, The LX50T and LX50, which is used in the actual MPSoC, are of the same speed and both are of commercial grade. They have the same number of CLB arrays, 6-input LUTs, distributed RAMs, shift registers and flip flops. In this chapter we study the details of time and space redundancy trade-offs and analytically estimate the in-orbit soft error reliability. That study would provide an understanding of the differences between space and time redundancy when implemented in 65 nm SRAM-based FPGAs. It would help in selecting the suitable fault tolerance approach according to the operating conditions. Some sections of this chapter were part of the author publications at [62].

4.1 Single Processor Design

Two separate systems were implemented to test the trade-offs of using time and space redundancy in the Virtex- 5 LX50T FPGA: 1) single processor system, and 2) Multi-Processor system. The Multi-Processor system design is identical to the MPSoC previously described in section 3.5. In the single processor system, as shown in

Figure **4-1**, a MicroBlaze Processor is connected to a Local Memory Bus (LMB) where a local BRAM is attached. The processor is connected to other peripherals through the PLB. A WDT is used to reset the system in case the processor stopped working. The processor receives an interrupt from the INTC that the WDT finished counting and should be reinitialized. If the processor was working and did not hang up, it will respond to the WDT interrupt and will reset it. In case the processor stopped working for any reason, it will not respond to the WDT interrupt. The WDT will send a reset request to the reset module. The reset module will then reset the whole system including the processor.

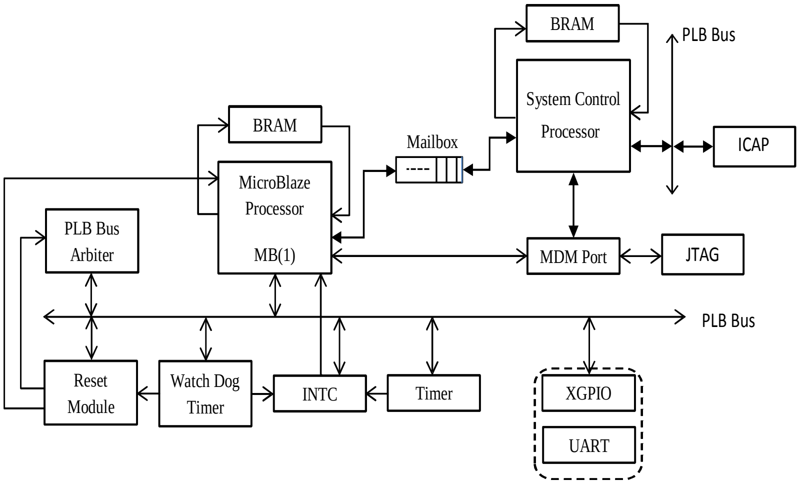


Figure 4-1. Single Processor System.

The system contains a system control processor to handle the check pointing of the current status, and restoring the system operation to the last known good status in case of reset. The communication between the system control processor and the MicroBlaze processor takes place through the mailbox IP core. Two Timers exist in the same timer IP core, one of them is used for measuring the execution time of the code in this experiment. The MDM port is used to debug the software application running on the MicroBlaze processor and the system control processor. Xilinx General Purpose Input Output (XGPIO) IP core is used to input and output digital signals. It provides a control interface to the outside world. UART is used to send and receive serial streams to and from the processor.

Table 4-1. Single Core and Multi-Core Power Consumptions.

On-Chip	**Single Core Power (W)**	**Multi-core Power (W)**
Clocks	0.170	0.347
Logic	0.003	0.007
Signals	0.005	0.014
BRAMs	0.010	0.096
DSPs	0.000	0.000
PLLs	0.263	0.263
DCMs	0.068	0.068
IOs	0.001	0.002
Leakage	0.458	0.460
Total	0.978	1.257

Table 4-2. Single Core Resources Utilization.

Slice Logic Utilization	**Used**	**Available**	**Utilization**
No. of Slice Registers	5,932	28,800	20%
No. of Slice LUTs	6,866	28,800	23%
No. of Occupied Slices	3,337	7,200	46%
Number of BRAM/FIFO	18	60	30%
Total Memory Used (KB)	648	2,160	30%

Table 4-3. Multi-Core Resources Utilization.

Slice Logic Utilization	**Used**	**Available**	**Utilization**
No. of Slice Registers	16,362	28,800	56%
No. of Slice LUTs	18,763	28,800	65%
No. of Occupied Slices	7,076	7,200	98%
Number of BRAM/FIFO	42	60	70%
Total Memory Used (KB)	1,512	2,160	70%

Table 4-1 shows the power consumption of single and multi-core designs. Table 4-2 shows the resources utilization of the single core system. Table 4-3 shows the resources utilization of the multi-processor system.

4.2 Time and Space Redundancy Check Bench

The test configuration is shown in Figure 4-2, a random number stream of specified length, in this case it is 100 numbers, is generated by random number generation in MATLAB. It is then sent to the processors through the serial port RS232 interface. The processors save the received vector and wait for a signal to start the bubble sort algorithm. When the start signal is issued by the MATLAB script, the processors start their local timers to calculate their execution speeds and then initiate the Bubble sort algorithm execution.

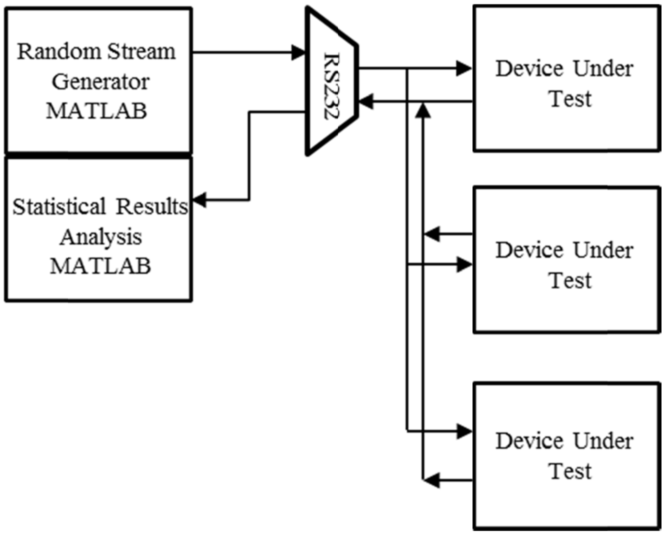

Random Stream Generator MATLAB
Statistical Results Analysis MATLAB
RS232
Device Under Test
Device Under Test
Device Under Test

Figure 4-2. Time and Space Redudnacy Test Configuration.

After the bubble sort finishes execution on each processor, the processors exchange the values of the sorted vectors among each other through the inter-processor mailboxes. Cross voting takes place at each processor. The local timer at each processor is stopped after the cross-voting and the execution time of each processor is sent to the MATLAB script. The test cycle continues until the required numbers of trials are executed.

In case of single processor, it stores three copies of the data vector in its local memory. The bubble sort function runs three times and the sorted vectors are stored in three different memory locations. The voting takes place between the vectors stored in the local memory. The values of the execution times, after finishing the voting process, are sent from the processor to the MATLAB script. The execution times of each processor are statistically analyzed by the MATLAB script to calculate the mean and the variance of the execution times. Figure 4-3 and Figure 4-4, show the flowchart of the test algorithm in case of single processor and triple processors.

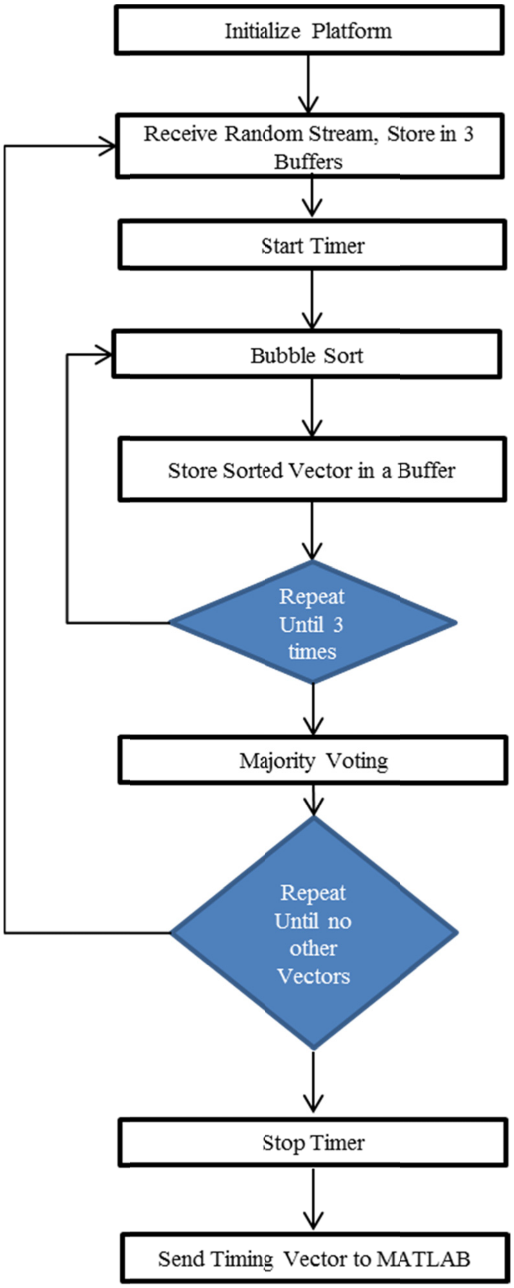

Initialize Platform
Receive Random Stream, Store in 3 Buffers
Start Timer
Bubble Sort
Store Sorted Vector in a Buffer
Repeat Until 3 times
Majority Voting
Repeat Until no other Vectors
Stop Timer
Send Timing Vector to MATLAB

Figure 4-3. Time Redundancy - Single Processor.

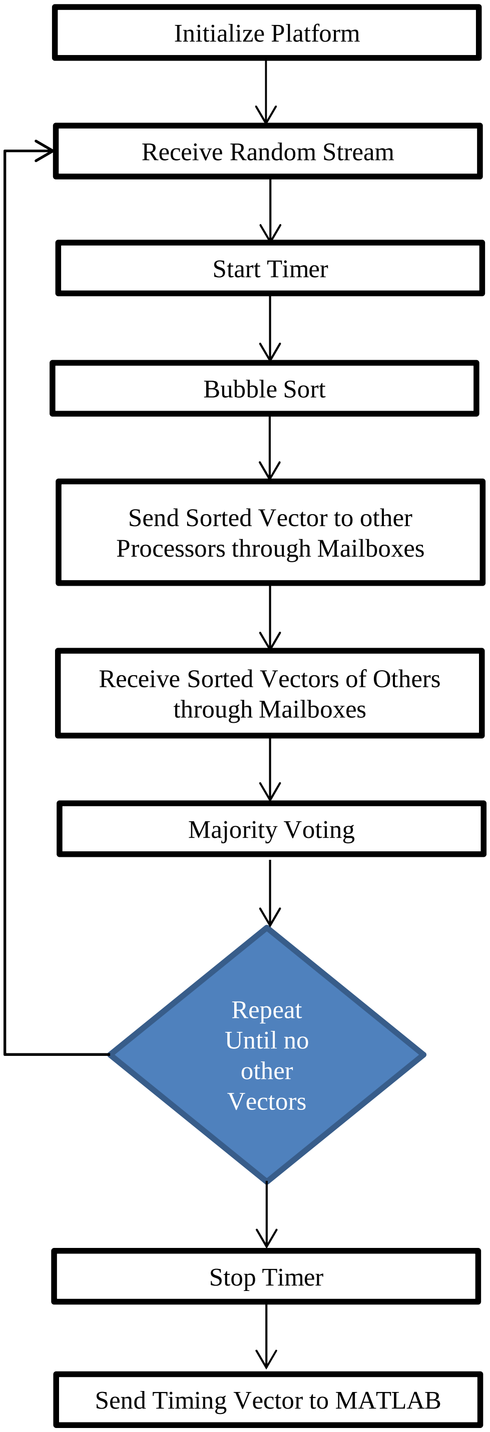

Initialize Platform
Receive Random Stream
Start Timer
Bubble Sort
Send Sorted Vector to other Processors through Mailboxes
Receive Sorted Vectors of Others through Mailboxes
Majority Voting
Repeat Until no other Vectors
Stop Timer
Send Timing Vector to MATLAB

Figure 4-4. Space Redundancy - Triple Processors.

The results of execution and comparison between the single and triple processors systems in terms of time and space redundancy are presented in chapter 7. Next we present how to evaluate the soft error reliability for both systems.

4.3 Estimation of Soft-Error Reliability

Investigation of in-orbit SEU rates should be used when estimating the reliability of the design against soft errors in the configuration bitstream. SEU rates can be accurately investigated using accelerator radiation testing however it needs complex test procedure and setup. The Xilinx Virtex-5 XC5VLX50T FPGA in the Gensys kit from digilent was used as an example to estimate the soft error reliability. It consists of approximately 11.37 Mbits of effective configuration cells. The device contains 355,190 configuration words and each word contains 32 bits (total of 11,366,080 bits). The procedure hereinafter is general and can be used for any SRAM-based FPGA device. In order to approximately estimate the reliability of the design, in the worst case, we multiply the percentage of used resources by the device size in Mbits. In calculating the effective configuration bits size of the XC5VLX50T FPGA device, only relevant configuration cells which would remain unchanged are included. No block memory contents are included in the calculation as their data might change during the design operation. Therefore the (effective configuration image size) is 11.37 Mbits without the block memory contents. The full configuration image size which contains the block memory contents is larger than 11.37 Mbits.

To estimate the expected reliability of the Virtex-5 XC5VLX50T device, as an example, while operating in space, we apply the following steps:

1) Define a target orbit upon which the estimations would take place. Use the target orbit data to calculate the trapped proton and electrons spectrum using the AP-8 and AE-8 models.

2) Use the static proton and heavy ion test results of the device, if they exist, or use the results of another device from the same technology, to estimate the (in-orbit SEU rates) using the Cosmic Ray Effects on Microelectronics (CREME-96) model within the Space Environment Information System (SPENVIS) online package.

3) Calculate the design specific SEU rate, soft error rate in design data, based on the design size in bits.

4) Calculate the system failure rate based on the design specific SEU rate. Assume

that each SEU would cause a failure in the system operation. According to Xilinx, this is a very pessimistic assumption as less than 20%, and typically less than 10%, of the configuration bits would cause design failures if they are subjected to upsets.

5) Reliability estimation based on the soft error failure rate (λ) and the time between scans of the FPGA configuration memory (T).

4.3.1 Target Orbit Definition

We select an orbit that is similar to the missions launched by Kyushu Institute of Technology to demonstrate the estimation procedure of the expected system reliability. Any other orbit can be selected depending on the target mission requirements. The orbit definition affects the expected charged particles fluxes. Table 4-4 shows the orbital parameters of the Horyu-2 satellite which is used as an example to carry out the reliability calculations.

Table 4-4. Horyu-2 Satellite Orbital Parameters.

Orbit Type	general
Perigee Altitude	651 km
Apogee Altitude	671.6 km
Inclination	98.17°
RAAN	223.04°
Argument of Perigee	31.95°
True Anomaly	328.25°

We use SPENVIS online package [63] to calculate the fluxes of the trapped protons and electrons, solar particles, and galactic cosmic ray in the chosen mission orbit. The output of the AE-8 and AP-8 models with solar maximum condition is shown in Figure 4-5 and Figure 4-6. It is clear that higher fluxes are focused at the South Atlantic region in what is known as the SAA. The spectra of the trapped electrons and protons energies versus the flux are shown in Figure 4-7 and Figure 4-8.

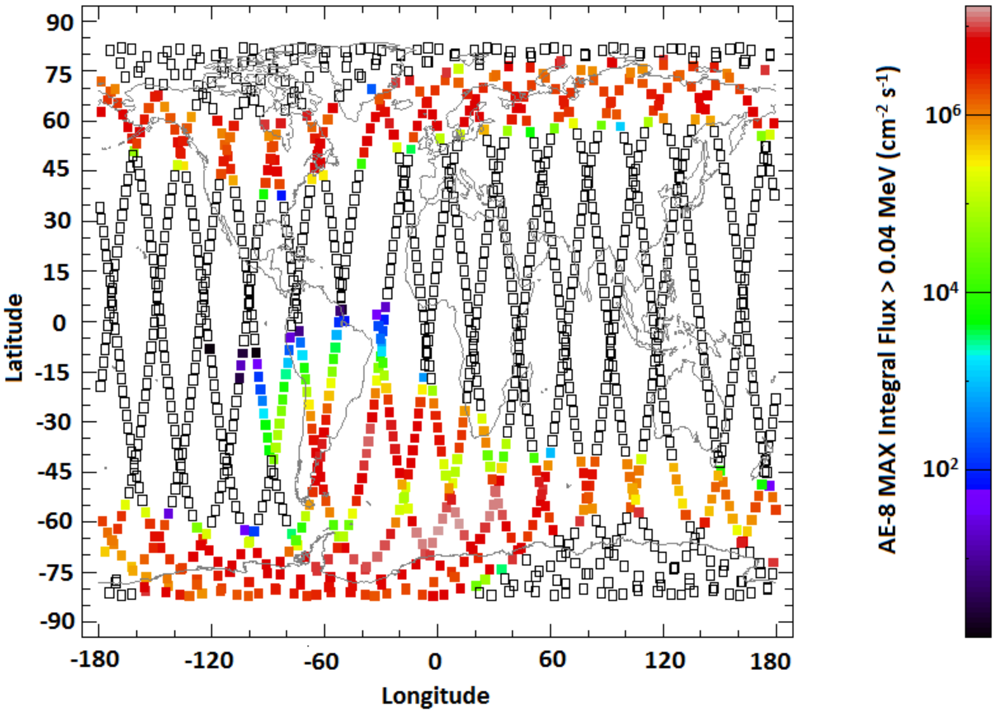


Figure 4-5. AE-8 Model Trapped Electrons Flux in the Orbit.

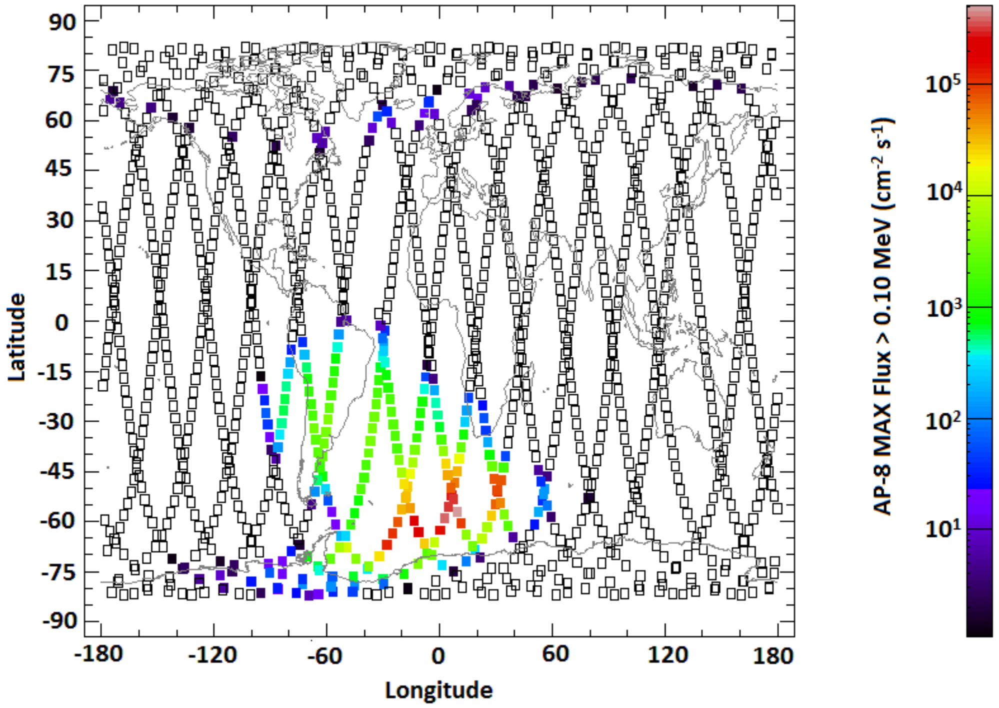


Figure 4-6. AP-8 Model Trapped Protons Flux in the Orbit.

The flux of the trapped electrons energy above 6 MeV is very low. It is almost less than 1 cm^{-2}sec^{-1}. The flux of the trapped protons energy at about 300 MeV is about 10 cm^{-2}sec^{-1}. Therefore, the expected effect of the trapped protons in causing bit upsets is higher than the effect of the trapped electrons. The threshold energy level at which the bit upsets are noticed depends on the specific device technology and can be found through radiation experimental results, static cross section estimation.

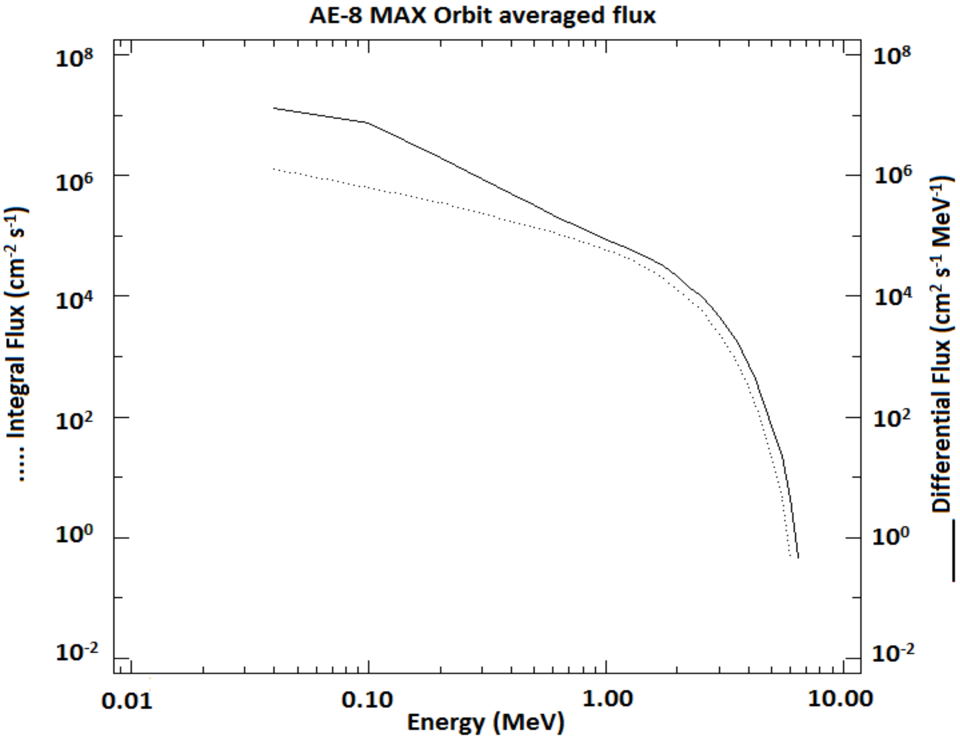


Figure 4-7. Trapped Electrons Energy Spectra Versus the Flux.

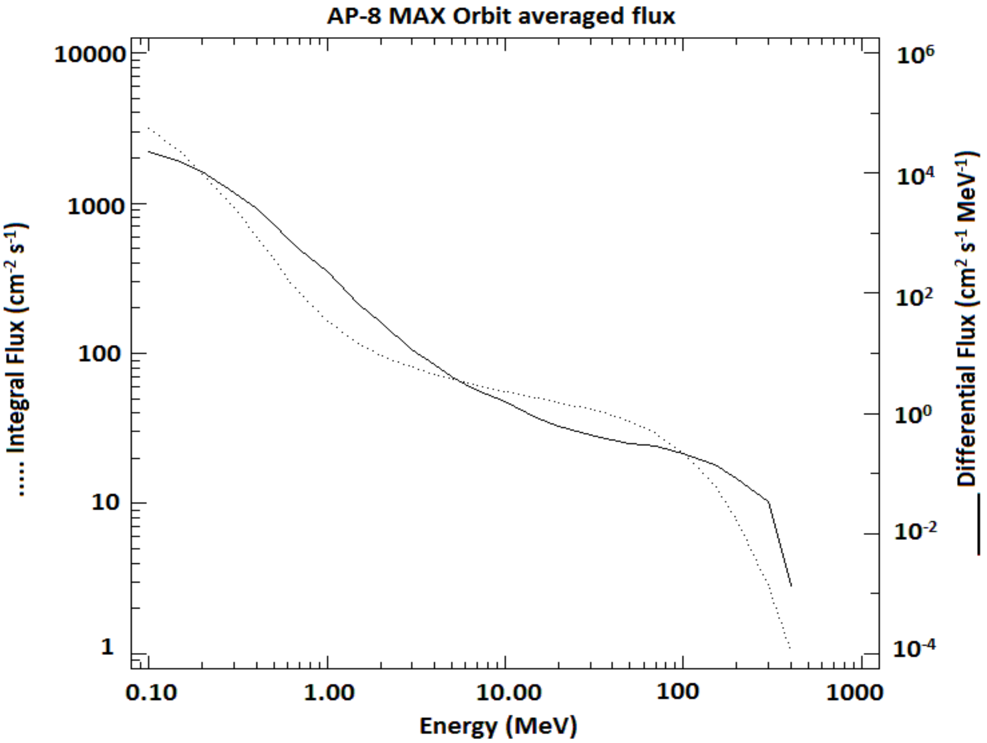


Figure 4-8. Trapped Protons Energy Spectra Versus the Flux.

4.3.2 SEU Rate Estimation

After defining the target orbit and estimating the trapped protons and trapped electrons fluxes at different energies using the AP-8 and AE-8, we use the output to estimate the SEU rate using CREME-96 model in SPENVIS online package. When running the simulations, we assumed that the satellite has an Aluminum shielding thickness of 0.5 g/cm^2. We used the real experimental results of the static proton and heavy ion testing of the XC5VLX50 FPGA device by NASA as inputs to the CREME-96 model, in order to estimate the device specific upset rates.

The system was built on the XC5VLXT50 FPGA device. However, both the XC5VLX50 and the XC5VLXT50 FPGA devices are identical in the number of slices except that the XC5VLXT50 FPGA contains some extra hard cores for advanced serial connectivity. The XC5VLX50 FPGA proton event bit cross-section is 8.61x10^{-14} ± 4.90x10^{-16} cm^2/bit for 200 MeV and 6.37x10^{-14} ± 1.17x10^{-15} cm^2/bit for 65 MeV [64].

The heavy ion event bit cross-section test results were fit using weibull curve. It is calculated from the number of single ionized particle-induced events in the data set (i.e., each MBU counts as one event) [64]. The four parameters of the weibull curve are the saturation cross-section or limit (σ_{lim}), the threshold or onset energy (E_o), the width of the rising portion of the curve (W) and the power that determines the shape of the curve (S). From [64] the values of the four parameters are (σ_{lim}) or (L=5.73E-8) cm^2/bit, (E_o) or (L_o=0.5) MeV.cm^2/mg, (W=15) MeV.cm^2/mg and (S=1.5). The heavy ion event bit cross-section for the highest tested LET of 68.3 MeV.cm^2/mg is $5.73x10^{-8}cm^2$/bit [64].

The SEU rate along the first 14 orbits, about 23 hours of flight, is shown in Figure 4-9. The peaks represent the rate at the SAA region. We calculate the worst case reliability based on the SEU rate at the SAA regions of the orbit (highest peak value corresponds to worst case) as well as the non-SAA regions (rest of the orbit). The highest SAA SEU rate is 3.73E-10 $bit^{-1}sec^{-1}$ and the non-SAA SEU rate is 2.07E-10 $bit^{-1}sec^{-1}$.

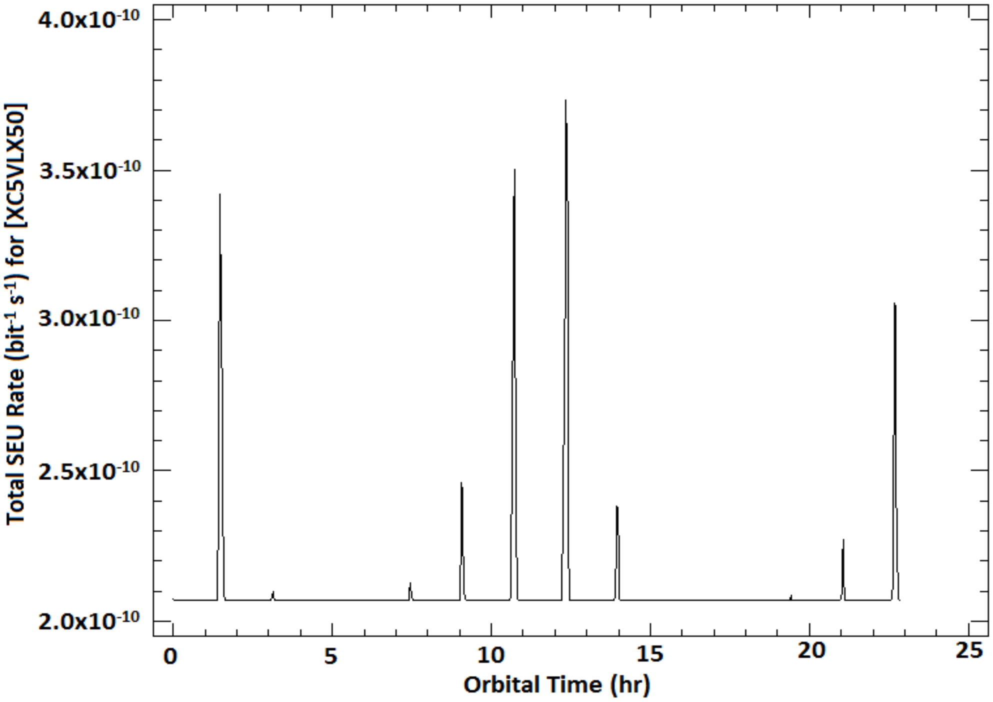


Figure 4-9. In-orbit SEU Rates Estimated by CREME-96 and based on Real Radiation Test Results of the Xilinx XC5VLX50 FPGA.

4.3.3 Design Specific SEU Rate

The single core and multicore embedded systems slices utilization, design sizes and upset rates are shown in Table 4-5. Each MicroBlaze system occupies 26% of the FPGA slices. The system control logic occupies 20% of the FPGA slices.

Table 4-5. Design Size in bits and the Upset Rates.

		% of Occupied Slices	**SEU rate at SAA sec^{-1}**	**SEU rate at non-SAA sec^{-1}**
Single core	**System control**	20%	0.0008479	0.00047056
	MicroBlaze system	26%	0.0011023	0.00061172
	Total occupied slices	46%	0.0019502	0.00108228
Multi core	**System control**	20%	0.0008479	0.00047056
	3 MicroBlaze systems	3 x 26%	0.0033068	0.00183517
	Total occupied slices	98%	0.0041548	0.00230572

The design specific SEU rate is calculated by multiplying the in-orbit SEU rate at the SAA and non-SAA regions ($bit^{-1}sec^{-1}$) by the design size (bits) which is estimated from the (% of occupied slices × FPGA bitstream size (11.37 Mbits)).

4.3.4 System Failure Rate

Assuming that each upset event would cause a system failure, which is a very pessimistic approach, the system failure rate per day can be estimated from the SEU upset rate per sec as in Eq.(4-1):

$$System\_Failure\_Rate\_per\_day = SEU\_Rate \times 86400 \quad \textbf{(4-1)}$$

Typically less than 20% of the design bits would be sensitive to the design operation. Table 4-6 shows the system failure rate calculated for the system control processor and each MicroBlaze processor system at the SAA and non-SAA regions. The failure rates are identical for the system control logic and the MicroBlaze processor system in the single and multicore designs as they occupy the same size of the bitstream in both designs. Calculations are carried out for two cases: 1) when considering that any bit upset would cause a failure and 2) when assuming that only 20% of the bit upsets would cause a failure.

Table 4-6. System Failure Rates.

	Failure rate at SAA **day^{-1}**	**(20% Sensitive Bits) Failure rate at SAA** **day^{-1}**	**Failure rate at non-SAA** **day^{-1}**	**(20% Sensitive Bits) Failure rate at non-SAA** **day^{-1}**
System control	73.26	14.65	40.66	8.13
MicroBlaze system	95.24	19.05	52.85	10.57

4.3.5 Reliability Estimation

Configuration memory scanning is performed to make sure that the memory which contains the design bitstream is free of errors. Configuration frames are readback through the ICAP. Each frame has 12 bits of ECC which is capable of SECDED. The frame data is checked through an ECC calculator to detect if an error exists or not. In case a single bit error exists the error syndrome would locate the error position. The error can be corrected by toggling the bit value at the position identified by the syndrome. If double errors or more exist in a single frame (i.e., MBU) then the error cannot be corrected. The FPGA should be reset if the bit is sensitive to the design. Xilinx also provides a CRC primitive to check the integrity of the whole bitstream image as well as an SEU controller IP core for bitstream scrubbing. The Scrubbing process consists of (scanning) reading back the frames, calculating the ECC, correcting single bit errors, and writing back the correct frames. The scan period is calculated from Eq.(4-2):

$$Scan\_Period = Total\_Configuration\_Words \times \frac{1}{Clk\_freq} \tag{4-2}$$

The maximum scanning rate per day is calculated from Eq. (4-3):

$$Max\_Scan\_Rate\_per\_day = \frac{86400}{Scan\_Period\_in\_seconds}$$

(4-3)

The XC5VLX50T FPGA contains 355,190 configuration words. When operating at 50 MHz scanning clock then the scanning period according to Eq. (4-2) is (scan_period = 7.1038 msec). The maximum scan rate per day is 12,162,504.58 scans. This rate is for continuous scanning where a new scan is started just after the previous scan finishes. According to Xilinx, the scanning rate is recommended to be at least 10 times the upset rate. As scrubbing takes place, we assume the maximum lifetime of any failure to be equal to the time between complete scans, call it the scan cycle time. The reason for this assumption is due to the fact that the errors in the configuration stream would be discovered during the scanning process and corrective action would be taken, either by resetting the FPGA or correcting the upset if possible. Therefore the lifetime of any error is limited by the scan cycle time. The basic formula to estimate the reliability is shown in Eq. (4-4):

$$R = e^{-\lambda T}$$

(4-4)

Where (λ) is the soft error failure rate in unit of time and (T) is the maximum soft error lifetime which is equal to the time between scans. The reliability of the single core system is estimated based on Eq. (4-5):

$$R_{single\_core} = R_{system\_control\_logic} \times R_{MicroBlaze\_system}$$

(4-5)

The reliability of the multicore system is estimated from Eq.(4-6) and Eq.(4-7):

$$R_{multicore\_system} = R_{system\_control\_logic} \times R_{TMR\_MicroBlaze}$$

(4-6)

$$R_{TMR\_Microblaze} = 3R^2_{Microblaze\_system} - 2R^3_{MicroBlaze\_system}$$

(4-7)

From Table 4-6, the highest failure rates are for the system control logic and the MicroBlaze system at SAA which are 73.26 and 95.24 failures/day respectively. These failure rates correspond to the worst case reliability estimation. The lowest failure rates for the system control logic and the MicroBlaze system at the non-SAA region, when only 20% of the design bits are considered to be sensitive for bit upsets, are 8.13 and 10.57 failures/day respectively. These failure rates correspond to the best case reliability estimation. In Figure 4-10, Figure 4-11, and Figure 4-12 we plot the reliability curve of the system at the worst case estimations.

The configuration memory scanning rates are selectable and can be designed to achieve the required system soft error reliability. If we use the SEU controller IP core from Xilinx, then we would have continuous scanning, thus the time between scans would be 7.1038 msec as calculated from Eq.(4-2). At this scanning rate the reliability of the single core and multicore systems is equal to 0.999999 as shown in Figure 4-12. If a custom configuration stream scanning technique was used through the internal ICAP or the external SelectMAP interface then we can estimate the reliability of the single and multicore systems based on Figure 4-11. For example when the time between scans equals 20 sec, then the single core system reliability is about 0.96 while the multicore system reliability is about 0.98. It is important to notice that the multicore TMR system provides better reliability than the single core system till a certain lifetime (time between scans) , as shown in Figure 4-10 which is equal to about 606 sec in this design. After that time the single core reliability becomes worse than the multicore TMR reliability. Therefore, it is always desired to select the time between scans to be lower than that value.

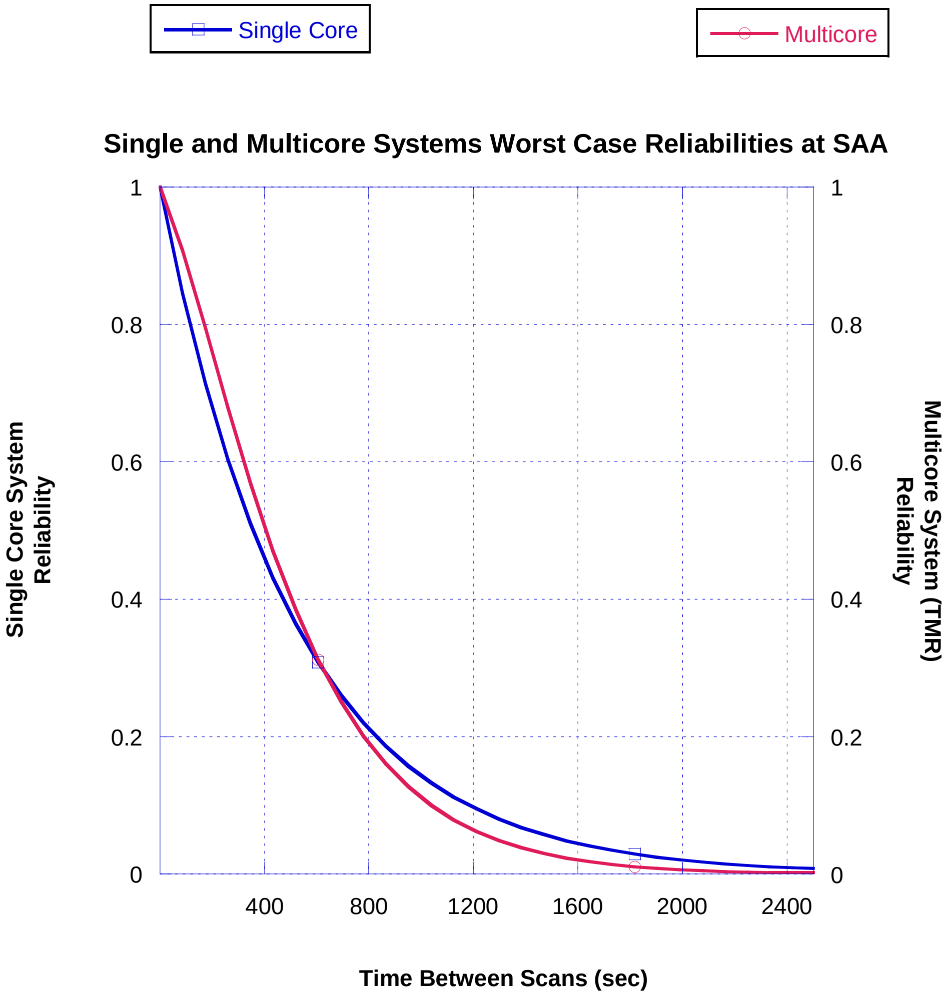


Figure 4-10. Single Versus Triple Processor Systems Soft-Error Reliability based on in-orbit Estimation.

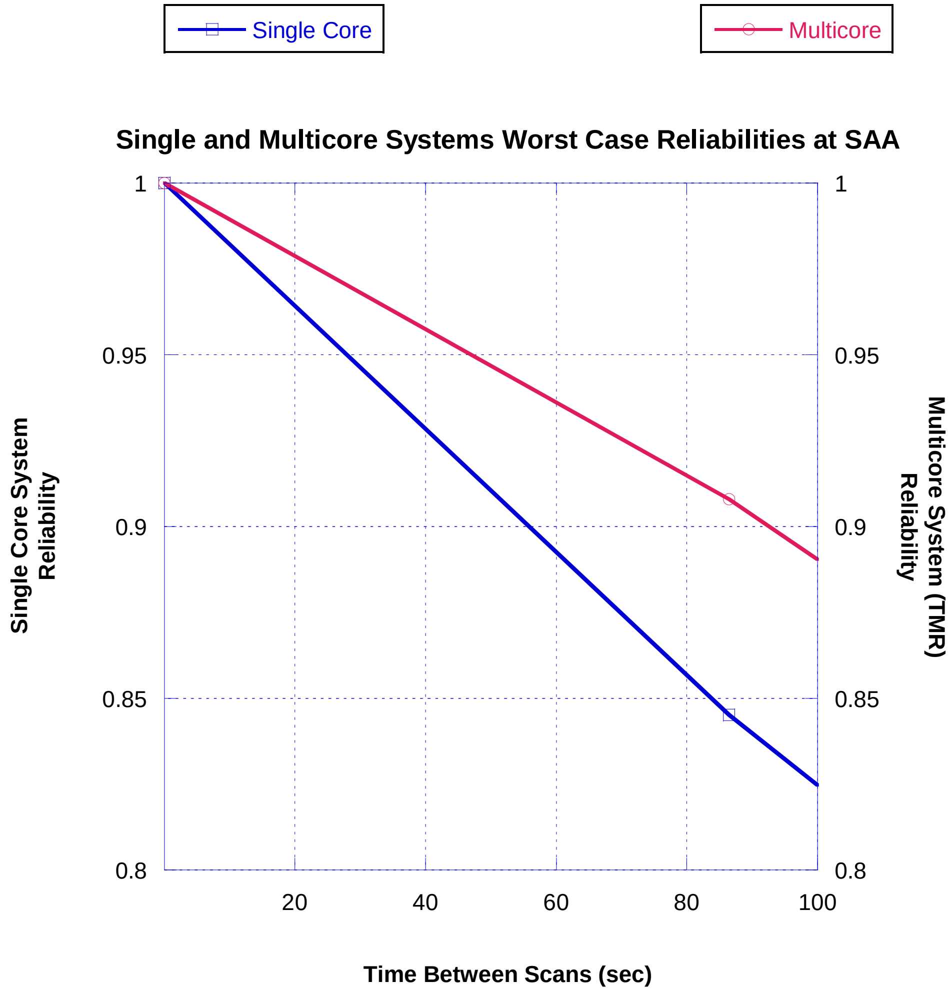


Figure 4-11. Single Versus Triple Processor Systems Soft-Error Reliability based on in-orbit Estimation, Time between Scans ≤ 100 sec.

If an external MRAM is used for storing code or as a platform configuration flash then the overall system reliability is estimated from Eq.(4-8).

$$R_{system\_soft\_error} = R_{MRAM\_soft\_error} \times R_{FPGA\_design\_soft\_error} \quad \textbf{(4-8)}$$

Where, $R_{MRAM\_soft\_rate}$ is the external MRAM memory soft error reliability while $R_{FPGA\_Design\_soft\_error}$ is the soft error reliability of the FPGA digital design. From the previous analysis, it is clear that increasing the scrubbing rate of the FPGA configuration stream leads to higher system reliabilities. Also, we noticed that the TMR multicore system provides better reliabilities than the single core system, in case non-continuous scanning is used, as long as it is below a certain value for the time between scans. As an example, if we assume the time between scans to be 20 sec, and MRAM soft error rate to be 0.05 failure/year then we can plot the total TMR system reliability as in Figure 4-13.

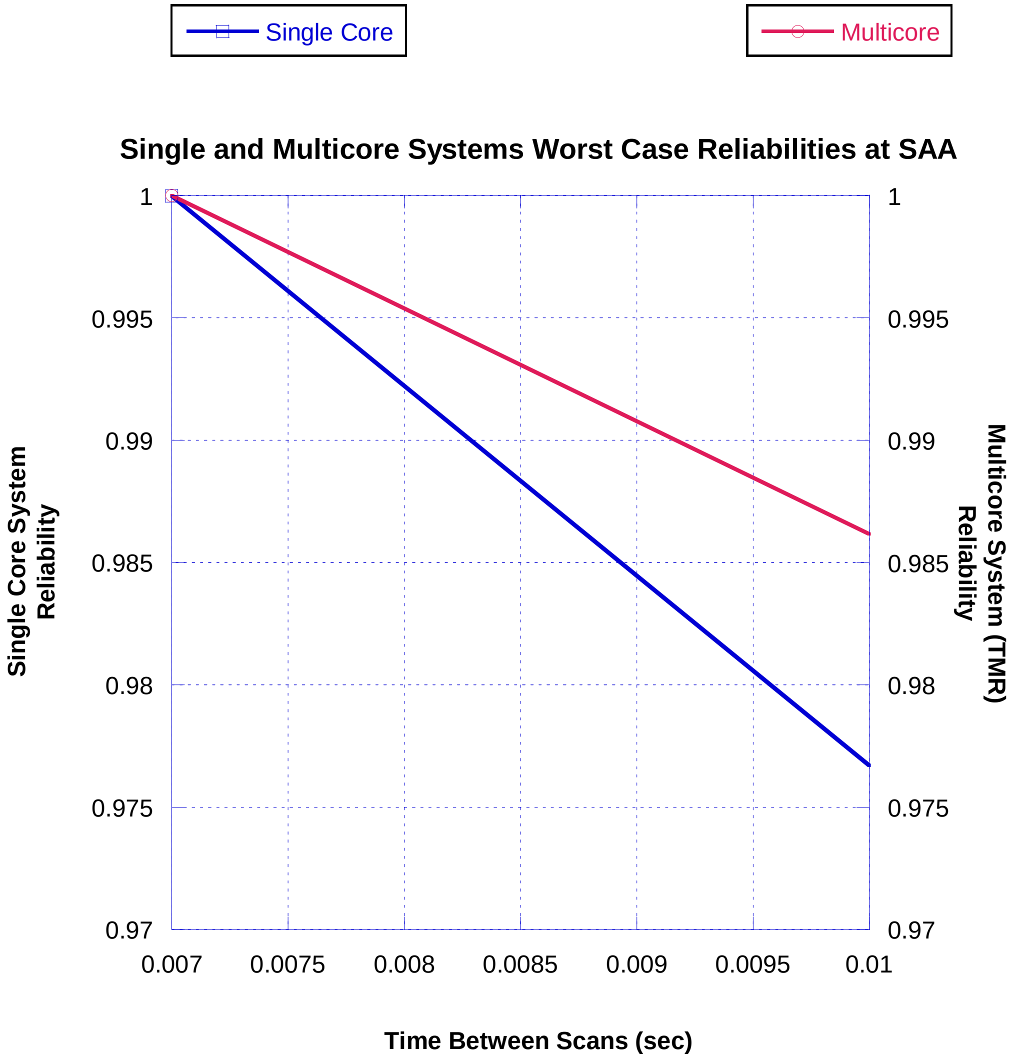


Figure 4-12. Single Versus Triple Processor Systems Soft-Error Reliability based on in-orbit Estimation, Time between Scans ≤ 10 msec.

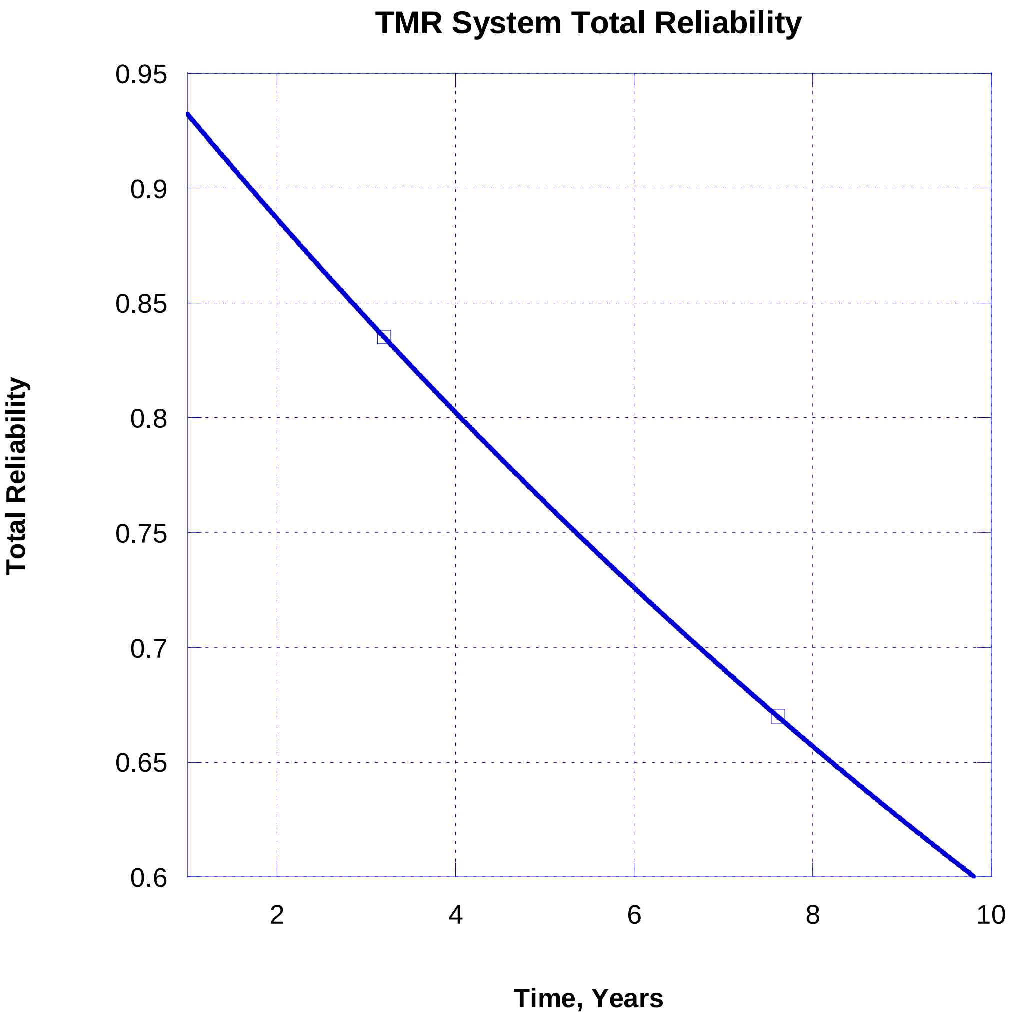


Figure 4-13. Total System Reliability of TMR System based on in-orbit SEU Rates and assuming an MRAM Soft Failure Rate of 0.05 failures/year.

In this chapter we conducted a tradeoff comparison between the time and space redundancy fault tolerance architectures. We compared the architectures in terms of the performance, power consumption, utilization and reliability. The results are presented in chapter 7.

In The next chapter, we handle the radiation test at the nuclear accelerator to estimate the static cross section using a new method through internal scrubbing. Also, we present the fault injection method to estimate the design dynamic cross section.

Chapter 5 : Radiation and Fault Injection Testing

In this chapter we introduce the radiation tests performed on the FPGA chip to estimate its static cross section and the fault injection tests to estimate its dynamic cross section. The Radiation tests were performed at TARRI - JAEA using proton beam at 65 MeV. At the beginning, we decided to perform the radiation test at Kyoto University Research Reactor Institute (KURRI) using Cf-252 as a radioactive isotopic radiation source. Using Cf-252 is much easier in the test procedures, test setup, and less complicated than the proton beam test. However, the de-capsulation of the Virtex-5 chip was not successful. Therefore, we finally decided to use the proton beam for radiation evaluation. We developed a fault injection bench to estimate the dynamic cross section of the design using fault injection in the configuration memory through DPR as will be described in the chapter. The radiation and fault injection test results are used in analyzing the system soft error reliability in a given orbit of choice. Some sections of this chapter were part of the author publications at [65] [66].

5.1 Radiation Test Objectives

Radiation tests are expensive and need a lot of preparation compared to laboratory based tests. If the test was to take place at a nuclear accelerator, as in the case of proton and heavy ion radiation, then the complications would even be greater. Safety is the main issue when conducting radiation tests. Therefore, a lot of measures should be taken to counteract any possibility for radiation contamination. Nevertheless, while radiation testing is still an expensive and complex test, it has some objectives which make it worth to conduct whenever possible. The following are the main objectives of radiation test:

1) Measurement of the static cross section which indicates the sensitivity of the chip’s wafer to radiation induced SEUs.
2) Measurement of the dynamic cross section which indicates the design sensitivity to radiation induced SEUs.
3) Measurement of the SEL incidences and the increment in the level of consumed current.

4) Accumulated dose, TID and DDD, identification and measurement.
5) Testing of the effectiveness of the applied mitigation techniques against SELs, SEUs, TID and DDD.
6) Estimation of the electronic components Soft Error Reliability (SER) and their lifetimes based on SEU rate, TID and DDD.

5.2 Accelerated Proton Beam Radiation Test Setup

Performing radiation test using accelerated proton beam requires special test setup. The Device Under Test (DUT) should be baked before placing it in the radiation chamber in order to reduce the outgassing emissions and hence reach the required pressure to start the radiation test in shorter time. Figure 5-1 shows the baking of the boards before radiation test at CeNT – Kyushu Institute of Technology (KIT). The boards are then kept in nitrogen sealed containers to avoid any water vapor condensation on their surfaces, which would require repeating the baking step, as well as isolating them from any contamination from the surrounding environment.

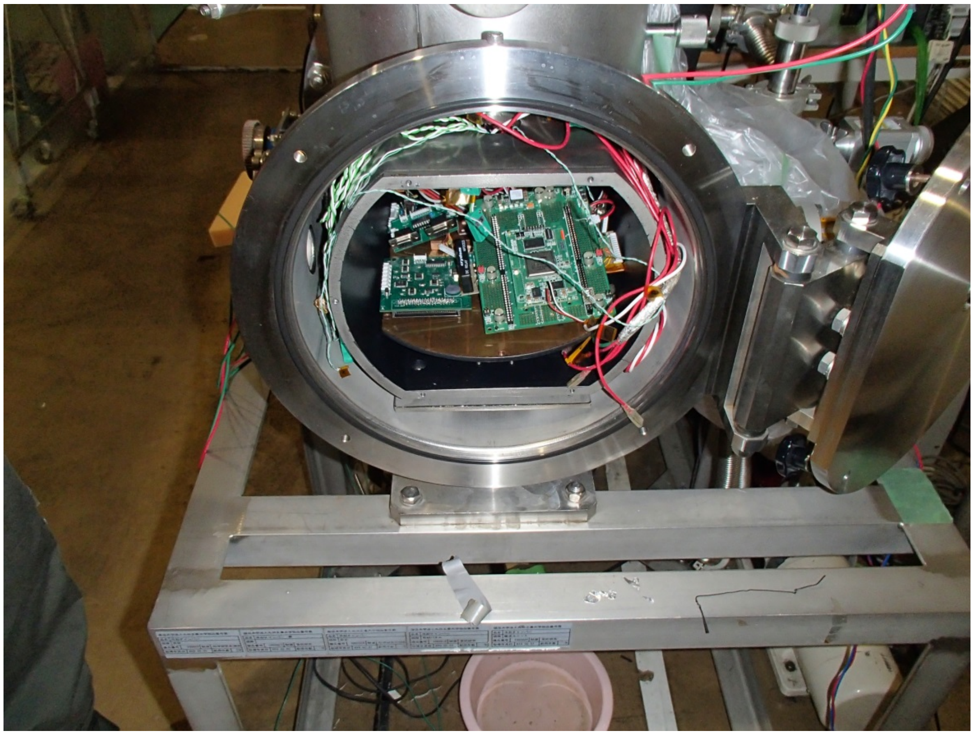


Figure 5-1. Baking of Boards before Radiation Testing.

During the radiation test no one is allowed to exist near the test chamber due to safety conditions. The test chamber, shown in Figure 5-2, is vacuumed at a pressure of 10^{-5} Pa. The DUT is placed in a vertical orientation to face the proton beam. The pressure level of 10^{-5} Pa guarantees that no molecular interaction would take place between the proton beam and any molecules that might exist inside the chamber as they would be sucked out. The chamber is located away from the control room and protected with a concrete wall surrounding it from all sides. The concrete wall has a thickness of about 4 m. A stainless steel door of the same thickness is used to close the test location, where the test chamber is installed, before starting the radiation beam.

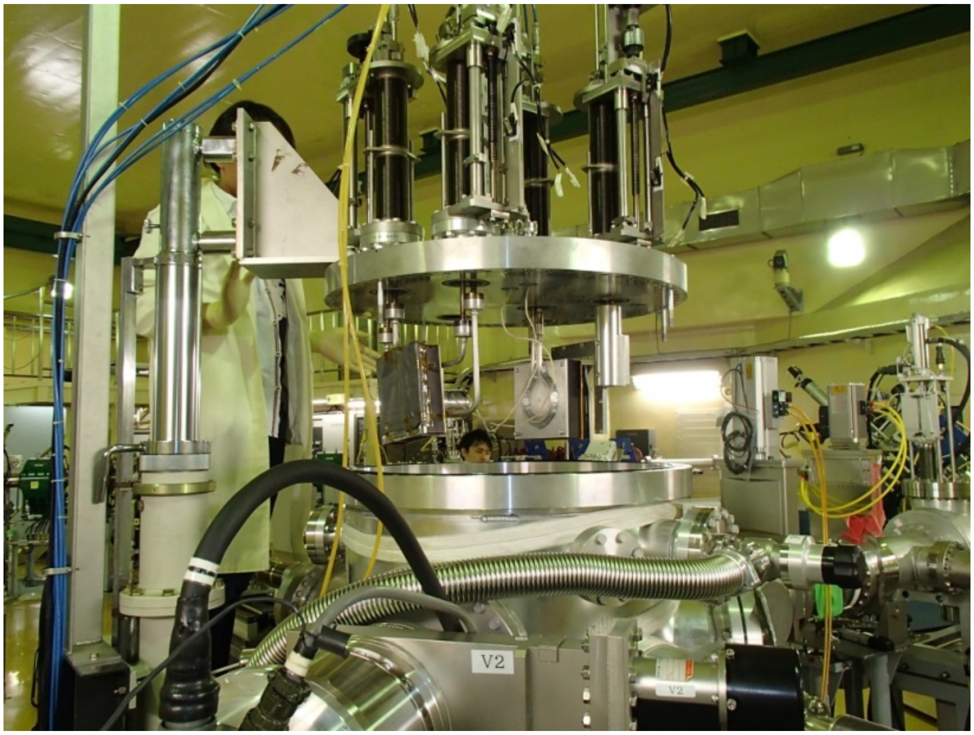


Figure 5-2. Radiation Test Chamber used in Accelerated Proton Beam Test.

Geiger counters are used to check the location before and after the test. The personal in charge of the test are checked at a stand-up facility to measure any radiation contamination in their bodies and/or clothes. Usually the level of radiation measured at the stand-up checking should be less than 0.001 mSv.

The beam line, shown in Figure 5-3, runs from the accelerator core to the vacuum test chamber where the DUT is installed. The proton particles are accelerated and kept in a straight beam using electromagnetic field.

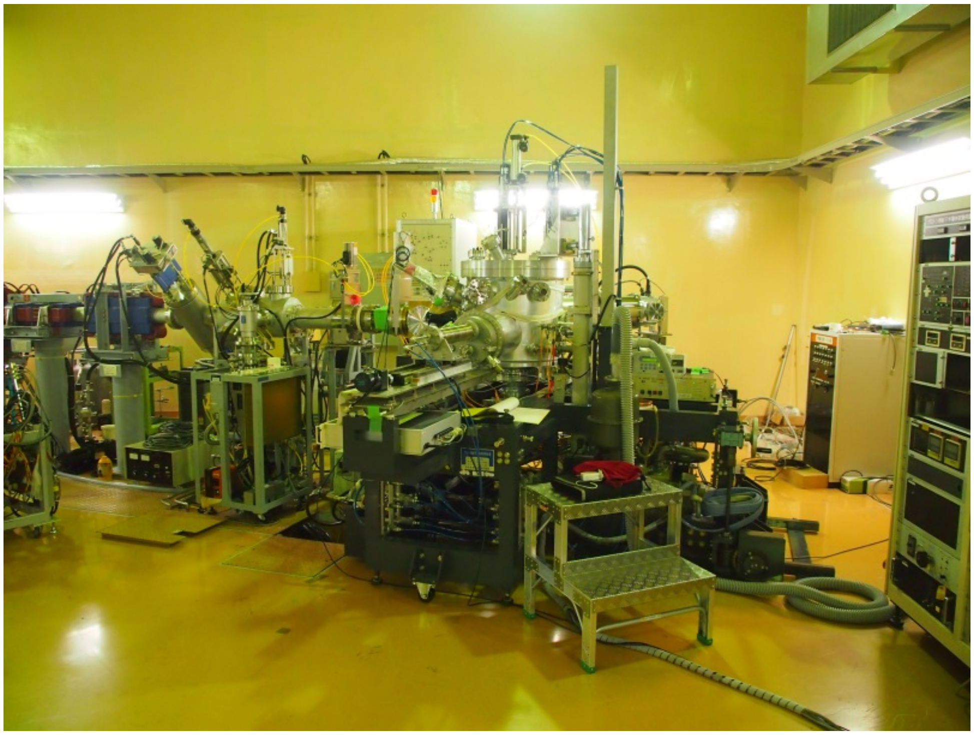


Figure 5-3. Proton Beam Line Connected to Test Chamber.

The beam is blocked from hitting the DUT using a shielding frame which can be removed remotely from the control room. It can be centered on the target through a setting mesh as shown in Figure 5-4.

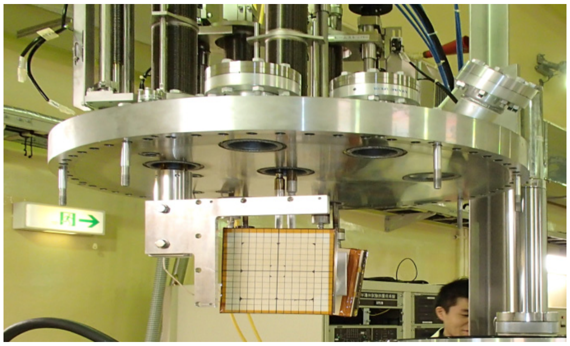


Figure 5-4. Beam Centering Grid.

A flange with two 25 pins male connections is used to carry the digital interface signals between the DUT inside the test chamber and the devices outside of it. The flange is used to guarantee that tight isolation is available in order not to have any air leakage from outside the chamber to its internal vessel while operating in vacuum. The flange is shown in Figure 5-5.

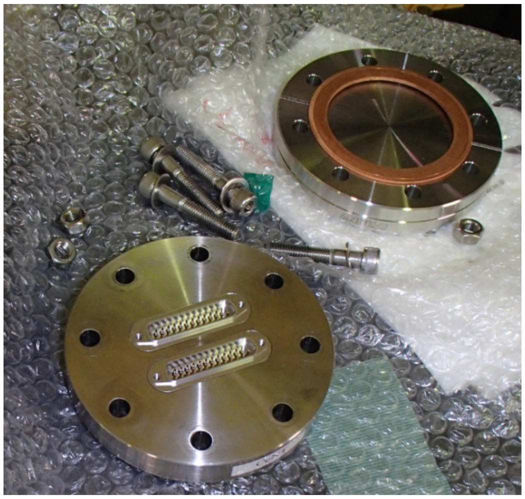


Figure 5-5. Interface Flange with Radiation Test Chamber.

The DUT is installed on a metal board facing the radiation beam. A copper plate of 1 cm thickness is installed above the DUT. The copper plate helps in shielding the components in the DUT that should not be subjected to radiation during the test, such as DC/DC converter and flash configuration memory. These components are: 1) either not used at all in the final design and they are just being used during the radiation test to provide supportive functions, 2) or they are used but they should be replaced with different types in the final design. An example of a supportive function, when there is an external microcontroller installed on the DUT to perform external scrubbing of the FPGA SRAM configuration memory. That microcontroller should not be the subject for radiation testing unless it would be used in the final design. In our test we focused on the FPGA to calculate its cross section using internal scrubbing.

The copper plate contains a hole to allow the radiation to pass through it and hit the surface of the FPGA in the DUT. Figure 5-6 shows the copper shield plate installed above the DUT and the hole just centered above the FPGA.

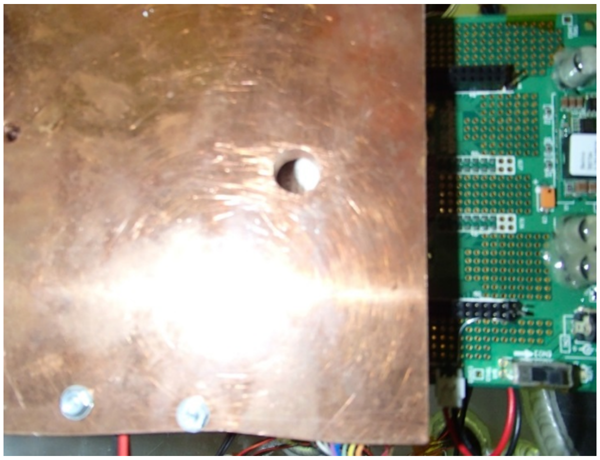


Figure 5-6. Copper Shield In Radiation Test.

The board is installed inside the test chamber with the minimum number of cables connecting it to the computers and power sources outside the test chamber through the flange interface. The cables are recommended to be as few as possible to avoid outgassing during the preparation stage when the test chamber is being vacuumed. If the cables are too many then it might not be possible to reach the required pressure level of 10^{-5} Pa which is mandatory to start the radiation test.

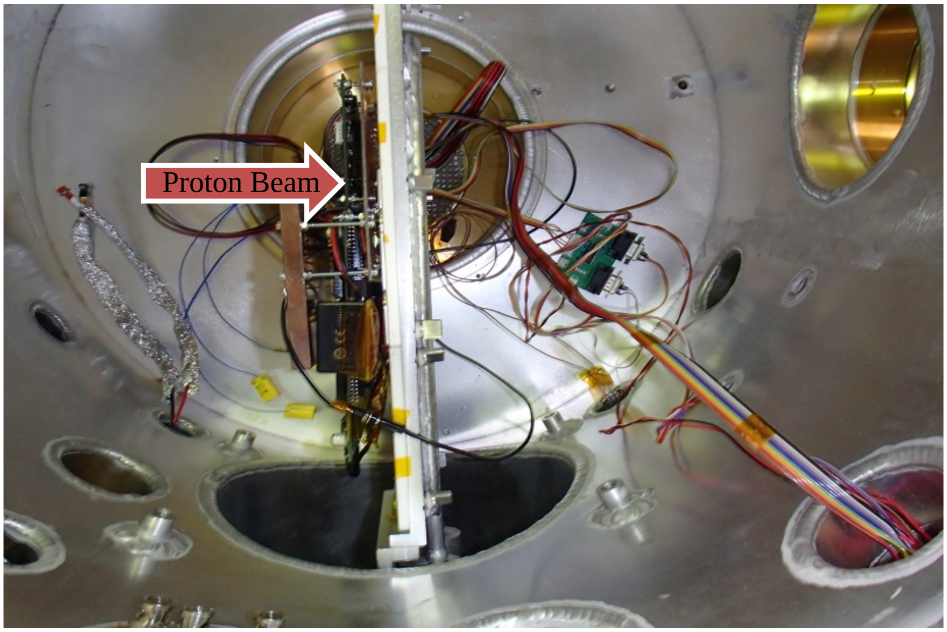


Figure 5-7. Top-view DUT in Radiation Test Chamber with RS-232 Interface.

Even if baking is performed on the cables and they were quite many then the outgassing is still possible to occur. We had a similar experience at the beginning of the radiation tests when we used RS-422 converters to extend the communication distance and avoid the electromagnetic noise. RS-422 is a differential serial communication technology which can reach up to 1000 meters with twisted pair cables at a speed of 10 Mbit/s. However, we could not reach the required pressure at the radiation test site as we were using bulk external RS-422 converters instead of compact ICs as shown in Figure 5-8. We had to redesign the communication board to be compact and of very small size and reduce the number of cables as shown in Figure 5-7.

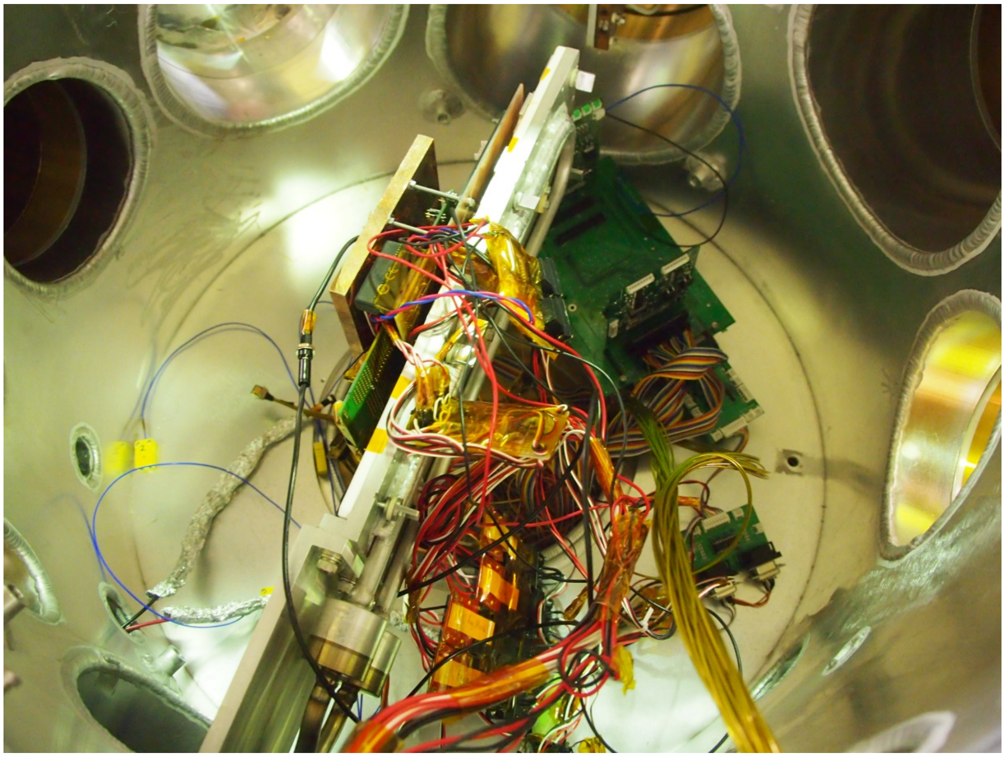


Figure 5-8. Outgassing due to External RS-422 Converters.

The overall test setup can be viewed in Figure 5-9. The proton beam runs across a 4 m concrete wall directly from the accelerator core. Coils are surrounding the beam tube to generate magnetic field in addition to the electric field to guide the beam and accelerate the particles. The DUT is installed on a handler and a copper plate is installed above it with a hole to pass the proton beam. Data is collected from the board through RS-232 interface which proved to work perfectly at low data rate of 9 Kbps. The interface cables as well as the power cables passed through the interface flange. A computer system with serial interfaces, Digital Input/Output (DIO), and

Data Acquisition (DAQ) cards managed the data logging and control. A Local Area Network (LAN) connection was provided between the room where the operators existed and the test site. During the test, no one was staying near the test chamber. All the monitoring and control actions were performed using remote desktop connection to the computer system at the test site.

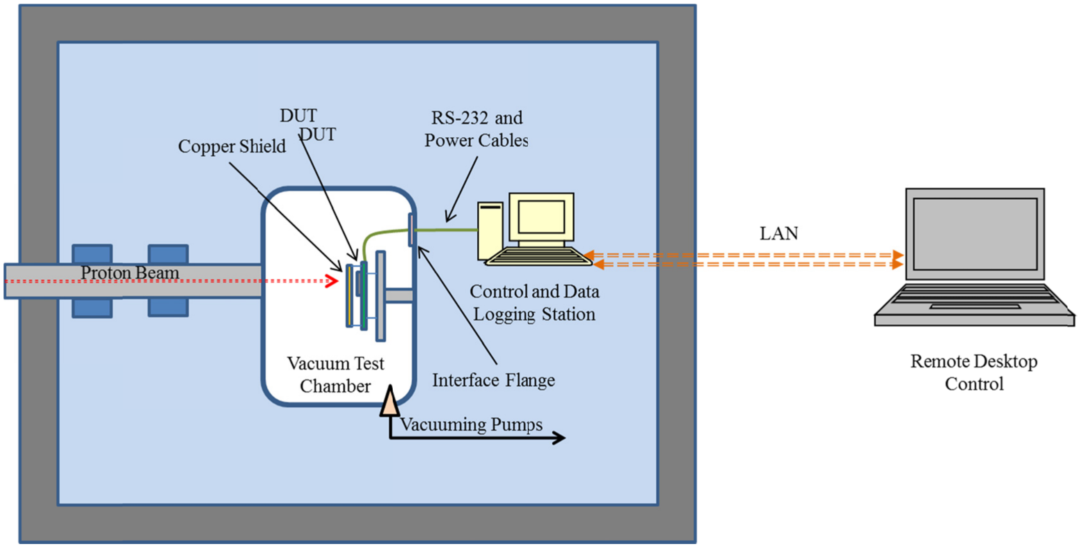


Figure 5-9. Accelerated Proton Beam Test Setup.

In summary the test setup can be arranged in the following steps:

1) Baking of the DUT for outgassing.
2) Installation at the test chamber.
3) Testing of interface, control, DUT and LAN.
4) Vacuuming of the test chamber.
5) Starting the radiation beam at specific flux and energy.

5.3 Radiation Test procedure

Before conducting accelerated proton beam test at TARRI, we thought to conduct the Cf-252 radiation test at KURRI as it is much easier than accelerated proton beam test in terms of the needed setup and necessary procedures. However, both tests would require a full health check as well as radiation test license to be obtained officially in order to have the permission to attend and participate in the tests. We were advised by the specialists at KURRI to de-capsulate the XC5VLX50-1FFG676C FPGA before the test to record the occurrence of SEUs. According to the nuclear physicists at KURRI, Cf-252 cannot penetrate through the package's material without de-capsulation. This is due to the fact that the stopping power ($-\frac{dE}{dx}$) (MeV/cm) of Cf-252, which is the retarding force acting against the charged particles to prevent it from traversing through the FPGA material, is high and therefore we might not record any SEU events. The XC5VLX50-1FFG676C FPGA layers are shown in Figure 5-1. The lid-heat spreader together with the thermal interface material should be removed out to reach the die.

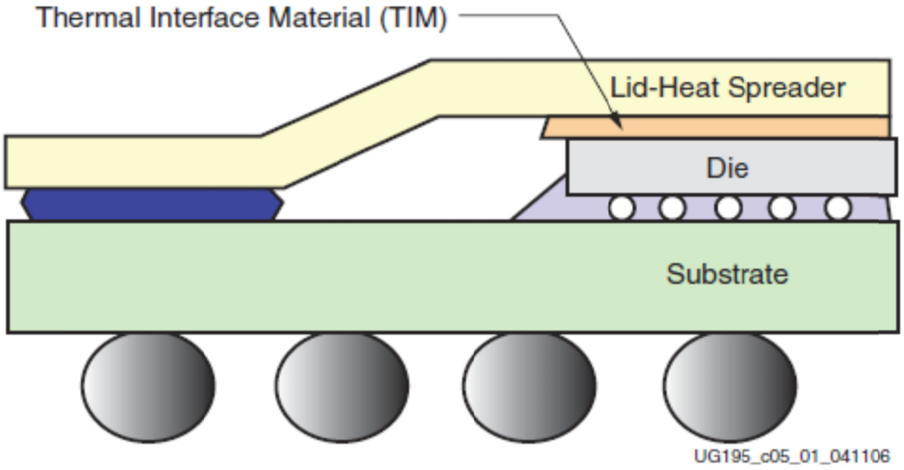


Figure 5-10. Virtex 5 Layers © Xilinx.

We performed de-capsulation for the XC5VLX50-1FFG676C FPGA at a specialized company in Japan. Before the de-capsulation process, an X-ray scan was conducted on the FPGA to get a clear understanding of the die connection to the substrate. The flip-chip method, where there are no wires bonding with the external pins, is used. Solder bumps locations can be seen in Figure 5-11. The space between the die and the board is filled with an electric insulator adhesion. This under-fill adhesion provides a mechanical support to fix the die well in its place as well as providing thermal compensation for the difference between the die and substrate Coefficient of Thermal Expansion (CTE).

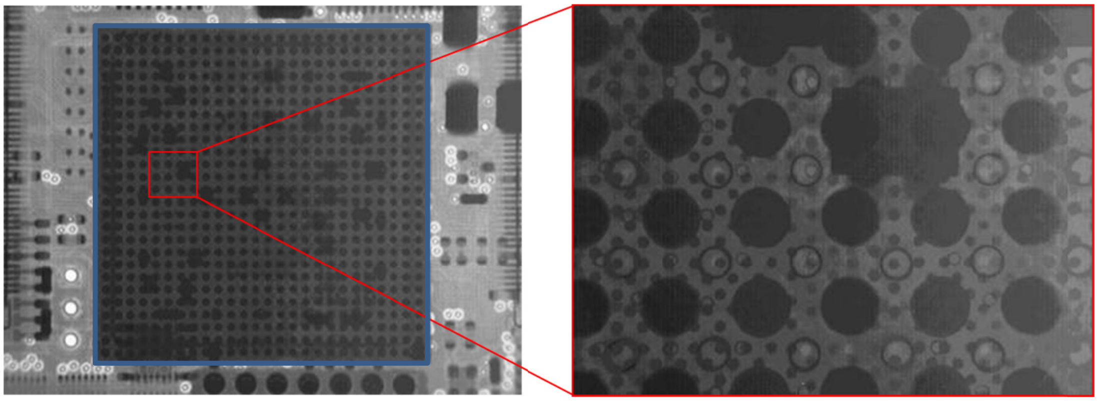
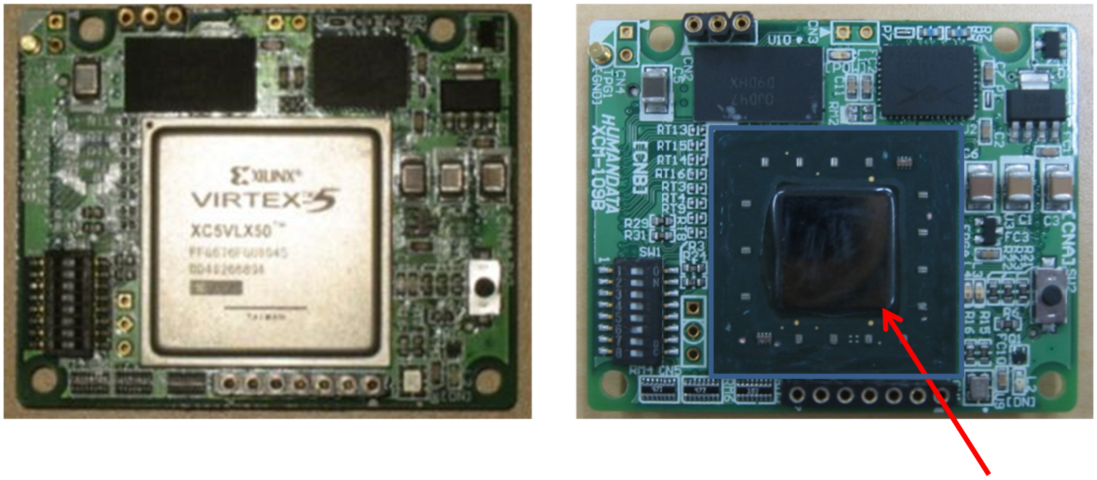


Figure 5-11. X-ray of the XC5VLX50-1FFG676C FPGA.

The FPGA was de-capsulated as shown in Figure 5-12. However, after de-capsulation the FPGA was not functional and the process was not successful. The company reported that the de-capsulation of Virtex FPGAs is difficult and would have a high failure rate if we chose to repeat the process. In order to be cost effective, we decided not to repeat the process of de-capsulation as it might lead to loosing high amount of FPGAs which is fairly expensive.



FPGA Silicon

Figure 5-12. FPGA before and after De-capsulation.

Instead of de-capsulation, we chose the accelerated proton beam testing as it was not required to de-capsulate the Virtex5 FPGA to be able to conduct the test.

Nevertheless, we did not stop the experiments for using Cf-252 in radiation testing. We had successful de-capsulation of the Spartan-6 FPGA which was used as the DUT to perform Cf-252 radiation testing at KURRI as shown in Appendix (C). The test showed the Cf-252 capabilities in inducing SEUs in the Spartan-6 die. We estimated the static and dynamic cross sections of the Spartan-6 based on the mean energy emitted by the Cf-252 which is estimated at 43 MeV.

Back to the Virtex5 FPGA, we chose the energy of the proton beam to be at the saturation level. In [64], it was shown that 65 MeV is a saturation energy for the XC5VLX50-1FFG676C FPGA. Although accelerated proton beam testing is difficult and needs a lot of complex setup, it does have an advantage when compared to Cf-252 testing. Proton beam has a consistent energy and flux while Cf-252 has a distribution of energies and fluxes. This means that when performing calculations of the cross section, proton beam would be easier, accurate and straight forward than Cf-252. In the following sections we illustrate the detailed procedure for the accelerated proton beam testing.

5.3.1 Safety Precautions

Radiation test at nuclear accelerators is an ultimately serious test. Although the test site is well prepared at both TARRI and KURRI, there are still some safety rules to follow. To be able to attend the test, a full health check should be made at an official governmental health check center to make sure that there are no health conditions that might lead to complex health problems during or after the test.

A basic training about radiation types, nuclear reactors operation, safety procedures at radiation sites and contingency behavior in case of radioisotope leakage and/or contamination. Upon finishing the training a license is issued for one year to conduct radiation testing at the JAEA based facilities or university based facilities. Both facilities are checked and inspected by JAEA and other official authorities in Japan. The license should be renewed annually to keep it valid. Renewal is done by attending refreshment training.

At the test site, radiation level should be checked for each participant before starting the test. The participants check the radiation level by standing on a device that measure the contamination, if any, by placing the hands in a Geiger scanner. While existing inside the nuclear reactor, all the participants wear a Geiger counter bracelet to continuously measure the radiation level in mSv. Participants wear a special coat and shoes while existing in the nuclear reactor for protection against radiation contamination.

The DUT is located at the radiation test location inside the test chamber. The participants are located at a different room. The participants can control the experiment and monitor the results in real time via remote desktop application over the LAN between the test site and their room. Before the experiment is started, the test commander locks the radiation site and activates the alarm that radiation test is in progress. The door is locked during radiation and no one is allowed to exist at that area.

Switching the DUT on/off should be considered carefully. As there is no possibility to interrupt the test by stopping the radiation beam and reopening the lock door, there should be a method to provide control and reconfiguration, if needed, remotely. In our test we designed an application using LabVIEW to control the switching of the DUT using relays and DIO cards. We controlled the LabVIEW application from our computers at the control room via remote desktop application over the LAN. In that way we kept controlling and monitoring the experiment without having to stop the beam and reentering the test site.

At the end of the radiation, the proton beam is stopped and then the radiation test alarm is deactivated. After opening the lock door, TARRI specialists start to scan the test site using handheld Geiger counters. All the devices used during the test are checked using the Geiger counters to make sure that they are not contaminated. The DUT itself is kept at TARRI for 3 months as it became contaminated. Radiation testing is very serious and should be handled very carefully and with great care as it might lead to catastrophic crises in case of carelessness or inaccuracy. Next we illustrate the accelerator parameters that should be controlled during the test, the software that was running on the MPSoC and how the SEU controller was used to calculate the upset rate.

5.3.2 Accelerator Parameters

The radiation test starts by defining the particles flux level, energy level and exposure time. The flux level (# of particles/cm^2/s), proton particles energy (MeV) and radiation exposure time (s) are the basic parameters used in calculating the radiation effects. Generally speaking, a flux level that generates sufficient statistically meaningful data in reasonable exposure time at the specified energy level is required. The radiation flux (# of particles/cm^2/s) is related to the radiation fluence (# of particles/cm^2) and radiation exposure time (s) as shown in Eq. **(5-1)** & Eq. **(5-2)** :

$$fluence = \frac{\#\, of\ Partricles}{Area}$$

(5-1)

$$Flux = \frac{fluence}{exposure\ time}$$

(5-2)

We recommend that the selection of the flux level be determined based on the target orbit where the system would operate. In our case we selected a polar orbit at a LEO altitude. The orbit was selected to match that of a previously launched satellite by KIT named HORYU-2. The orbit is also typical for earth observation missions. Table 5-1 shows the Horyu-2 satellite orbit parameters.

Table 5-1. Horyu-2 Satellite Orbit Parameters.

Orbit Type	general
Perigee Altitude	651 km
Apogee Altitude	671.6 km
Inclination	98.17°
RAAN	223.04°
Argument of Perigee	31.95°
True Anomaly	328.25°

Figure 5-13 shows the expected energy spectra versus the proton flux levels at the Horyu-2 orbit. The graph was produced by running simulation using the AP-8 model under SPENVIS.

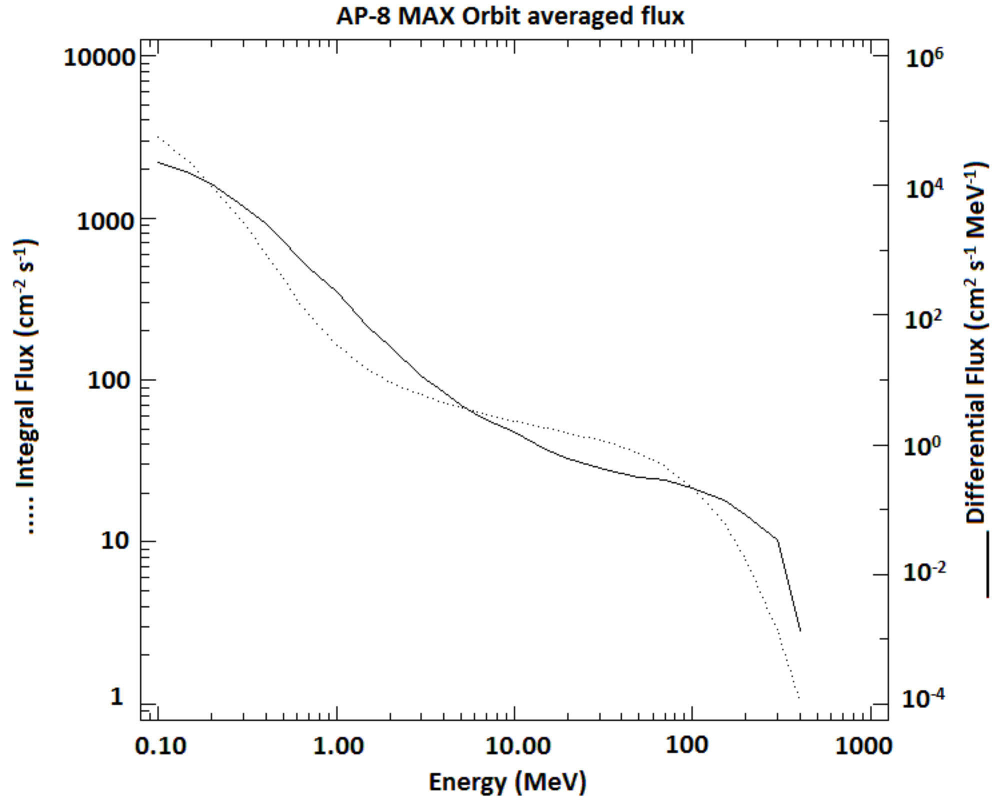


Figure 5-13. Proton Energy Spectra Versus Flux Level at Horyu-2 Orbit.

From the graph, we expect that a flux level of about 30 (protons/cm^2/s) is suitable to test at an energy of 65 MeV. However, at that flux level the exposure time might need to be very long to collect sufficient upset events. The exposure time to generate sufficient errors at that flux and energy level depends on the static-cross section of the DUT. The static cross section is defined in Eq. **(5-3)** :

$$\sigma_{static} = \frac{\#\ of\ upsets}{fluence} \tag{5-3}$$

If the test purpose was to determine the static cross section then several exposure times/flux levels should be tried. If the test purpose was to determine the dynamic cross section then the test flux level should be close to the in-orbit expected flux level at the same energy. The dynamic cross section is defined in Eq. **(5-4)** :

$$\sigma_{dynamic} = \frac{\#\ of\ system\ failures}{\#\ of\ upsets} \tag{5-4}$$

In our case, we wanted to check the static cross section, so we decided to have a very high flux to reduce the exposure time. In fact, beam time is very expensive and it should be paid for each minute of irradiation. Therefore a cost effective way would be to use a high flux level at the selected test energy level when performing static cross section measurement to reduce the exposure time of irradiation. In the case of dynamic cross section measurement, the test flux level should be comparable to the in-orbit flux level at the chosen particle energy. An increment test margin of two orders of magnitude would be fine to work with. However, very high flux levels, above two orders of magnitude, would be impractical to use as they would not represent the target environment. On the other side, taking into consideration that radiation occurs at different energies and fluxes in space as shown in Figure 5-13, a test flux level with an increment of two orders of magnitude above the expected in-orbit flux level would compensate for the absence of other particles at different energies. However, if the design would be subjected to very high unrealistic flux levels then the test would not represent the target environment and the capabilities of the used fault tolerance mitigation techniques would not be effectively evaluated.

We performed the radiation test at two flux levels, 3e06 and 10e06 ($cm^{-2}.s^{-1}$). Three times of testing were conducted at flux level of 3e06 ($cm^{-2}.s^{-1}$). One test was conducted at flux level of 10e06 ($cm^{-2}.s^{-1}$). The reason for test repetition was to get different data sets to avoid the effect of single time errors thus we could calculate a fairly accurate average value of the static cross section. We did not specify the exposure time for each test run. Instead of that, we were running the tests until the maximum number of FPGA configuration frame errors was reached. The embedded SEU controller was responsible for counting the number of frame errors. A total of 128 frame errors was the maximum record that the SEU controller could count. When 128 frame errors were reached we knew that we had to stop the test run and restart again as will be explained later. The energy level was set to 65 MeV during the tests and the corresponding exposure times are shown in Table 5-2:

Table 5-2. Accelerator Parameters Settings.

Flux Level ($cm^{-2}.s^{-1}$)	Energy (MeV)	Exposure Time (s)
3e06	65	25.16
		23.4
		21.7
10e06		82.4

5.3.3 Embedded Software

The MPSoC consists of four embedded processors as was explained earlier. Three processors implement TMR mitigation and the fourth processor acts as the master controller for handling system level tasks. During the radiation test the three processors run an Interrupt Service Routine (ISR) that enables them to print their status to a serial terminal via their UART ports and communicate an acknowledgement message to the master controller through the mailbox IP core.

The master control processor was responsible for receiving the messages from the slave processors to make sure that they are alive and working as well as resetting the external WDT. An SEU controller IP core was used to estimate the number of errors in the bitstream configuration frames stored in the internal FPGA SRAM. The embedded software of radiation and thermal vacuum testing are similar and will be described in details in chapter 6. Code Excerpt is presented in Appendix (D).

5.3.4 Upset Rate Estimation

The Xilinx SEU controller IP core was explained earlier in the MPSoC system design chapter. However, we would like to focus on how to use it in calculating the static cross section. The SEU controller operates in one of two modes, the Detection Only Mode (DOM) or the Auto Correction Mode (ACM). At the beginning of the test, the SEU controller initially starts in the ACM mode. Due to the very high flux level, there is a high probability that MBUs would occur in a single frame. When the first MBU occurs in a frame, the SEU controller switches to the DOM mode as it cannot correct MBUs. Data is logged through the SEU controller UART and displayed on the remote monitoring computers via remote desktop connection over the LAN. The SEU controller can count up to 128 frame errors. The upset rate can be estimated by calculating the time since the beginning of the DOM mode, which matches the start of an MBU occurrence, till reaching the 128 frames' errors.

Figure 5-14 is a sample of the output from the SEU controller IP core as a result of scrubbing the FPGA configuration frames during the radiation test at a flux level of 10e06 (cm^{-2}.s^{-1}).

#of complete scans	**Error Flag**	**#of frame errors**	**#of corr. errors**
000080	1	04	00
000100	1	08	00
000180	1	0B	00
000200	1	0C	00
000280	1	0E	00
000300	1	11	00
000380	1	13	00
000400	1	18	00
000480	1	1E	00
000500	1	20	00
000580	1	25	00
000600	1	28	00
000680	1	29	00
000700	1	2B	00
000780	1	33	00
000800	1	36	00
000880	1	3B	00
000900	1	40	00
000980	1	46	00
000A00	1	4B	00
000A80	1	52	00
000B00	1	58	00
000B80	1	61	00
000C00	1	67	00
000C80	1	6C	00
000D00	1	6E	00
000D80	1	75	00
000E00	1	7C	00
000E80	1	80	00

Figure 5-14. Sample Output from the SEU Controller During Radiation Test.

The static cross section can be calculated in terms of the event bit cross section or the upset bit cross section. The event bit cross section means that we count the number of events where we had upsets regardless of how many bits were actually upset during the event. For example if two bits had an upset at the same time, we would consider this as a single event upset. The upset bit cross section means that we count the number of bits that were upset either simultaneously or at different times. The two bits in the previous example would denote an upset bit of two. The calculation of the static event cross section is shown in Eq. **(5-5)**, the static bit cross section is shown in Eq. **(5-6)**, and the average static cross section is shown in Eq. **(5-7)**:

$$Static\ Event\ Cross\ Section\ per\ bit = \frac{\#of\ event\ errors \Big/ (flux\ \times exposure\ time)}{device\ size\ in\ bits}$$

(5-5)

$$Static\ Bit\ Cross\ Section\ per\ bit = \frac{\#of\ bit\ errors \Big/ (flux\ \times exposure\ time)}{device\ size\ in\ bits}$$

(5-6)

$$Average\ Static\ Cross\ Section\ per\ bit = \frac{Static\ Event\ Cross\ Section\ per\ bit + Static\ bit\ Cross\ Section\ per\ bit}{2}$$

(5-7)

The results of the radaition test to estimate the static event cross section are detailed in chapter 7. So far we investigated the use of radiation test with high flux, low exposure time, to estimate the static cross section. In the following sections we investigate the use of fault injection to estimate the dynamic cross section as an alternative for low flux, high exposure time and hence high cost, radiation testing.

5.4 Fault Injection Test Objectives

Fault Injection test is used to estimate the effects of SEUs on the designs within the lab. In a well-designed fault injection test, dynamic cross section could be estimated accurately. Fault injection test is an inexpensive alternative for radiation test in estimating the dynamic cross section of a design.

The following are the main objectives of Fault Injection test:

1) Performance evaluation of designs by emulating fault events through injecting faults at random positions.

2) Estimation of design dynamic cross section as an alternative for expensive and complicated radiation tests.
3) Evaluation of the effectiveness of the fault tolerance mitigation techniques used to protect the design against SEUs.
4) Provide a fast, inexpensive and easy to use tool to test the designs within the labs.
5) The test can be run at different upset rates to simulate different orbits; the upset rates can be used as an input for estimating the adequate flux level and energy if the radiation test is already planned.

5.5 Fault Injection Concept

Fault injection in functioning systems is a technique used to insert deliberate faults at selected and/or random units of the design to assess its sensitivities. This technique is implemented by adding additional hardware and software to the system to handle the insertion of faults, monitoring of performance and collection of results. Figure 5-15, shows the architecture of a fault injection system. The faults are injected to the DUT and statistical results are issued as a feedback to the injecting machine for test vectors adjustment.

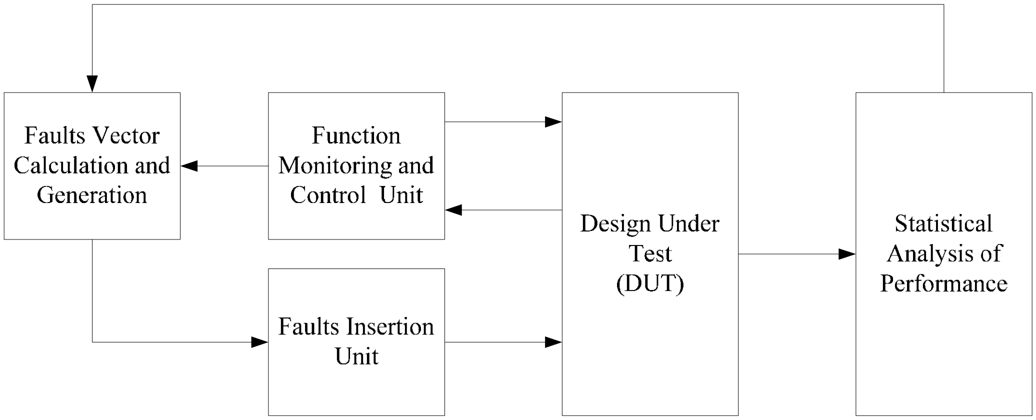

Faults Vector Calculation and Generation
Function Monitoring and Control Unit
Design Under Test (DUT)
Statistical Analysis of Performance
Faults Insertion Unit

Figure 5-15. Fault injection cycle.

The DUT is interfaced to a faults insertion unit which has access to the design units where faults are to be injected. The faults vector calculation and generation unit prepares faults vectors that match the required test objectives. The fault insertion unit could be a combination of hardware and software. It handles the overriding of the normal operation into a faulty one. For example, the fault insertion unit could be a code that reads back a previously calculated value by the normal DUT code and then overwrites it with a faulty value to simulate a specific condition. The insertion could be done without stopping the main operation. In some designs it might be inevitable to interrupt the normal operation flow by suspending it and then resuming after the injection takes place. The function monitoring and control unit takes care of monitoring the operation of the DUT. It stops the DUT operation in case of noticing an emergency and provides a control path to set the DUT in specific operating modes and operation settings. The performance of the DUT is statistically analyzed to detect anomalies in normal operation as faults are injected. The feedback about how the DUT behaves while in fault injection mode is provided to the faults vector calculation and generation unit. It uses that information in generating new fault vectors. For example, the feedback statistical information might show that there is a repetitive pattern in the output when certain fault sequence is followed. The faults vector generation and calculation unit might repeat the vectors with different variations to study the statistical dependence between injected faults and output vectors. Fault monitoring and control unit also feedback the faults vector calculation and generation with information about the behavior of the DUT during the fault injection process. For example, it might be necessary to feedback the faults vector calculation and generation unit with the moments where the system completely stopped working and needed a deep reset. This information is used in detecting the types of faults that lead to total failure.

The SEUs which occur in space are probabilistic. Poisson distribution is used to estimate the expected number of upsets (k) which happens in the time interval (T) with an average number of upsets (μ) according to the probability density function shown in Eq. **(5-8)**. The exponential distribution is used to estimate the expected time between upsets (τ) with an average number of upsets in unit time interval (λ) as shown in Eq. **(5-9)**. The relationship between both distributions can be set as (μ = λT).

$$P(x|\mu) = (\mu^x/x!)\, e^{-\lambda}$$

(5-8)

$$P(\tau|\lambda) = \lambda e^{-\lambda\tau}$$

(5-9)

5.6 SEU Fault Injector

The fault Injector system architecture is shown in Figure 5-16. The injector uses an internal hardware unit that can reconfigure the FPGA bit stream, the SEU controller. It is an IP core that is provided by Xilinx which can be controlled from outside the FPGA to produce faults in the form of bit flipping in the FPGA configuration frame. The control of the SEU controller is through serial communication over the RS232 channel to send commands to it and receive responses from it. The fault injector system contains three external computers to support its function. The fault injector computer which runs MATLAB script to generate random faults lists based on the Poisson distribution of the SEUs in the target orbit. It generates the timing at which faults will be injected which follows the exponential distribution as described earlier. Another computer is used for configuring the FPGA with the bit-stream which contains the hardware design.

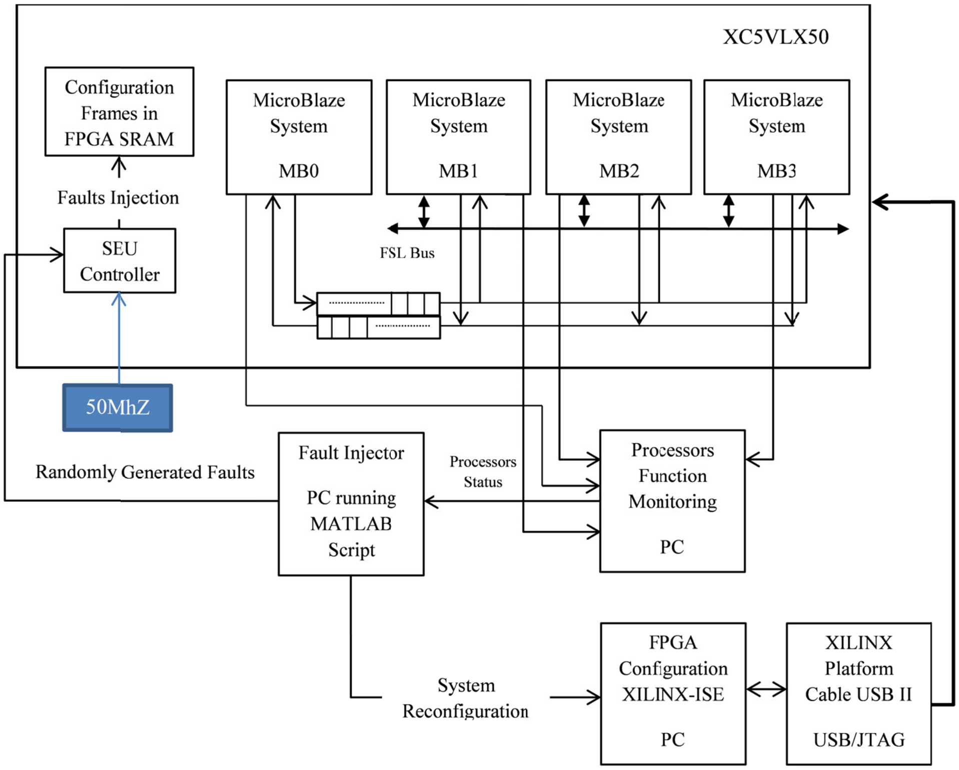

XC5VLX50
Configuration Frames in FPGA SRAM
Faults Injection
SEU Controller
50MhZ
MicroBlaze System MB0
MicroBlaze System MB1
MicroBlaze System MB2
MicroBlaze System MB3
FSL Bus
Randomly Generated Faults
Fault Injector PC running MATLAB Script
Processors Status
Processors Function Monitoring PC
System Reconfiguration
FPGA Configuration XILINX-ISE PC
XILINX Platform Cable USB II USB/JTAG

Figure 5-16. Fault Injector System Architecture.

The function monitoring of the DUT, the MPSoC, is done through sending the processors status and results of executing a simple counter program to the UART interfaces which are monitored by an external computer to collect the results and analyze them. The system runs the simulation for number of times and it generates a new fault injection vector at each time. The fault injection vector contains the bit location that will be flipped which is a random number from (0 to 1311) and the frame number where flipping will take place which is a random number from (1 to 8662), these numbers are device specific to the Virtex5 LX50. Appendix (E) shows a script used for the generation of the frame addresses in the format specific to the Virtex-5 through invoking the SEU controller. Appendix (F) shows a sample of the single bit upsets injection script.

The faults are accumulated and their effects are watched as they are injected. At the end of the injection cycle, ACM is enabled to recover the injected faults and restore the operation of the cores. The flow chart in Figure 5-17, shows the test flow.

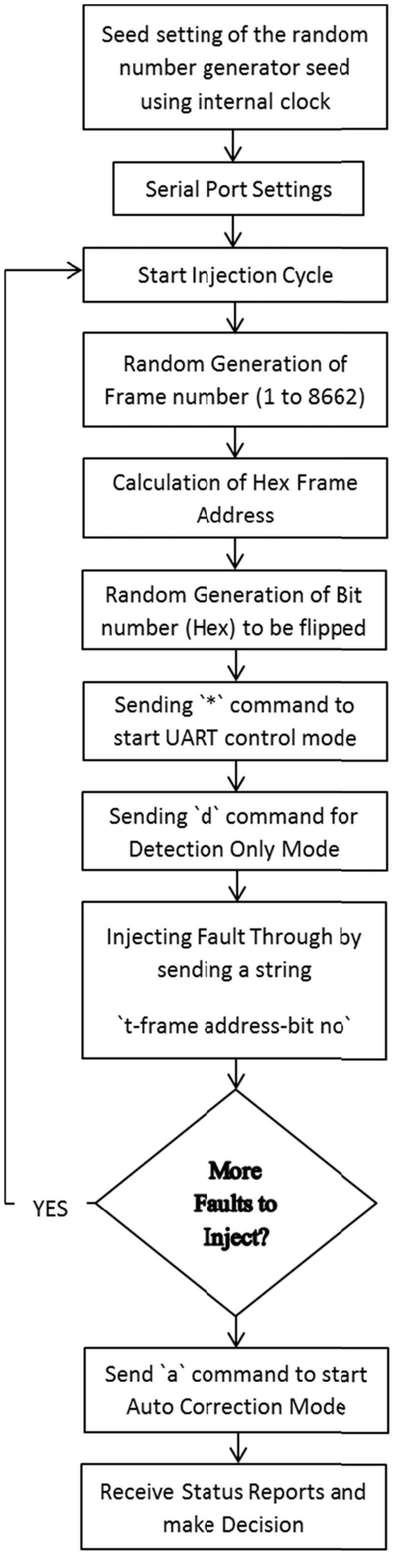

Seed setting of the random number generator seed using internal clock
Serial Port Settings
Start Injection Cycle
Random Generation of Frame number (1 to 8662)
Calculation of Hex Frame Address
Random Generation of Bit number (Hex) to be flipped
Sending `*` command to start UART control mode
Sending `d` command for Detection Only Mode
Injecting Fault Through by sending a string
`t-frame address-bit no`
More Faults to Inject?
YES
Send `a` command to start Auto Correction Mode
Receive Status Reports and make Decision

Figure 5-17. Fault Injection Test Flow Chart.

DOM is used during the accumulated fault injection process. The SEU controller only monitors and reports faults when operating in the DOM without performing any correction of the upsets. The faults are injected using specific commands over serial interface with the SEU controller. The result of fault injection is observed through monitoring the MPSoC operation over the serial interfaces.

The function of the TMR fault mitigation approach is tested through injecting faults in the data carried by the redundant modules. Each module carries the same set of data which might represent operation state code, software variables, communication message or any other form of application specific information. The TMR concept depends on voting among the data as shown in Figure 5-18.

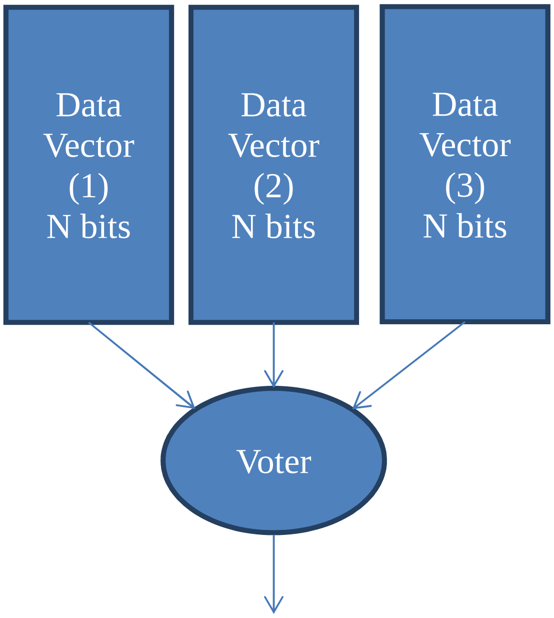


Figure 5-18. TMR among N bits Vectors.

The TMR approach is used in protecting the data by comparing among the data values of three functionally identical modules. In case differences among the data values exist then consensus among the values would be used to propose the most accurate value. If no consensus is found then an error is signaled. In simulating faults in the TMR data vectors of (N bits), a random fault list is generated to indicate the locations of the bits to be flipped. The data sets are then compared to each other and errors are calculated based on the comparison results as shown in the script excerpt in Figure 5-19. The results of fault injection are detailed in chapter 7.

```
% Fault Simulation
for i = 1:Total_Upsets
    Byte = ceil(random_bits(i,1)/8);
    Bit = rem(random_bits(i,1),8);
    Buffer_1(Byte) = bitxor(Buffer_1(Byte),2^(Bit));
    Byte = ceil(random_bits(i,2)/8);
    Bit = rem(random_bits(i,2),8);
    Buffer_2(Byte) = bitxor(Buffer_2(Byte),2^(Bit));
    Byte = ceil(random_bits(i,3)/8);
    Bit = rem(random_bits(i,3),8);
    Buffer_3(Byte) = bitxor(Buffer_3(Byte),2^(Bit));
end
%calculation of TMR system failure rates
for i = 1 : Buffer_Size(BS_i)
if (Buffer_1(i)~= Buffer_2(i)) && (Buffer_1(i)~= Buffer_3(i)) && (Buffer_2(i)~= Buffer_3(i))
Unsim_Failures(j) = Unsim_Failures(j) + 1;
end
end
```

Figure 5-19. Part of the MATLAB Script to Simulate TMR Errors.

In this chapter we presented the radiation test using accelerated proton beam at TARRI. We estimated the static cross section using a new technique through internal scrubbing with the SEU controller core. Traditional techniques, which use a monitoring and control device to scrub the FPGA externally, give accurate results on the expense of complicated test setups. Our technique, gives accurate results with the advantage of reducing the number of connections and components inside the test chamber which is already of limited volume. Reducing the number of connections and components helps in quickly reaching the 10^{-5} Pa vacuum pressure which is necessary to start the radiation test. Also, we presented the fault injection test method to estimate the dynamic cross section. This method is a cost effective alternative to expensive radiation tests which might take long times to estimate the dynamic cross section depending on the static cross section, flux level, energy level and exposure time. Both the dynamic and static tests showed that the SEU controller is robust as it showed no halts during the tests and it effectively scrubbed mitigated the FPGA against induced upsets. In the next chapter we introduce the thermal vacuum tests and the mitigation techniques to reduce the thermal power through reducing the static and dynamic power consumption of the FPGA.

Chapter 6 : Thermal Vacuum Testing

Satellites' subsystems suffer from the harsh conditions in the space environment. As was illustrated in the previous chapter, radiation environment has severe effects on the binary design data in SRAM-based FPGAs. In this chapter we handle the effects of the thermal environment on the operating temperature and power consumption. We performed thermal vacuum testing to reveal whether the FPGA package would fit and sustain its operation during the mission lifetime or not. In the following sections we present the test setup and procedures as well as the applied thermal mitigation techniques. We used two approaches to thermally mitigate the FPGA design: 1) Active control technique through Reconfiguration and Clock Frequency Control (CFC) and 2) Passive control technique through heat sink. DTM technique was used to achieve the active thermal control. Some sections of this chapter were part of the author publications at [67].

6.1 Space Thermal Environment

The heat sources in LEO are the direct solar radiation, solar reflection (Albedo), and Earth Infra-Red (IR) emission. Figure 6-1 shows the schematic of the external heat sources in LEO space.

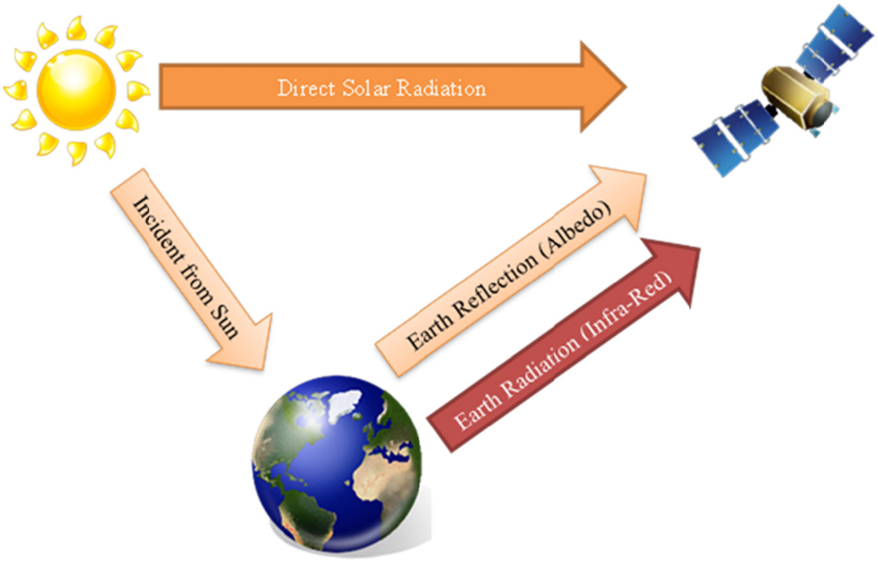


Figure 6-1. Space Thermal Environment.

Satellites can have very high temperature at one side, which directly faces the sun, while very low temperature at the other side, which faces the deep space. The amount of temperature difference between surfaces is inversely proportional to the satellite specific heat capacity $\boldsymbol{c_p}$ (J/kg.K) and mass **m** (kg). The amount of heat transfer per unit time $\boldsymbol{Q}$ (W) used to change the temperature by $\mathbf{\Delta T}$ (K) in time $\mathbf{\Delta t}$ (s) is calculated by Eq. (6-1) :

$$Q = mc_p \frac{\Delta T}{\Delta t} \tag{6-1}$$

Where: $\boldsymbol{Q}$ is the heat transfer per unit time (W), $\boldsymbol{m}$ is the satellite mass (kg), $\boldsymbol{c_p}$ is the specific heat capacity (J/kg.K), $\mathbf{\Delta T}$ is the temperature difference (K) and $\mathbf{\Delta t}$ is the duration (s).

The direct solar radiation flux is about 1371 W/m^2 at 1 AU from the sun, the average reflected solar radiation (albedo) is 0.3 of the direct solar radiation, and the planetary IR flux for planet Earth is 234 W/m^2 as shown in **[68]**, which is adapted from **[69] [70] [71]** .

Table 6-1. LEO Thermal Environment [68].

	Aphelion	**Perihelion**	**Sun Light**	**Eclipse**
Direct Solar	1414 W/m^2	1323 W/m^2	1371 W/m^2	0
Albedo (average)	0.30 ±0.01	0.30 ±0.01	0.30 ±0.01	0.25
Planetary IR (average)	234 ±7 W/m^2	234 ±7 W/m^2	234 ±7 W/m^2	220 W/m^2

The thermal balance equation is represented in Eq. (6-2):

$$Q_{External} + Q_{Internal} = Q_{Radiated} \tag{6-2}$$

Where: $\mathrm{Q_{External}}$ is the external heat transferred to the satellite from the thermal space environment (W), $\mathrm{Q_{Internal}}$ is the internally generated heat due to the power consumption of the subsystems' electronic components (W), and $\mathrm{Q_{Radiated}}$ is the emitted heat from the satellite to space in the form of electromagnetic radiations (W).

Eq. (6-3) expands the external heat transfer term:

$$Q_{Sun} + Q_{Albedo} + Q_{Earth} + Q_{Internal} = Q_{Radiated}$$

(6-3)

Where: Q_{Sun} is the heat transferred due to the direct solar radiation (W), Q_{Albedo} is the heat transferred due to the reflected solar radiation from the Earth surface (W), and Q_{Earth} is the planetary infrared of planet Earth (W).

Generally, heat is transferred in three ways: 1) Conduction, 2) Convection, and 3) Radiation. Conduction heat transfer is due to the flow of heat inside the same body or the contact between two surfaces. Convection heat transfer is due to the flow of a fluid on a surface. Radiation heat transfer is due to the propagation of heat in the form of electromagnetic waves such as IR waves.

6.1.1 Conduction Heat Transfer

Heat flow through conduction is due to the interactions that take place inside the materials. The conduction heat transfer is described by the Fourier's Law as in Eq. (6-4):

$$Q_x = -K\frac{dT}{dx}$$

(6-4)

Where: Q_x is the local heat flux density, it represents the amount of energy that flows per unit area in unit time (W/m^2), K is the thermal conductivity of the material (W/m.K), $\frac{dT}{dx}$ is the temperature difference over the length of the material (K/m).

An analogy between current flow through a resistance in electrical circuits and heat flow via conduction can be represented through the thermal resistance in Eq. (6-5).

$$R_{th} = \frac{L}{KA}$$

(6-5)

Where: R_{th} is the thermal resistance (K/W), L is the length of the material (m), K is the thermal conductivity (W/m.K), and A is the cross sectional area through which the heat flows (m^2).

The conduction heat transfer between two arbitrary nodes (i) and (j) can be rewritten in terms of the thermal resistance as in Eq. (6-6):

$$Q_{ij} = \frac{T_i - T_j}{R_{ij}}$$

(6-6)

Where: Q_{ij} is the heat flow between nodes (i) and (j) (W), T_i is the temperature of node (i) (K), T_j is the temperature of node (j) (K), and R_{ij} is the thermal resistance between nodes (i) and (j) (K/W).

Conduction heat transfer takes place between the different materials when they have contact surfaces. The contact conductance C_{cond} (W/K), is used to characterize the amount of heat transfer per temperature unit, it is the inverse of the thermal resistance R_{th} in Eq. (6-5).

The contact conductance depends on the type of materials in contact, the surfaces' finishing, pressure exerted to keep the surfaces in contact and the pressure uniformity [72] [73].

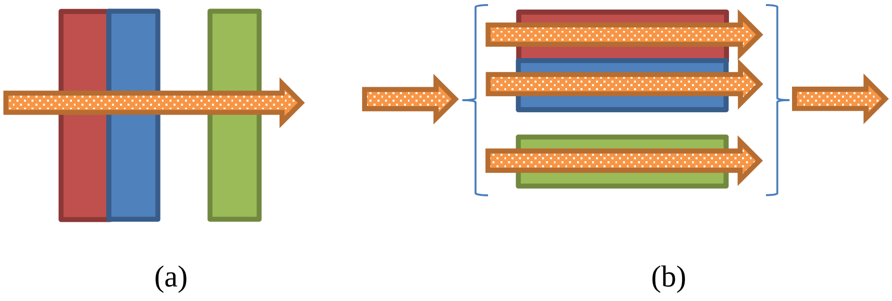


Figure 6-2. a) Heat Flow in Series and b) Heat Flow in Parallel.

If different materials touch each other as shown in Figure 6-2, such that heat flows either in series as in (a) or in parallel as in (b), then the overall equivalent thermal resistance can be calculated from Eq. (6-7) and Eq. (6-8):

$$R_{series} = \sum_{i=1}^{n} R_i$$

(6-7)

$$R_{parallel} = \sum_{i=1}^{n} \frac{1}{R_i}$$

(6-8)

Where: R_i is the thermal resistance of material (i).

6.1.2 Convection Heat Transfer

Space is a vacuumed environment, where air density from the LEO altitudes and above is negligible. Therefore, heat is mainly transferred in space through radiation and conduction. However, in some thermal control techniques, such as heat pipes, a fluid is used to carry the heat from hot regions to cold regions in a thermal cycle of evaporation and condensation. Usually Fluids that can evaporate at low temperatures such as Ammonia are used in such methods [72] [73] [13]. Convection heat transfer is used in the design of heat pipes in conjunction with conduction heat transfer. It is used in calculating heat transfer in fluid fuel tanks or pressurized biological experiments on a spacecraft [72]. The convection heat transfer between two arbitrary nodes (i) and (j) can be rewritten in terms of the thermal resistance as in Eq. (6-6). The thermal resistance in convection heat transfer is shown in Eq. (6-9):

$$R_{th} = \frac{1}{hA}$$

(6-9)

Where: h is the convection heat transfer coefficient (W/m^2.K), and A is the cross sectional area through which heat flows (m^2).

6.1.3 Radiation Heat Transfer

The radiation heat transfer is significant when the temperature difference between the radiating surfaces is high. On the other side, low temperature differences between the radiating surfaces causes the value of the radiation heat transfer to be very low such that it can be totally neglected. In case the satellite electronics have close power consumption values, then the mutual radiation heat transfer among them would be very small and likely to be neglected.

The radiation heat transfer for an ideal black body with emissivity ($\varepsilon = 1$) is described by Stefan-Boltzmann's law as shown in Eq. (6-10):

$$\mathrm{E} = \sigma \mathrm{T}^4$$

(6-10)

Where: E is the radiated heat flux (W/m^2), σ is the Stefan-Boltzmann constant (5.67x10^{-8} W/m^{2}K^4), and T is the radiating surface temperature (K). The amount of heat transfer between two radiating surfaces can be described by Eq. (6-11) [70]:

$$q_r = \varepsilon\sigma F_{1,2} A(T_1^4 - T_2^4)$$

(6-11)

Where: q_r is the amount of heat transfer between two radiating surfaces (W), ε is the emissivity of the radiating surface material which ranges from (0) for reflective surface to (1) for absorptive surface, $F_{1,2}$ is the view factor between surface (1) and surface (2) (≤1.0).

It is important to note that the summation of the solar radiation amount that is absorbed, transmitted and reflected over a surface equals to 1 as shown in Eq. (6-12):

$$\alpha + \rho + \tau = 1$$

(6-12)

Where: α is the absorbitivity, ρ is the reflectivity, and τ is the transmissivity [74]. When the surface is in thermal equilibrium then Kirchhoff's law implies that absorptivity equals to emissivity as shown in Eq. (6-13):

$$\alpha = \epsilon$$

(6-13)

6.1.4 Radiation View Factors

Radiation heat transfer depends on the orientation of the surfaces with respect to each other. The view factor F_{ij} between surfaces (i) and (j), also known as the shape factor, is used to define the fraction of radiation emitted by surface (i) and intercepted by surface (j) [75]. The fraction of radiation emitted by surface (j) and intercepted by surface (i) is the view factor F_{ji} between surfaces (j) and (i). The relationship between F_{ij} and F_{ji} is shown in Eq. (6-14):

$$A_i F_{ij} = A_j F_{ji}$$

(6-14)

Where: A_i and A_j are the surface areas (m^2).

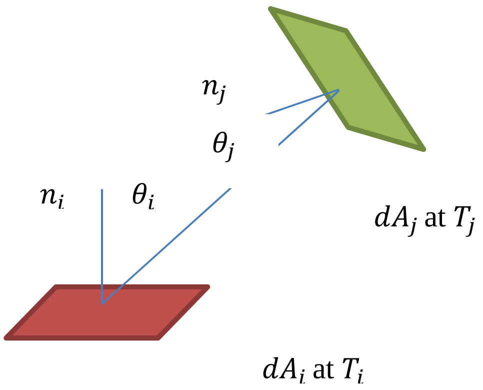


Figure 6-3. View Factor Calculation Geometry **[72]**.

The calculation of the view factors can be estimated using Figure 6-3. Two differential areas dA_i and dA_i are shown at surfaces T_i and T_j. The normal to the surfaces are n_i and n_j. The angles θ_i and θ_j are located between the normal to the surfaces and the shortest vector (S) connecting between both surfaces.

The view factor between two differential areas is calculated in [76] and is shown in Eq. (6-15):

$$F_{di-dj} = \frac{\cos\theta_i \cos\theta_j}{\pi S^2} dA_i dA_j$$

(6-15)

The exact solution of the view factor can be found by integrating over the areas as shown in Eq. (6-16):

$$F_{ij} = \frac{1}{A_i} \int \int \frac{\cos\theta_i \cos\theta_j}{\pi S^2} dA_i dA_j$$

(6-16)

The view factors of common geometric configurations used in space crafts are previously calculated [70] [76]. Computer Aided Design (CAD) tools can perform numerical calculation of view factors for arbitrary geometric configurations.

6.1.5 Equilibrium Temperature

When we perform thermal analysis of a system then we divide the system into nodes where we calculate the thermal conditions at each of them. The thermal equilibrium or thermal balance is achieved at any node when the total amount of heat flow input to the node equal to the total amount of heat flow output from it as shown in Eq. (6-17):

$$\sum Q_{in-i} = \sum Q_{out-i}$$

(6-17)

Where: $\mathrm{Q_{in-i}}$ is the heat flow input to node i (W) while $\mathrm{Q_{out-i}}$ is the heat flow output from node i (W).

It should be noted that the higher the number of the nodes at which we perform the analysis, the more accurate the analysis results would be.

To illustrate the concept, we construct a mesh of nodes as shown in Figure 6-4 :

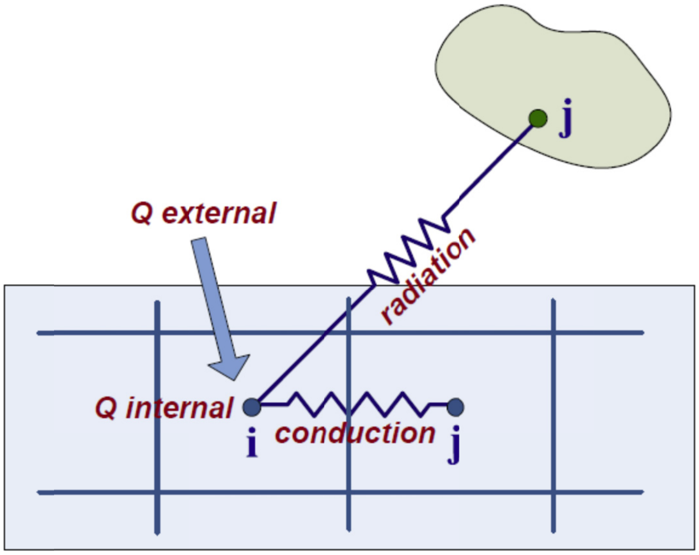


Figure 6-4. Thermal Equilibrium Analysis through Mesh Nodes **[68]**.

The thermal equilibrium equation at node i, assuming that analysis is performed in reference to node j and node z which have higher temperatures than node i, can be written as in Eq. (6-18):

$$mc_p \frac{\Delta T_i}{\Delta t} = \frac{T_j - T_i}{R_{th-ji}} + \varepsilon\sigma F_{zi} A_z (T_z^4 - T_i^4) + Q_{ext} + Q_{int}$$

(6-18)

Where: m is the mass (kg) at node i, $\mathrm{c_p}$is the specific heat capacity (J/kg.K) at node i, $\Delta\mathrm{T_i}$ temperature change at node i (K), $\Delta\mathrm{t}$ is the duration (s), $\frac{\mathrm{T_j - T_i}}{\mathrm{R_{th-ji}}}$ is the conduction heat transfer term from node j to node i (W), $\varepsilon\sigma\mathrm{F_{zi}A_z(T_z^4 - T_i^4)}$ is the radiation heat transfer term from node z to node i, $\mathrm{Q}_{\mathrm{ext}}$ other external heat input to node i (W), $\mathrm{Q}_{\mathrm{int}}$ internal heat dissipation input to node i (W).

The importance of understanding the heat balance or thermal equilibrium of a system is that it helps in constructing accurate Thermal Mathematical Model (TMM) for the system. The thermal balance test is used in measuring the heat balance conditions; it can be combined with the thermal vacuum test, in one step, if the thermal control subsystem was not the subject of the test. TMM also defines the maximum and minimum limits for conducting the thermal vacuum tests. Thermal balance tests should be conducted in vacuum condition where the pressure is preferred to be less than 10^{-4} Pa.

6.2 Thermal Effects on 65 nm Virtex-5 FPGA

The effect of thermal conditions on electronic components operation is mainly characterized by changing the power consumption. In Xilinx FPGAs there are 6 power modes: 1) Power on, 2) Configuration, 3) Standby, 4) Active, 5) Suspend and 6) Hibernate [77]. The active power consumption is the power consumed during the design operation and it is equal to the summation of the standby power and the design dynamic power. The standby power is the summation of the device static and design static power consumptions. The active power is estimated from Eq. (6-19) and Eq. (6-20) [77]:

$$Standby\ power = Device\ Staic\ power + Design\ Static\ power \tag{6-19}$$

$$Active\ Power = Standby\ power + Design\ Dynamic\ power \tag{6-20}$$

The device static power depends on the transistor feature size, applied voltage, junction temperature and the manufacturing process [77]. The junction temperature itself depends on the ambient temperature, applied voltage and total current supplied. However, the total current supplied depends on the device static power. It is a circular dependency and tools used to estimate the power consumption use successive iterations to converge to an approximation of the static power [77]. The device static power is known as the transistor leakage power and it is the power that is consumed when the FPGA is switched on but not configured [78].

The design static power depends on the design which is loaded in the FPGA. It represents the static power that is consumed by the design when the FPGA is configured but no clock and no inputs are applied hence no outputs are generated. The design static power also depends on the junction temperature, applied voltage and the configuration details. The larger the design is, the larger its use of logic gates and on-chip blocks would be. On-chip blocks are components inside the FPGA which already exist and can be accessed by the design such as BRAMs, PLLs and DCM clock generators, I/O terminators and transceivers. The static power is directly proportional to the cube of the applied voltage as shown in Eq. (6-21) [79]. Also, it is proportional to the junction temperature and transistor speeds as shown in Figure 6-5.

$$P_{Static} \propto V_{cc}^{3} \tag{6-21}$$

The static power increases with the junction temperature as explained in semiconductor physics due to Poole-Frenkel effect. The effect explains the thermal excitation received by an electron which exists in an electric field. The thermal energy is used to increase the energy of the electrons at the valance energy band which enables them to reach the conduction energy band. When more electrons reach the conduction energy band, higher current consumption is noticed. This is due to the fact that the current starts to flow by the electrons movement from one atom to another atom until they rest down after losing their thermal excitation energy.

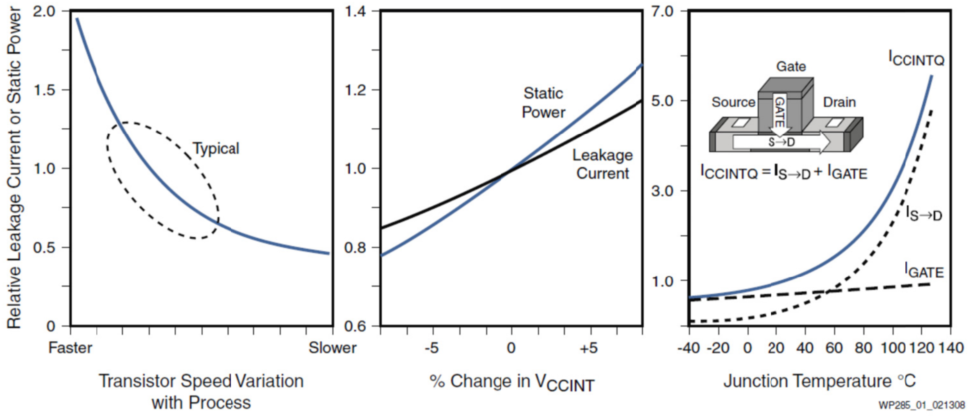


Figure 6-5. Change of Static Power with Process, Voltage and Junction Temperature in 65 nm SRAM-based FPGA Xilinx Virtex-5 **[79]**.

The design dynamic power is the power consumed by the design due to switching. It depends on the number of nodes, nodes' capacitances, applied voltage, and switching frequency. The dynamic power can be calculated from Eq. (6-22) [79].

$$P_{dynamic} = nCV^2f \tag{6-22}$$

Where: n is the number of toggling nodes, C is the node capacitance, V is the voltage swing, and f is the toggling frequency.

It is clear from the power consumption profiles of the SRAM-based FPGA that reducing the thermal power would take place by reducing the static and/or dynamic power. Reducing the static power can be done through reducing the temperature using heat sink or reducing the applied voltage by 5% to be 0.95 V. However, we prefer to use intolerant regulators that keep the power at 1 V, in order not to have power fluctuations. Reducing the dynamic power takes place through reducing the number of toggling nodes by reducing the configured logic such as using two processors instead of 4. Also, we can reduce the toggling frequency to reduce the dynamic power.

Xilinx introduced a new approach in manufacturing its FPGAs called triple oxide. That technique is used to reduce the static device power consumption increment, due to increase in the leakage current, which was expected due to reducing the transistor feature size. Xilinx have different gate oxides thickness in the transistors inside the FPGA: 1) Thin-oxide is used in the small. low threshold voltage, and fast transistors in the FPGA core, 2) Thick-oxide is used in higher voltage, swing transistors for I/O. and 3) Middle-oxide which is middle thickness transistors used in configuration memory cells and interconnect transistors which operate at higher threshold voltage than the thin-oxide transistors. Xilinx made use of the fact that the design uses hundreds of millions of transistors as configuration cells. It is meaningless to use the higher leakage thin-oxide fast transistors for storing the configuration data while they can be of lower leakage if middle-thickness transistors are used. That technique greatly reduces the leakage current, compared to the case if triple-oxide was not applied, while keeping the performance of the FPGA as required [80].

It is important to notice that reducing the junction temperature would increase the reliability and according to Xilinx it increases the lifetime by 2X [81].

6.3 Thermal Control Techniques

Thermal control is used to preserve the operating temperature limits for the different components inside the satellite. All electronic components have a temperature range within which they can operate. Thermal control tries to keep the lowest and highest operating temperatures, which a component might face, within the limits specified in the component datasheet. Thermal control techniques can be grouped into two main categories:

1) Active control
2) Passive control

Active thermal control is used to keep the thermal conditions within the desired operating ranges by using techniques that consumes electrical power. Heaters and Louvers are examples of active thermal control.

Passive thermal control uses techniques which do not consume electrical power to preserve the operating temperature limits. Multi-Layer Insulators, Heat sinks, Heat shields, thermal blankets, heat pipes, surface finishing and color, and radiators are examples of passive control techniques [73].

In the design of the MPSoC, we used DTM to actively control the junction temperature by reducing the dynamic power consumption through two methods:

1) Design reconfiguration by changing number of processor systems.
2) Operating frequency control by changing from 100 MHz to 50 MHz.

Also we used a passive technique to reduce the static power consumption through a low thermal resistance black anodized heat sink, thermal resistance was about 7 C˚/Watt. Combining all three techniques we get the following possibilities:

1) Heat Sink, 4 Processors, 100 MHz.
2) Heat Sink, 4 Processors, 50 MHz.
3) Heat Sink, 2 Processors, 100 MHz.
4) Heat Sink, 2 Processors, 50 MHz.
5) No-Heat Sink, 4 Processors, 100 MHz.
6) No-Heat Sink, 4 Processors, 50 MHz.

7) No-Heat Sink, 2 Processors, 100 MHz.
8) No-Heat Sink, 2 Processors, 50 MHz.

6.4 Geometric Thermal Model

The TMM is used to estimate the expected equilibrium temperatures of the system components before the actual thermal vacuum tests. The model is generated after defining the geometric thermal model of the system which is drawn in accordance to the real test chamber dimensions and boundary conditions. It can be used in extrapolating the design performance under different operating conditions as well as giving an initial estimation about the expected FPGA temperature during the tests.

It should be noted that the model has a limited accuracy within ±10 ˚C. The accuracy depends on the geometric modelling, estimation of the materials emissivity (ε), power consumed by the components, and thermal resistances models at conductions. It is difficult to estimate all of the conductance thermal resistances accurately, therefore testing is performed and the parameters are back annotated to the model to achieve better accuracy.

To establish the TMM the following steps are accomplished:

1) Build the geometric model of the system.
2) Set the thermal parameters of all the system components.
3) Define the mesh nodes.
4) Run simulation and perform results rehearsals.

The system geometric model was built using AutoCAD-2104. The correct dimensions of the system and its surrounding shroud during the thermal vacuum tests were defined. The thermal parameters were set through Thermal Desktop. The system consisted of a Printed Circuit Board (PCB) from glass-epoxy FR-4 with components attached to it. The PCB was placed in a sandwich-like structure between top and bottom Aluminum sheets. Iron bolts were used to attach the top sheet (ceil) to the bottom sheet (floor). The emissivity values for all of the materials were set based on their types. Twelve nodes were selected for running the simulations. The geometric model and the simulation results are shown in Figure 6-6 and Figure 6-7 respectively. The estimated FPGA temperature was about 97.74 ˚C while the actual test data showed values up to 105 ˚C.

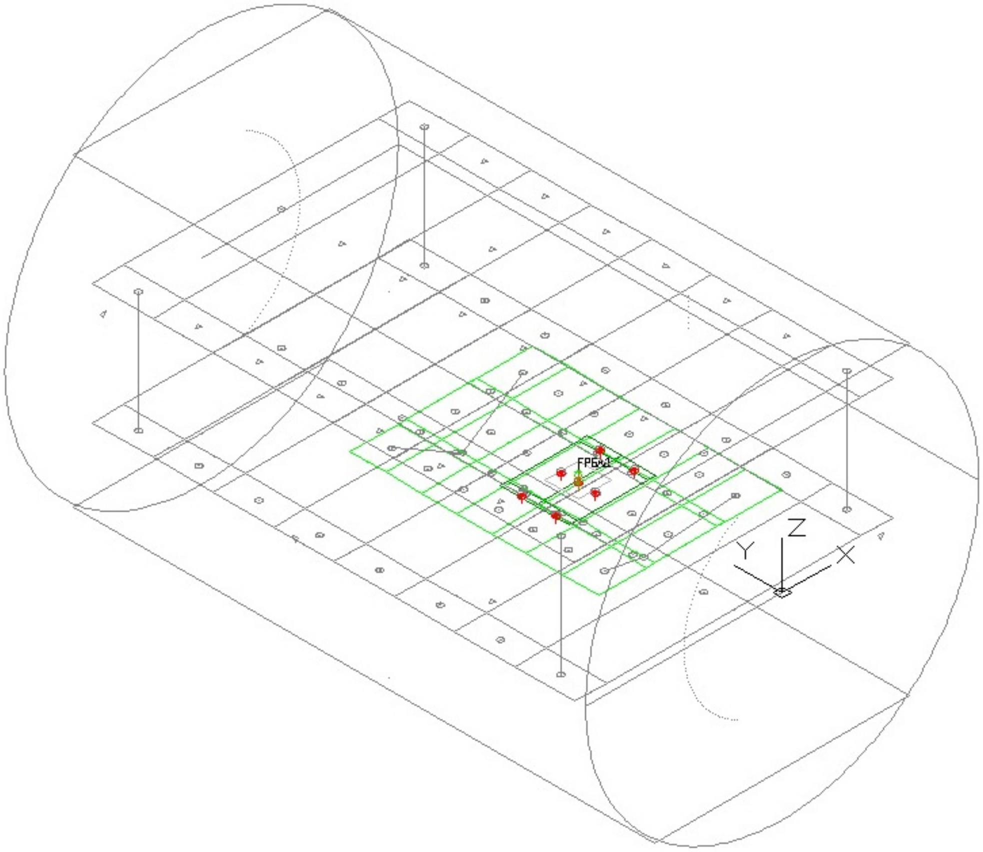


Figure 6-6. MPSoC OBC Geometric Thermal Model.

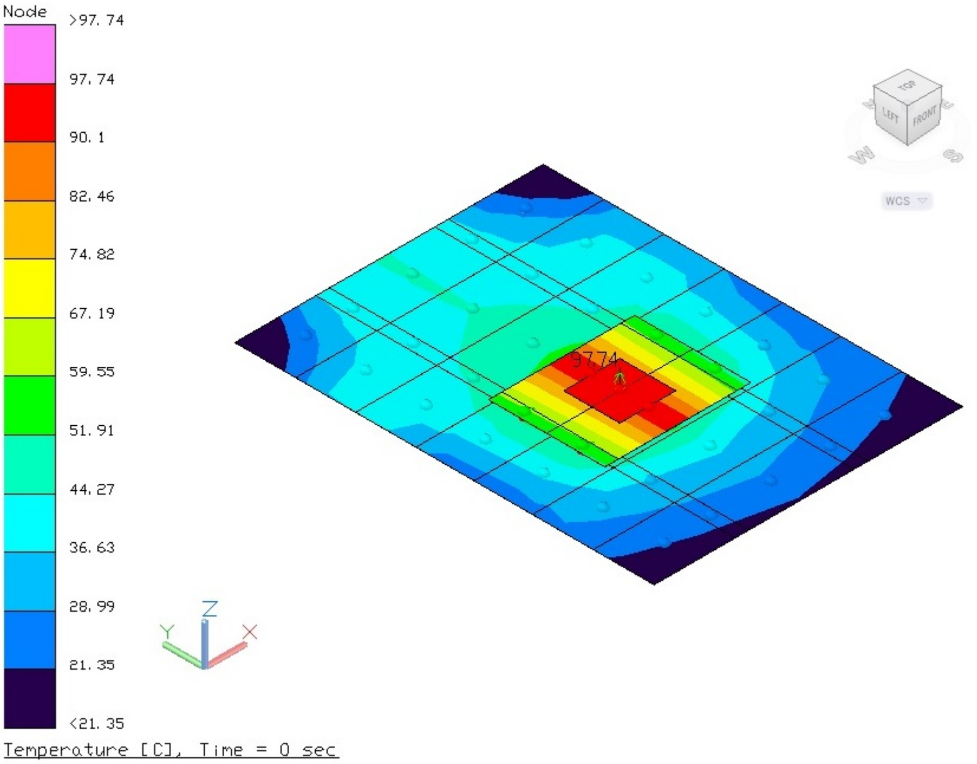


Figure 6-7. Thermal Mathematical Model Simulation Results.

6.5 Thermal Vacuum Test Objectives

The major thermal vacuum test objectives can be summarized in the following points [82]:

1) Verify the thermal margins of the system through which it can operate.
2) Test the effectiveness of the thermal mitigation techniques.
3) Judge the performance of the system in thermal vacuum environment at high and low cycles.
4) Validate the TMM through back annotation with the test results and rerunning of the simulation.
5) Identify anomalies in the system design, PCB manufacturing and/or assembling.
6) Testing the thermal stress effects on the PCB, soldering pads and soldering materials. Performing X-ray scan after the test is useful to identify any cracks in the soldering pads due to thermal stress. The Xilinx Virtex-5 FPGA package uses Ball Grid Array (BGA). The soldering material is Sn/Ag/Cu

solder. The Coefficient of Thermal Expansion (CTE) for the solder material and the glass epoxy FR-4 PCB are about 21 and 50 ppm/˚C respectively. These values of CTEs are good enough to keep the pads without cracks.

6.6 Thermal Vacuum Test Setup

Thermal vacuum testing of the MPSoC OBC took place at CeNT, Kyushu Institute of Technology, Japan. The center provides facilities for space environment testing of 50 kg satellites with sizes around 50 cm × 50 cm × 50 cm. Also, it contains facilities for testing of smaller satellite subsystems. We used a small thermal vacuum chamber, shown in Figure 6-8, during the tests to save the consumption of liquid nitrogen which was about 0.1 kg/hour.

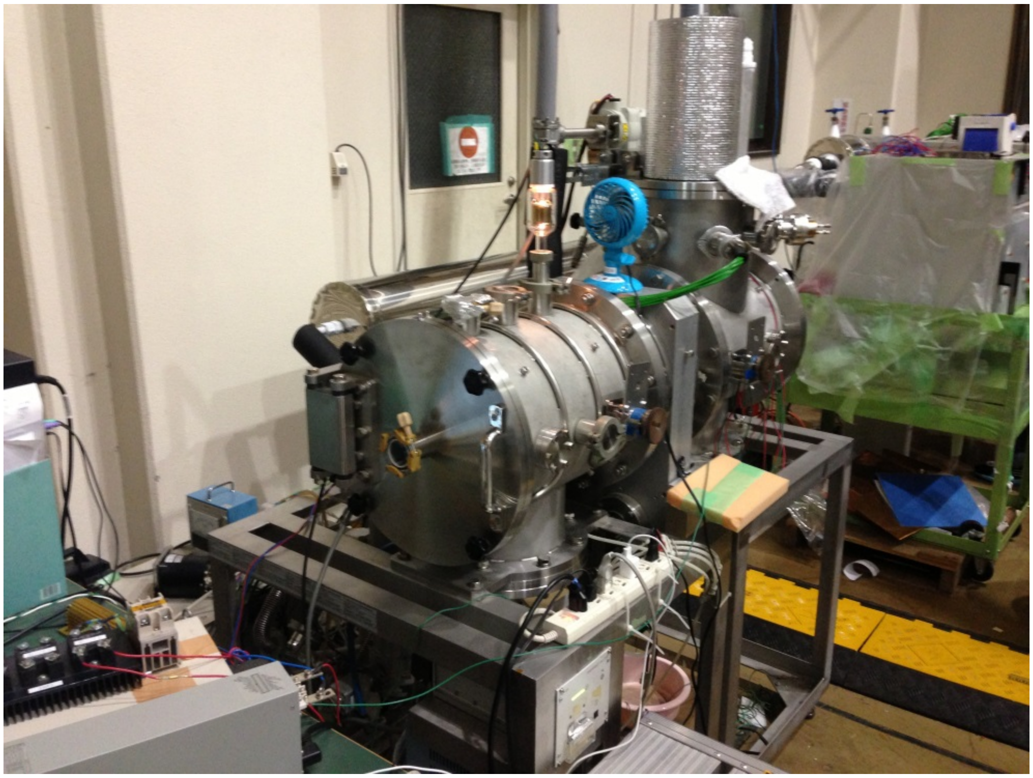


Figure 6-8. Small thermal vacuum chamber used during the tests.

The chamber outer diameter is about 40 cm, inner shroud diameter is 24.5 cm and the chamber depth is about 80 cm. Figure 6-9 shows the inner shroud dimensions in mm.

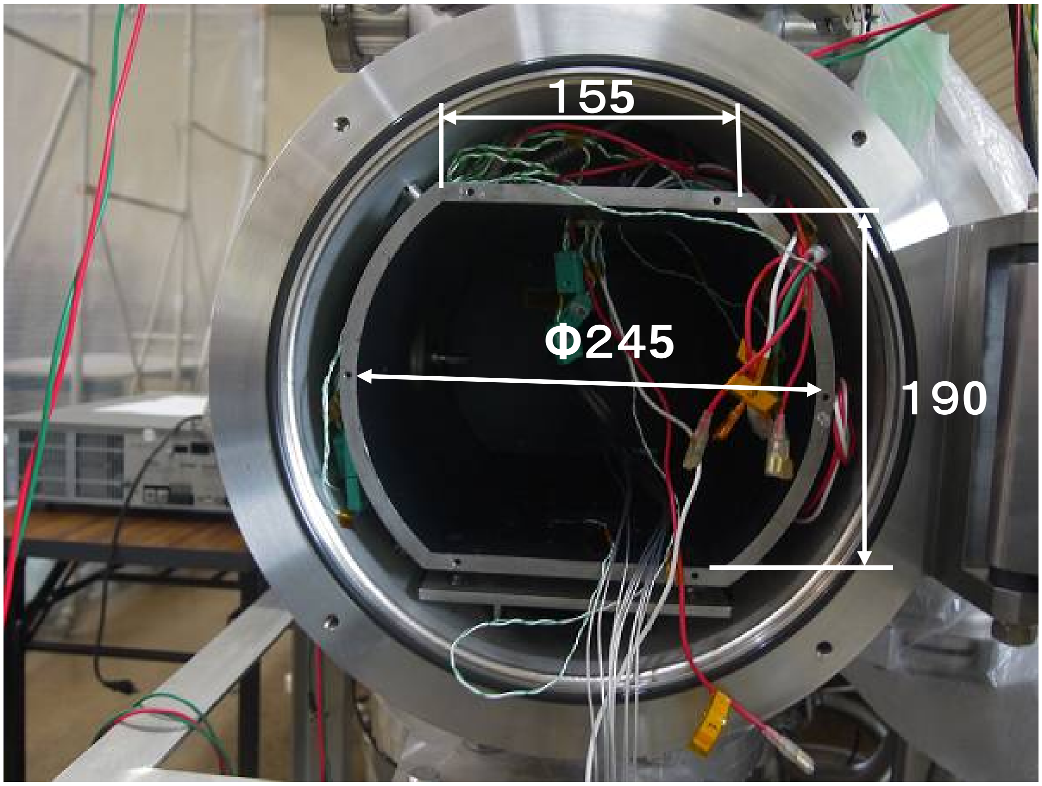


Figure 6-9. Thermal Vacuum Chamber Shroud Dimensions.

The pressure during the test reached 7.8 × 10^{-5} Pa. To reach that pressure we used a Turbo Molecular Pump. Liquid nitrogen was injected to maintain the shroud temperature in the range of -170 °C to -180 °C. The shroud acts as a Cryogenic pump, due to its cold surface, which helps in condensing the evaporating outgases from the device under test. A data acquisition system running on LabVIEW was used to collect the measurements from 10 temperature sensors, thermocouples, at a rate of 20 readings per second.

Figure 6-10, Figure 6-11, and Figure 6-12 show the temperature sensors attachments during the thermal vacuum test. Three temperature thermocouple sensors labeled (2), (3) and (4) were located on the FPGA top surface. These sensors are attached to the heat sink side walls when the heat sink was used to reduce the temperature as shown in Figure 6-12. Two sensors were located on the PCB board labeled (5) and (6). Sensor (7) was located on the edge of the board close to the mounting point. This sensor is used as the control point for achieving the test temperature profile. Sensor (8) was located at the bottom side of the PCB underneath the FPGA. Sensors (9) and (10) were used for the shroud top and bottom respectively. Sensor (1) was used to monitor the polyimide heater temperature. The polyimide heater was used to raise the system temperature and simulate the total effect of the heat input to the system from the sun, albedo and reflection from the earth surface.

The test temperature profile was achieved by heating up the system and monitoring the sensor at point (7). The closed loop control was implemented using LabVIEW. The temperature of point (7) was kept at 60 C° during the hot soaking time and at -24 C° during the cold soaking time to match with the SMC acceptance test requirements [83]. The soaking time was initially two hours in hot temperature and two hours in cold temperature without applying the mitigation techniques. The soaking time was reduced to one hour while testing the previously mentioned 8 thermal control mitigation techniques. Two test cycles were conducted at each test. The system was assembled in a sandwich like structure, Figure 6-13 shows the assembly that was placed in the test chamber. The system setup schematic is shown in Figure 6-14.

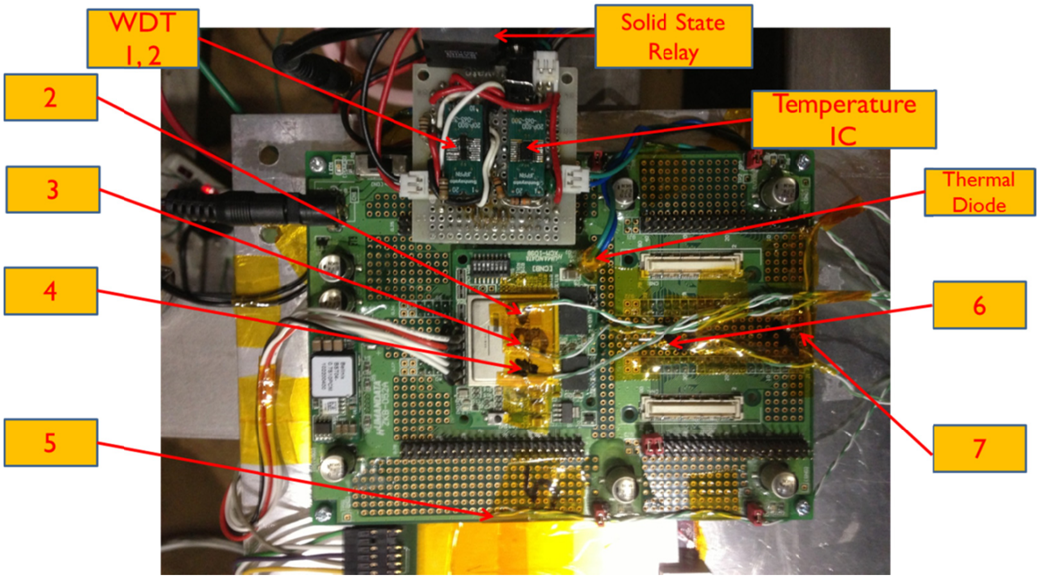


Figure 6-10. Thermocouples Positions During Thermal Vacuum Tests - Top Positions.

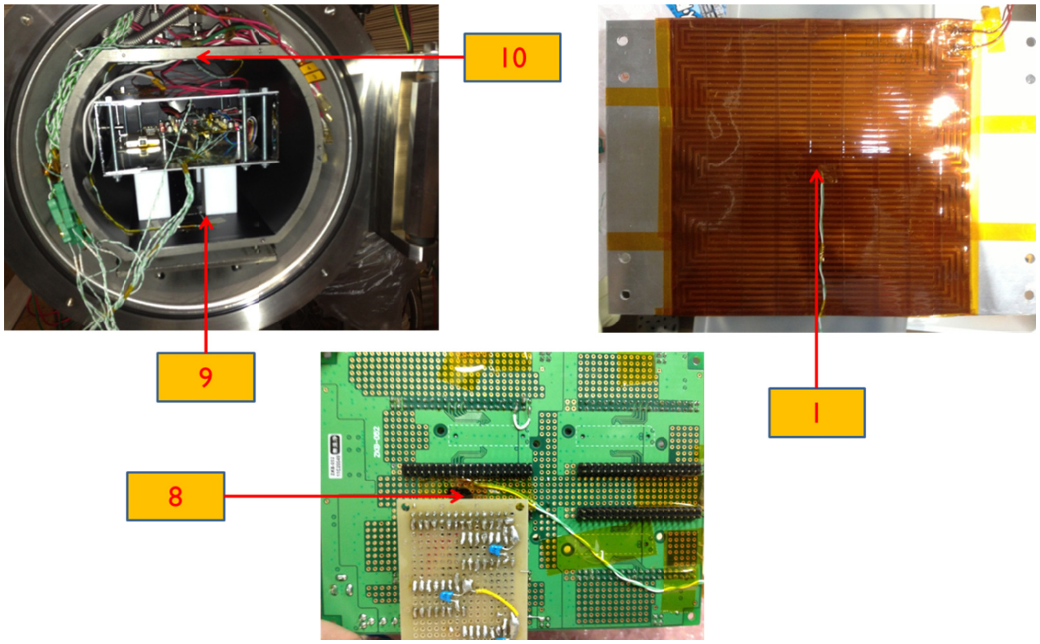


Figure 6-11. Thermocouples Positions During Thermal Vacuum Tests - Bottom Positions.

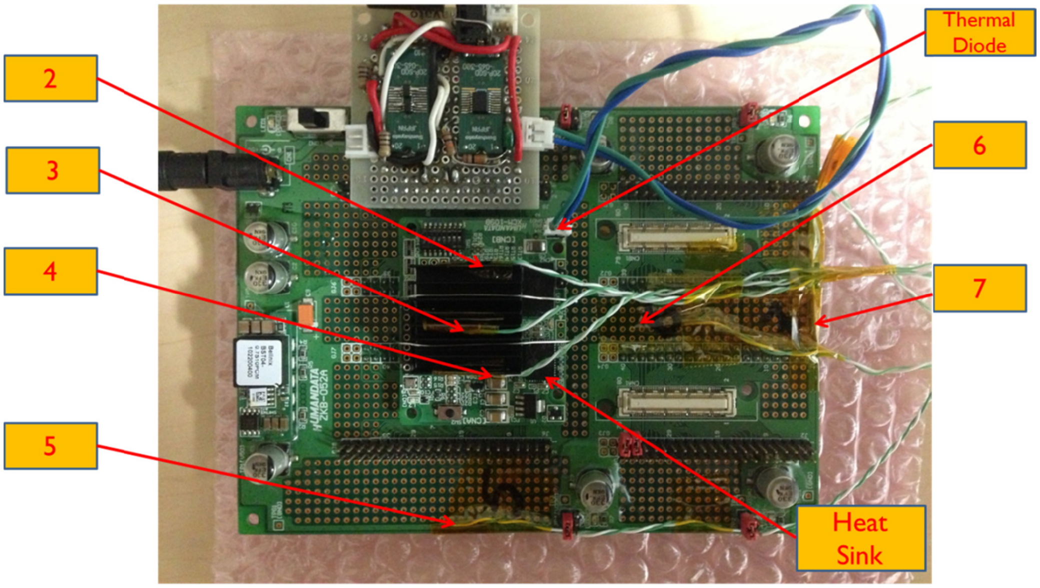


Figure 6-12. Thermocouples Positions During Thermal Vacuum Tests - Top Positions - with Heat Sink.

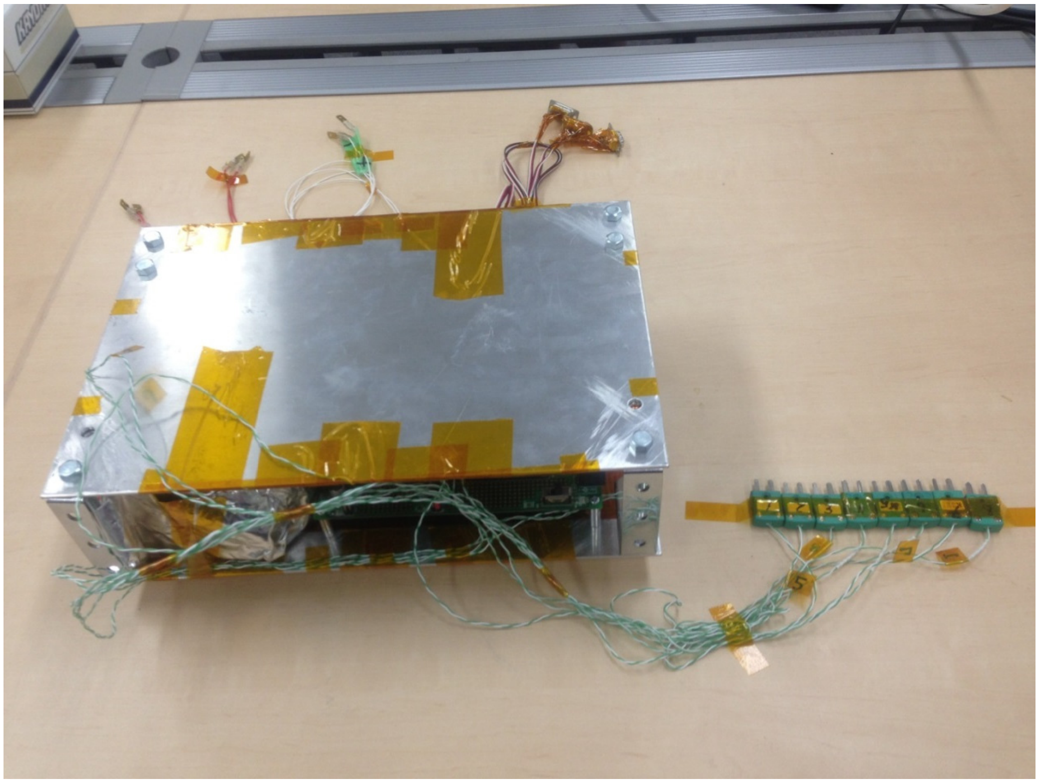


Figure 6-13. Assembled Sandwich Structure Before Thermal Vacuum Testing.

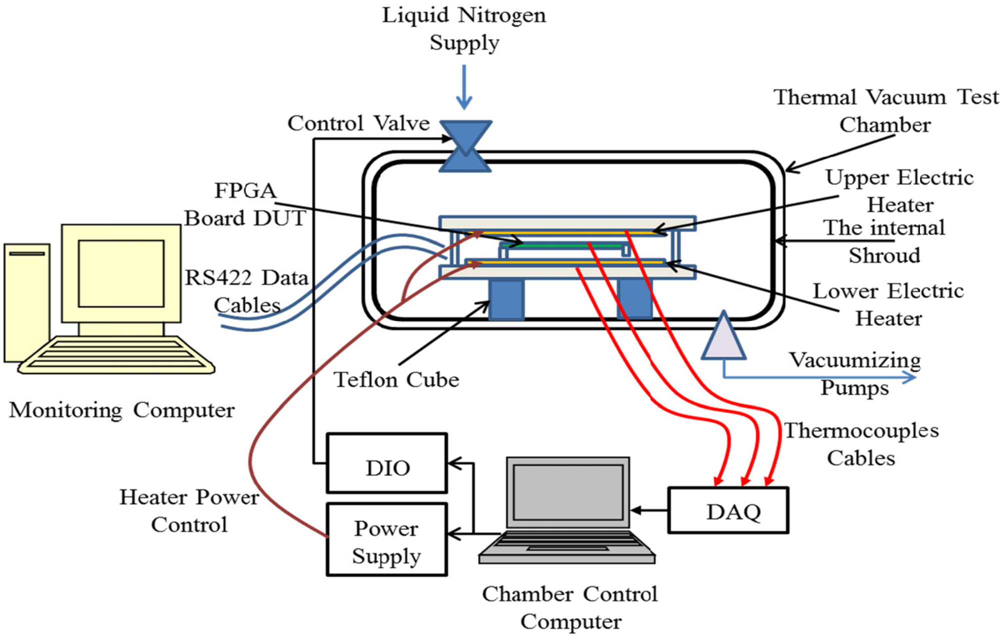


Figure 6-14. Thermal Vacuum Test Setup Schematic.

The schematic in Figure 6-14 shows the setup of the system inside the chamber and the outside connections as well. The system is placed in a sandwich structure where it was surrounded by upper and lower Aluminum metal sheets of A4 size. The thermocouples where attached to different locations as mentioned before. The data from the thermocouples was collected by LabVIEW through a DAQ module. The chamber control computer maintained the shroud temperature at around -170 °C by controlling the opening of an electric valve to supply liquid nitrogen inside the chamber.

The control was carried out using LabVIEW through a DIO module. Also the chamber control computer controlled the power supply to the heater in order to maintain the required target profile temperature. The feedback for the shroud temperature was provided to the control computer through the sensors attached to the top and bottom of the shroud surfaces. The feedback for the target point temperature was provided through the sensor labeled (7) in Figure 6-10 and Figure 6-12. An additional heater was attached to the lower Aluminum sheet as an optional heating element but we did not need to use it during the tests. Monitoring computer was collecting the data from the software running on the processor systems during the thermal vacuum test to judge their correct operation. The interface between the FPGA, DUT, and the monitoring computer used serial communication.

A circuit was implemented to measure the internal die temperature of the FPGA through the internal FPGA thermal diode. The circuit used the temperature measurement IC MAX6655 which was interfaced to the MSCP through I2C link. Two external WDTs, MAX6369, timer were used to detect the failure of the FPGA. The MSCP was resetting the WDTs which were connected in parallel to the Crydom MPDCD relay. Whenever any of the WDTs fails to reset its internal timer, based on a reset signal received from the MSCP, it asserts a low signal on its output which switches off the relay control side. Power is recycled on the relay output side and the FPGA is restarted. WDT asserts the low signal for one clock cycle. The WDT timer timeout period was set to 15 sec.

6.7 Thermal Vacuum Test Procedure

The software that was running during the tests was developed to achieve a full test cycle of all the processor systems in the MPSoC design. The algorithms of the software that run on the master processor and the slave processors is shown in Figure 6-15 and Figure 6-16. The internal WDT in the FPGA which are connected to the master and slave processors kept generating an interrupt every 5.4 sec. When the slave processors receive the interrupt signals through their INTC, they invoke an ISR. The routine starts by sending an acknowledgement message of four characters (ACK0) to the master processor to inform it that the slave processor is alive and to test the mailbox connection between the master and slave processors. The slave processor starts an internal timer to count the interval between the successive interrupts received from the WDT. The internal timer value is captured by the next new WDT interrupt and the internal timer is restarted. The slave processor reports its status to the external monitoring computer together with the value of its internal timer through the UART interface. The last step is that the slave processor restarts the WDT to allow for a new interrupt to take place. If the slave processor fails to restart the watch dog timer it would be restarted by a reset signal to be sent from the WDT to the reset module of the processor.

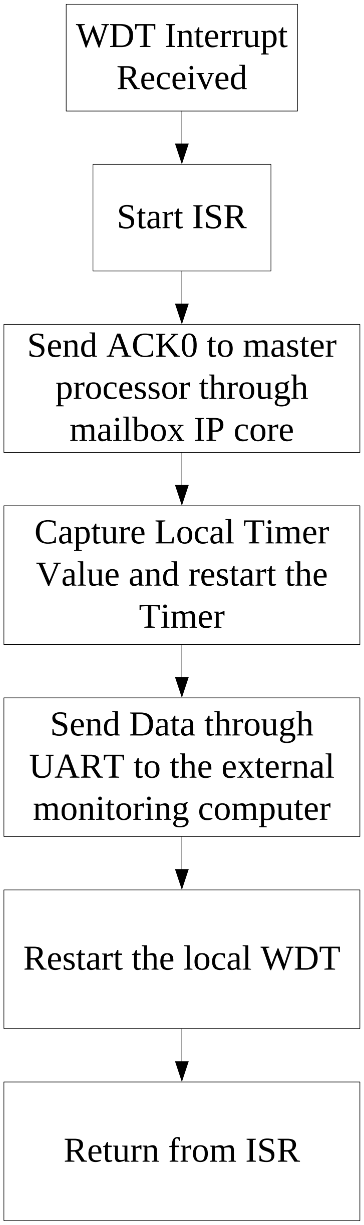

WDT Interrupt Received
Start ISR
Send ACK0 to master processor through mailbox IP core
Capture Local Timer Value and restart the Timer
Send Data through UART to the external monitoring computer
Restart the local WDT
Return from ISR

Figure 6-15. Algorithm Executed at the Slave Processor at Thermal Vacuum Test.

The master processor receives interrupts from its local WDT every 5.4 sec. The master and slave processors work on a master system clock so the interrupts timings are synchronized. The master processor ISR starts by reading the mailboxes FIFOs to receive the (ACK0) messages sent from the slave processors. If no (ACK0) is found from any processor then the whole system is reset and FPGA is reloaded. This takes place by not resetting the external WDT which leads to power recycling of the system through the signal sent from the external WDT to the SSR. Data is sent through UART to the monitoring computer.

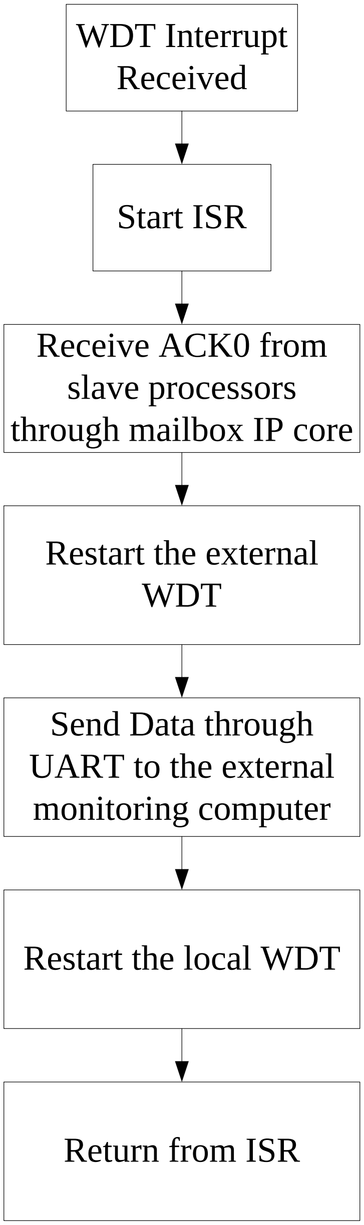

WDT Interrupt Received
Start ISR
Receive ACK0 from slave processors through mailbox IP core
Restart the external WDT
Send Data through UART to the external monitoring computer
Restart the local WDT
Return from ISR

Figure 6-16. Algorithm Executed at the Master Processor at Thermal Vacuum Test.

Before conducting the full thermal vacuum test, we conducted a check test to make sure that the FPGA would perform an automatic shutdown if the junction temperature reaches 125 ˚C. We conducted durability test to make sure that the FPGA would withstand the thermal cycling under vacuum condition. However, we noticed that the FPGA operating temperature was exceeding the recommended commercial grade upper limit of 85 ˚C during the operation of the 4 processor cores. We had an initial test cycle to better characterize the FPGA temperature profile in thermal vacuum testing. Sensors were attached as shown in Figure 6-17. The results shown in. Figure 6-18 were obtained. Thus, based on the results of the FPGA temperature profile, we decided to mitigate the design using the 8 previously mentioned thermal control techniques and repeated the tests again. Thermal mitigation test results are discussed in chapter 7.

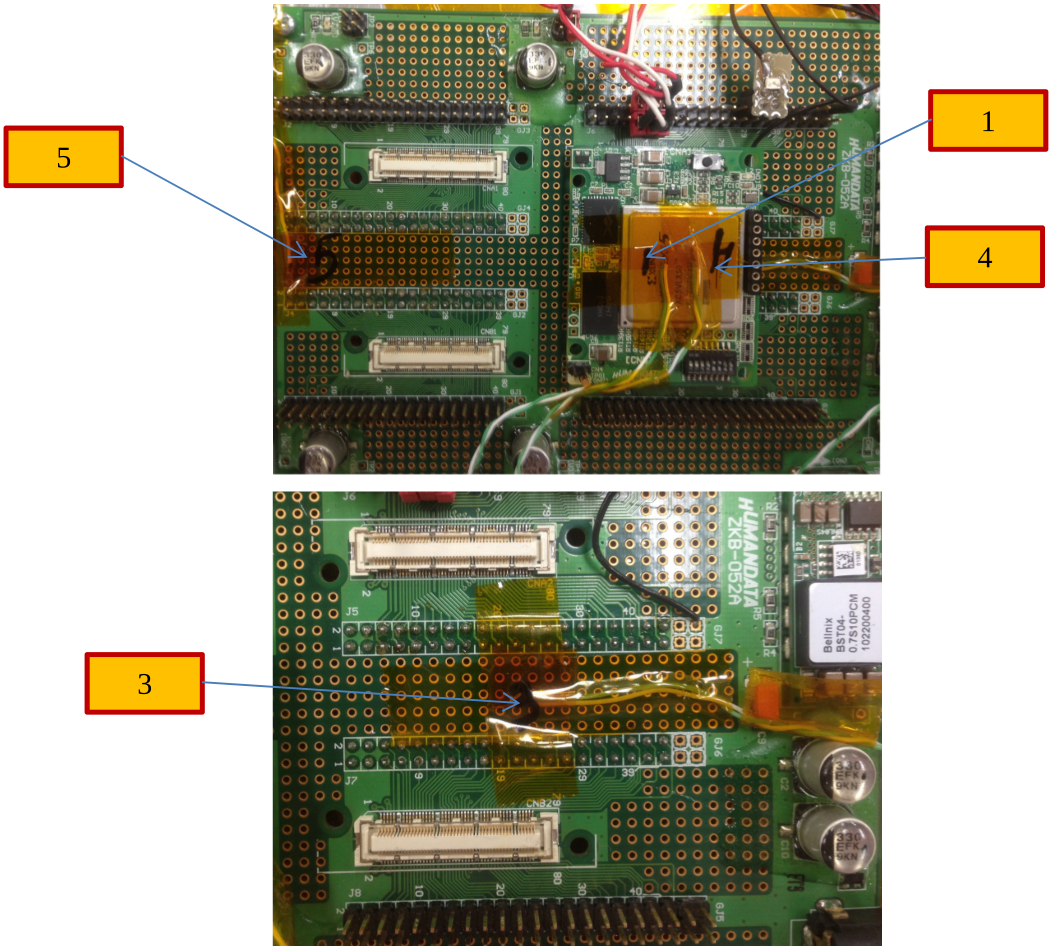


Figure 6-17. Initial Screening Thermal Vacuum Test Sensors Attachemnt.

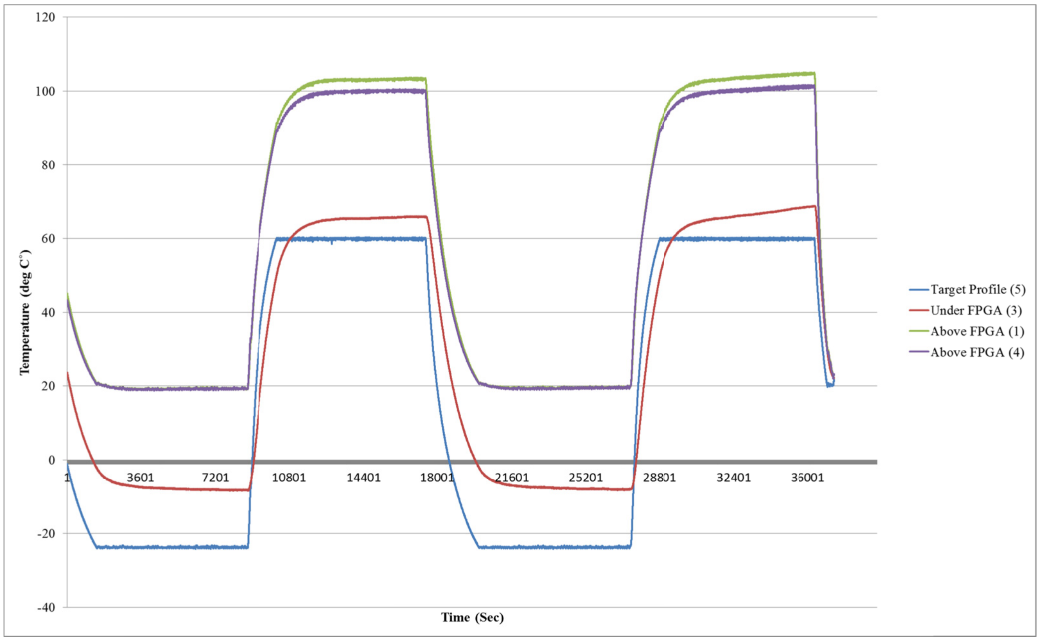


Figure 6-18. Initial Screening Thermal Vacuum Test Results **[67]** ©IEEE 2013.

In this chapter we presented the thermal vacuum test setup and procedure. The initial screening of the FPGA temperature profile showed an over limit performance when the 4 processor cores were operating simultaneously. Mitigation techniques to reduce the operating temperature were suggested based on the concept of reducing the static and dynamic power consumptions. In the next the chapter we present the details of the tests' results and discussion about their significance.

Chapter 7 : Results and Discussion

The results of the space and time redundancy systems performance, the radiation test, and the thermal vacuum test are presented in this chapter. The discussion about the meaning and significance of the results is covered.

7.1 Time and Space Redundancy Performance

The time and space redundancy systems were tested to execute the bubble sort algorithm using vectors of randomly generated data words as described in chapter 4. The time span of execution was collected for the non-fault tolerant system, the time redundant system and the space redundant system. Figure 7-1, Figure 7-2, and Figure 7-3 shows the execution times histograms of the three cases. The execution times data were collected and grouped to depict their statistical distributions. We made curve fitting on the grouped data of each case using normal distribution. The graphs in the figures show the red line of the normal distribution fitting and the blue bars of the grouped data histograms.

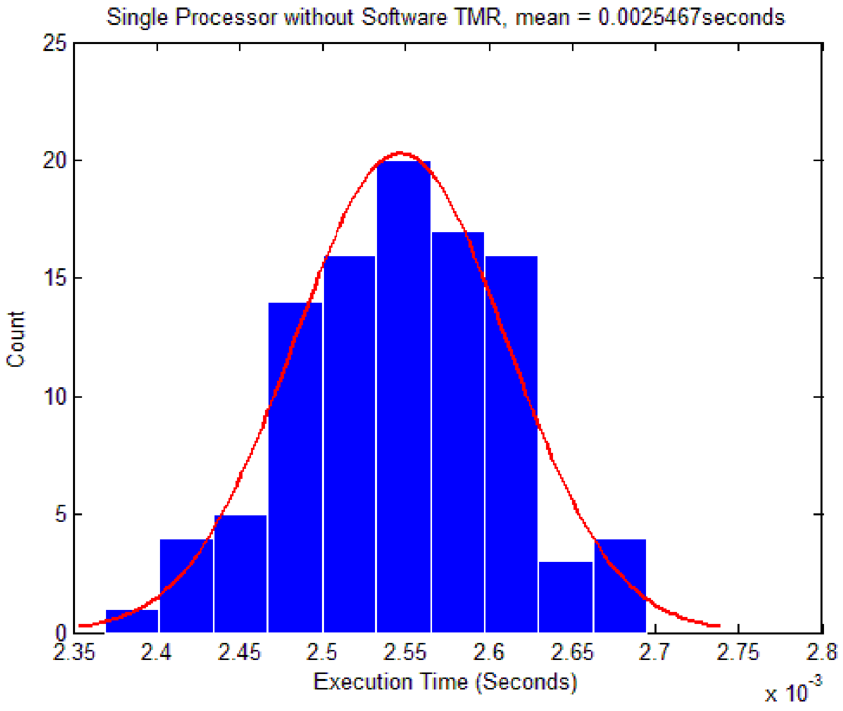


Figure 7-1. Non-Fault-Tolerance Execution Time Histogram.

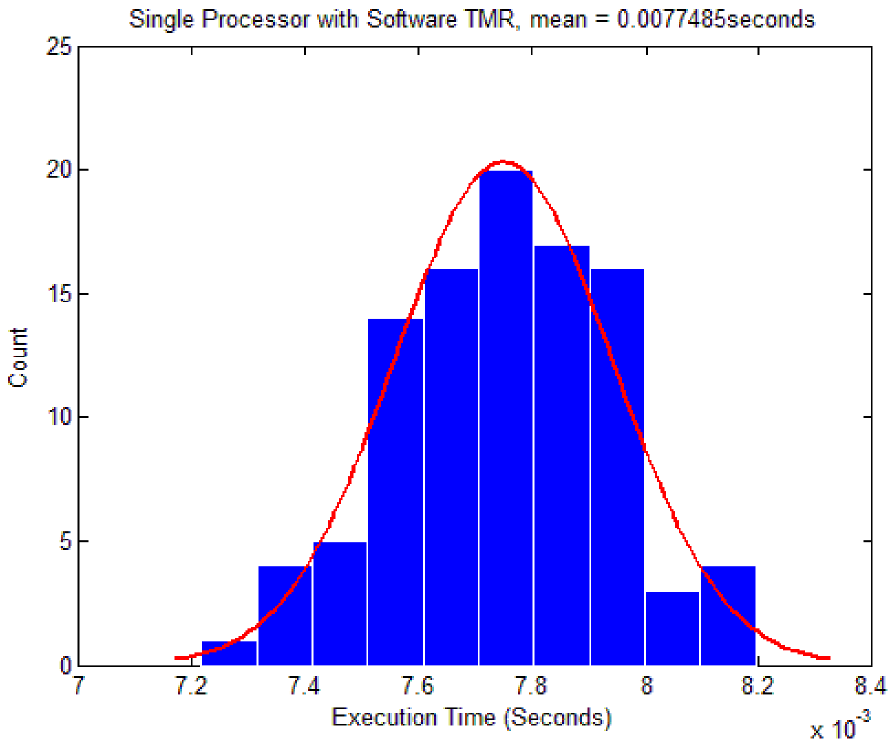


Figure 7-2. Time Redundancy Fault-Tolerance Execution Time Histogram.

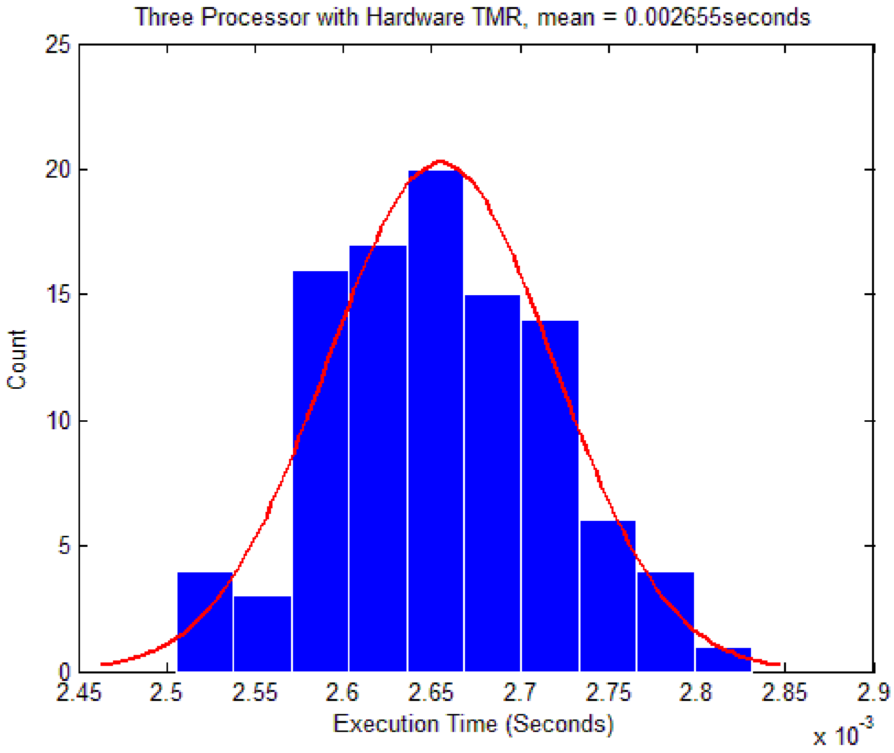


Figure 7-3. Space Redundancy Fault-Tolerance Execution Time Histogram.

Each time a random sequence vector was generated, based on the clock seed of the computer system running MATLAB, to avoid repeating patterns of random sequences. The length of each vector was 100 words. The numbers of vectors applied during the test were 100 vectors. The statistical parameters of the normal distribution fitting of each histogram are shown in. Table 7-1.

Table 7-1. Statistical Analysis of Execution Times Histograms in seconds.

Parameter	**No-Fault Tolerance**	**Time Redundancy**	**Space Redundancy**
Min	0.002354	0.007171	0.002463
Max	0.002739	0.008326	0.002847
Mean	0.002547	0.007749	0.002655
Median	0.002547	0.007749	0.002655
Mode	0.002354	0.007171	0.002463
Std	0.0001129	0.0003386	0.0001127
Range	0.0003852	0.001156	0.0003847

The mean execution time of the single processor without software TMR is about 2.55 ms. It is almost the same as the mean execution time of the hardware TMR, space redundancy, which is 2.65 ms. The mean execution time of the single processor with software TMR, time redundancy, is 7.75 ms. Almost 3 times higher than the non-fault tolerant and the space redundancy fault tolerance cases. This means that hardware redundancy is better in terms of execution time as it is as fast as the non-fault tolerant system which contains no overheads. However, we have to carefully notice that adding complicated data exchange protocols between the processors will add an execution overhead hence increasing the execution time significantly. If too much time is spent in handling the communication between the processors then the execution time might be close to the time redundancy case.

The power consumption varies between the single core and the multicore systems as was shown in chapter 4, Table 4-1. The power consumption was estimated using the XPower Analyzer Tool from the Xilinx-ISE toolset. The total power consumed by the multi-core system is about 1.26 Watt which is only about 28% higher than the power consumed by the single core system.

The total power consumption consists from: 1) dynamic power, and 2) static power. The dynamic power consumption is affected by the size of the design and its operating frequency. The static power consumption is mainly affected by the transistor feature size of the FPGA, junction temperature and the operating voltage. The static power consumption in the FPGA transistors is considered as leakage. The leakage current and hence the leakage power increase as the feature size of the FPGA transistors is minimized. This leakage power is consumed even if the FPGA is not configured with any bitstream. About 0.46 Watt is dissipated in the form of leakage power in the multicore and single core designs.

Table 4-1 in chapter 4, represents the dynamic power consumed by the design resources: The on-chip clock trees, the logic gates, the design signals, the Block Random Access Memories (BRAM), the Digital Signal Processors (DSP) which has a value of zero as we did not use DSPs in the design, the Phase Locked Loops (PLL) and Digital Clock Managers (DCM) for generating the required operating frequency and the Input/Outputs (IO) logic. The difference in the dynamic power consumption between the two designs is about 0.278 Watt. This difference is due to the use of more cores, which means the use of more logic, signals, BRAMs, IOs and mostly more clock-trees, in implementing the multicore system. From Table 4-1 in chapter 4, we conclude that the dynamic power consumption is the main responsible for the difference in the total power consumption. The FPGA resources are almost utilized by the multi-core system while about less than one-third is utilized by the single core system as shown in Table 4-2 and Table 4-3 in chapter 4.

This comparative study investigated the effects of using hardware redundancy and software redundancy on the resources utilization, power consumption, execution speeds and reliability in single and multi-core designs of the Virtex-5 FPGA. The multi-core system makes better use of the resources and it executes at almost triple the speed of the single core system while its power consumption is only 28% higher than it. Thus, the multi-core system which uses space redundancy for implementing fault tolerance in the 65 nm Virtex 5 FPGA through repeating redundant hardware processor cores is more efficient and effective than the single core design. In terms of reliability the TMR design is better than the single core design as long as the time between scans is kept below a specific threshold which is 606 sec for this design as was explained in chapter 4. Table 7-2 summarizes the time and space redundancy comparative study results.

Table 7-2. Summary of Time and Space Redundancy Architectures Comparison.

Parameter	Time Redundancy	Space Redundancy	Evaluation
Performance	X	3X	**Good**
Power	X	1.3X	**Moderate**
Reliability (at T_{scan} = 100sec)	X	1.1X	**Good**
Utilization	X	2.1X	**Good**

It is recommended according to the obtained results to make use of space redundancy approaches when designing digital systems using the Xilinx Virtex5 (65nm) FPGA. This is valid due to the fact that considerable portion of the power consumed is dissipated in the form of leakage power. Adding extra logic did not add much to the total power consumed. The reliability of the space redundant system is higher or at least equal to the time redundant system below the critical threshold of the time between scans, and its execution speed is better as far as the communication protocol between the cores does not add much overhead. Resources are better to be utilized in the FPGA device rather than wasting them. The space redundant system makes higher utilization of the FPGA resources. That is an advantage for the space redundant over the time redundant systems in case no additional design modules are needed to be added in the design. In case larger packages are needed then it would be straight forward to use the larger Virtex5-LX85 which is pin compatible with virtex5-LX50. However, we should note that larger package means higher device and design static power consumptions.

7.2 Static Cross Section from Accelerated Proton Beam Test

The radiation test of the XC5VLX50-1FFG676C FPGA, which has a 65 nm technology, was conducted at TARRI to estimate the static cross section using a new method. We designed the method to make use of a single FPGA. Traditional radiation test setup contains two main sections: 1) The DUT, and 2) the monitoring and control system. In our approach we used the SEU controller IP core to count the number of frames containing errors in every 128 full scans of the configuration FPGA frames. We approximated the problem by calculating the event bit cross section instead of the upset bit cross section. This means that the frame errors are counted as a single event therefore there would be no need to know the actual number of upset bits. Nevertheless, we should always consider that the actual situation in-orbit is a combination of both single and double errors. Triple and more bit errors are rare to occur [36]. The TARRI test results are compared to the test results previously obtained in the experiment of the Los Alamos National Laboratory (LANL) and NASA Goddard Space Flight Center (GSFC) at Indiana University Cyclotron Facility (IUCF) [64]. That experiment was conducted on the same commercial grade FPGA, Virtex 5 LX50, at an energy level of 65 MeV of accelerated proton beam. The event bit static cross section obtained in NASA's experiment was ($6.37\text{x}10^{-14} \pm 1.17\text{x}10^{-15}$ cm^2/bit).

Figure 7-4 shows all of the radiation test results at TARRI in one graph. The results are collected during 4 radiation sessions. Three sessions were at a flux level of 10e06 ($\text{cm}^{-2}.\text{s}^{-1}$) and one session at a level of 3e06 ($\text{cm}^{-2}.\text{s}^{-1}$). All the sessions continued to collect data until the SEU controller recorded 128 configuration frame errors.

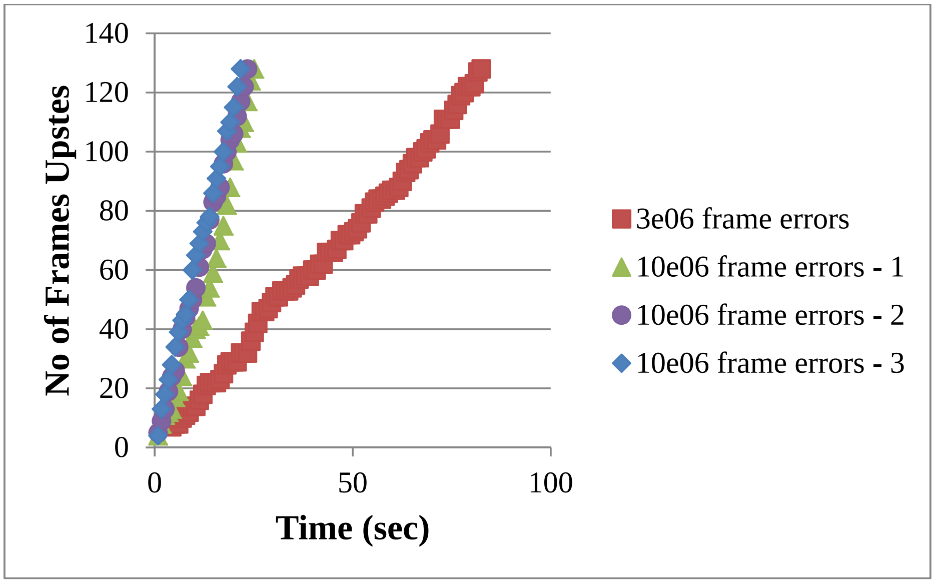


Figure 7-4. Accumulated No. of Frame Upsets versus Time at Flux Levels of 10e06 and 3e06 ($\text{cm}^{-2}.\text{s}^{-1}$) – Energy Level of 65 MeV.

Figure 7-5 to Figure 7-8, show the individual results of conducting radiation tests using flux levels of 10e06 ($cm^{-2}.s^{-1}$) and 3e06 ($cm^{-2}.s^{-1}$). The graphs show the number of configuration frames that suffered multiple bit upsets which could not be corrected by the SEU controller core. The graphs are linearly curve fitted with the equation representing the curve shown as y(x). The R^2 value is the coefficient of determination which is calculated to judge the goodness of the curve fitting to the plotted data.

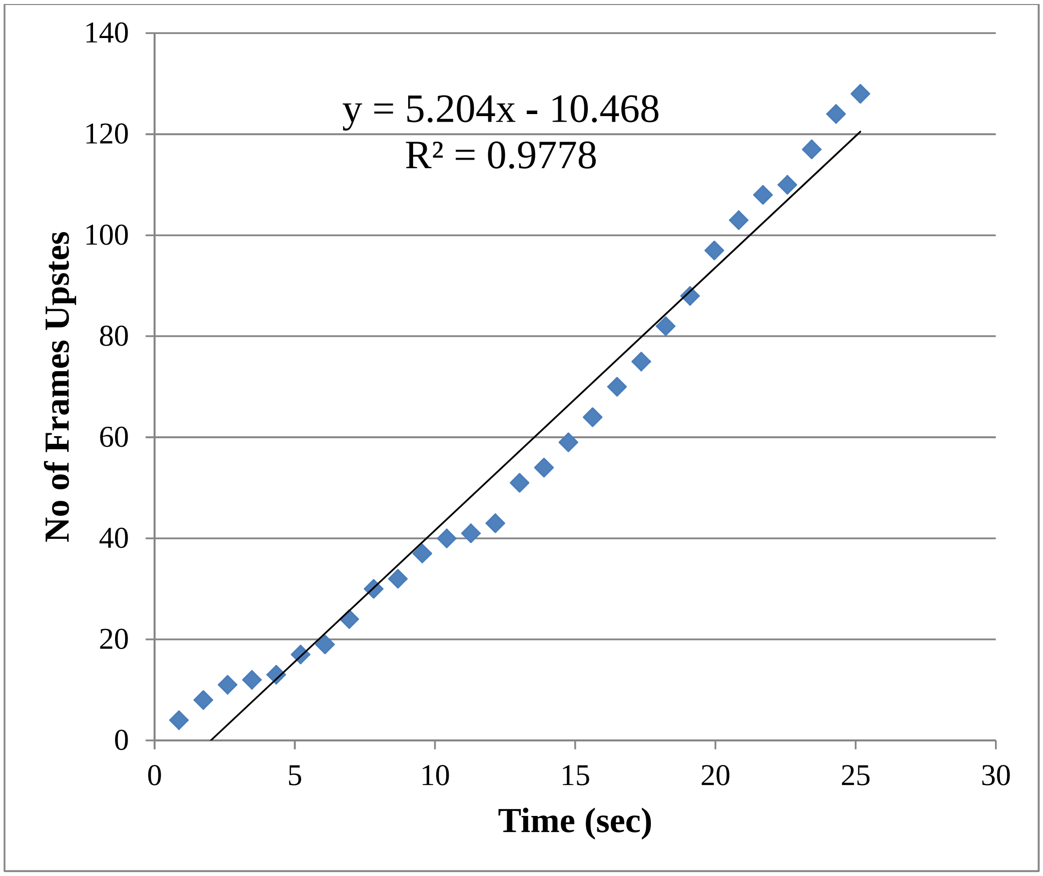


Figure 7-5. Accumulated No. of Frame Upsets versus Time at Flux Level of 10e06 ($cm^{-2}.s^{-1}$) – Energy Level of 65 MeV – 1st Trial.

Our average static cross section value over the 4 radiation sessions is (6.34e-14 cm^2/bit) while NASA's value is (6.37e-14 ± 1.17e-15cm^2/bit). If we consider the differences in devices accuracies between TARRI and NASA then we judge our results as valid and accepted. Our proposed technique is new and simple when compared to the traditional techniques for calculating the static cross section [19]. We used the DUT with an embedded SEU controller IP core, which performed read-back of the configuration frames through the ICAP, instead of the traditional external scrubbing through JTAG or Select-MAP interface. The SEU controller core design is ultimately very small therefore no failures were detected in its operation during the short exposure times to radiation. The traditional methods are complex and have more connections as they perform external scrubbing, monitoring and control through a specific FPGA or microcontroller. More complex setups are not welcomed in radiation test experiments especially inside the radiation chamber.

In order to calculate the static event cross section as in Eq. **(5-5)**, the actual device size in bits has to be estimated. The actual device size depends on the number of bits being exposed to radiation. The opening in the copper shield, through which the proton beam was projected to the FPGA, was about 12 mm in diameter. This gives an exposed area of approximately 1.13 cm^2. However, the FPGA die area is 1.44 cm^2 (12 mm × 12 mm). The amount of configuration words which can be read back in the scanning process initiated by the SEU controller are 339,200 words [84]. Each word contains 32 bit, therefore a total of 10854400 configuration bits can be read back by the SEU controller. By multiplying the total number of configuration bits by the area ratio (1.13/1.44), we get the total number of bits in the area exposed to radiation which is equal to 8525025.826 bits. By applying Eq. **(5-5)**, we get the static event cross sections shown in Table 7-3 to Table 7-6. The average static event cross section from the four measurements is 6.34e-14 cm^2/bit.

Table 7-3. Static Cross Section Calculations at Flux Level of 10e06 ($cm^{-2}.s^{-1}$) – Energy Level of 65 MeV – 1st Trial.

Flux level ($cm^{-2}.s^{-1}$)	10e06
Exposure time (s)	25.16736
No. of frame errors (events)	128
Device size (bits)	8525025.826
Static event cross section	5.97e-14

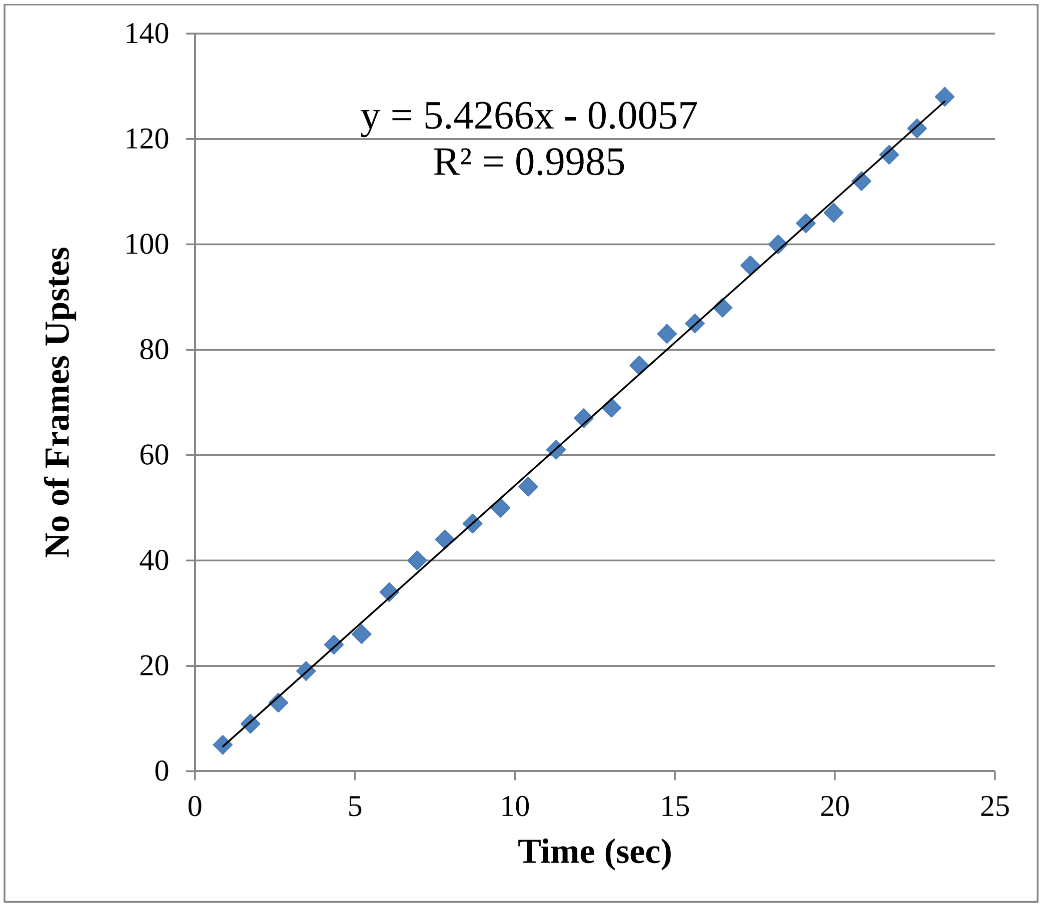


Figure 7-6. Accumulated No. of Frame Upsets versus Time at Flux Level of 10e06 (cm^{-2}.s^{-1}) – Energy Level of 65 MeV – 2nd Trial.

Table 7-4. Static Cross Section Calculations at Flux Level of 10e06 (cm^{-2}.s^{-1}) – Energy Level of 65 MeV – 2nd Trial.

Flux level (cm^{-2}.s^{-1})	10e06
Exposure time (s)	23.43168
No. of frame errors (events)	128
Device size (bits)	8525025.826
Static event cross section	6.41e-14

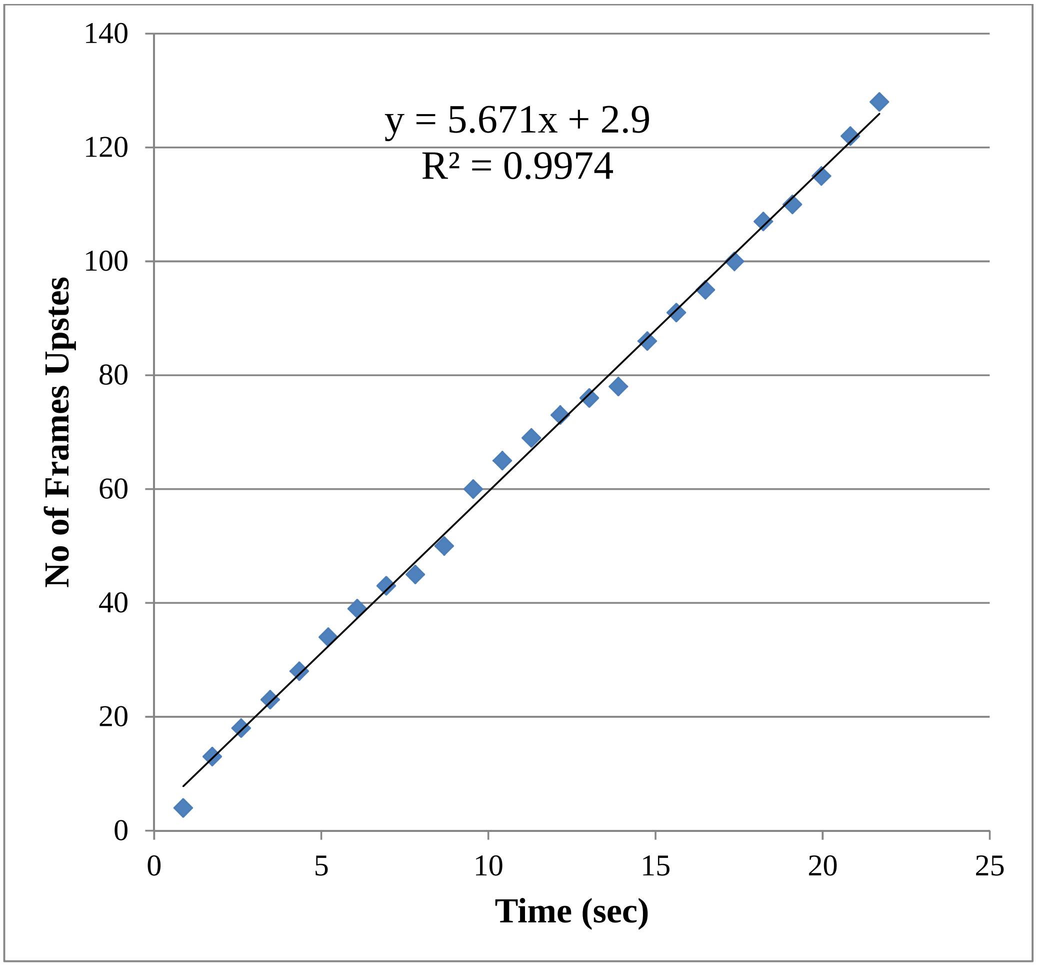


Figure 7-7. Accumulated No. of Frame Upsets versus Time at Flux Level of 10e06 (cm^{-2}.s^{-1}) – Energy Level of 65 MeV – 3rd Trial.

Table 7-5. Static Cross Section Calculations at Flux Level of 10e06 (cm^{-2}.s^{-1}) – Energy Level of 65 MeV – 3rd Trial.

Flux level (cm^{-2}.s^{-1})	10e06
Exposure time (s)	21.696
No. of frame errors (events)	128
Device size (bits)	8525025.826
Static event cross section	6.92e-14

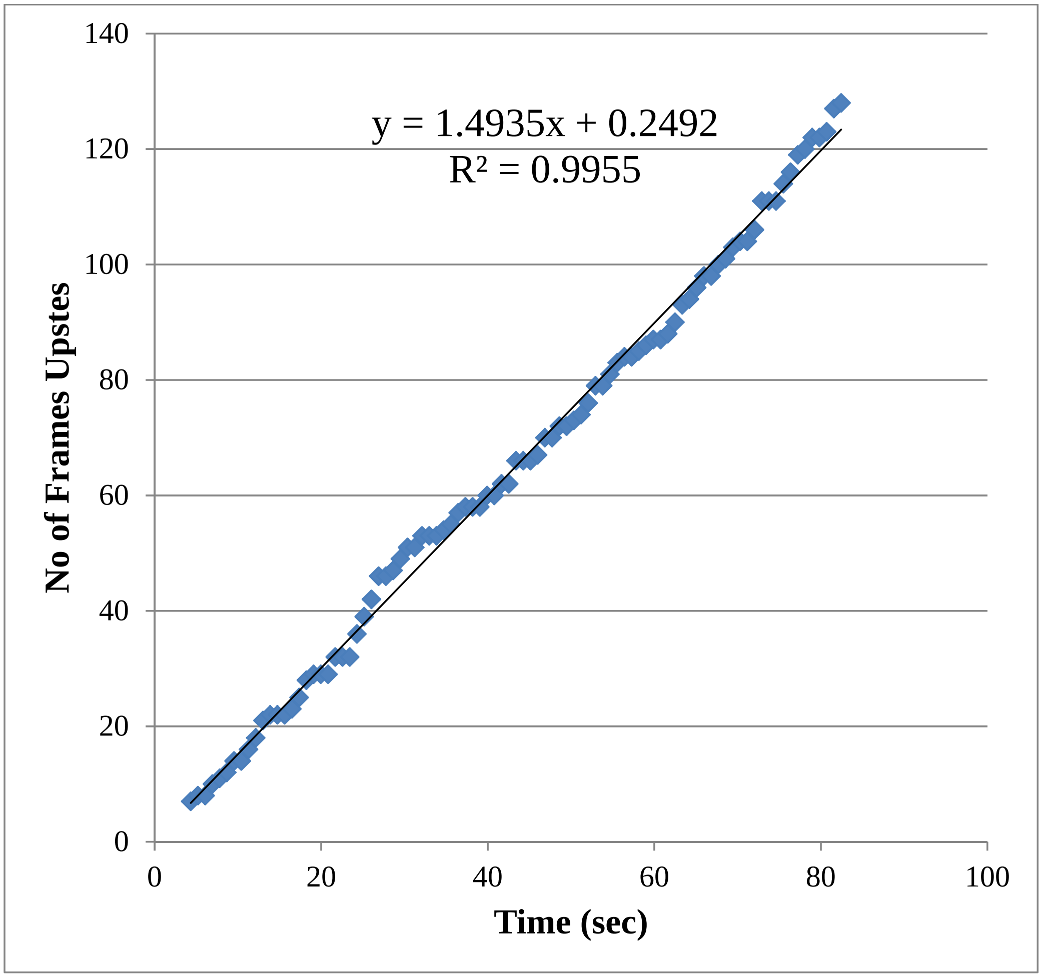


Figure 7-8. Static Cross Section Calculations at Flux Level of 3e06 ($cm^{-2}.s^{-1}$) – Energy Level of 65 MeV.

Table 7-6. Static Cross Section Calculations at Flux Level of 3e06 ($cm^{-2}.s^{-1}$) – Energy Level of 65 MeV.

Flux level ($cm^{-2}.s^{-1}$)	3e06
Exposure time (s)	82.4448
No. of frame errors (events)	128
Device size (bits)	8525025.826
Static event cross section	6.07e-14

The measured static cross section value gives an indication about the device sensitivity to the charged particles radiation. The sensitivity depends on the used manufacturing technology such as epitaxial CMOS, Silicon-on-Insulator (SOI) or Triple Oxide (TO). The static cross section is further used in estimating the soft error reliability as indicated in the chart in Figure 7-9.

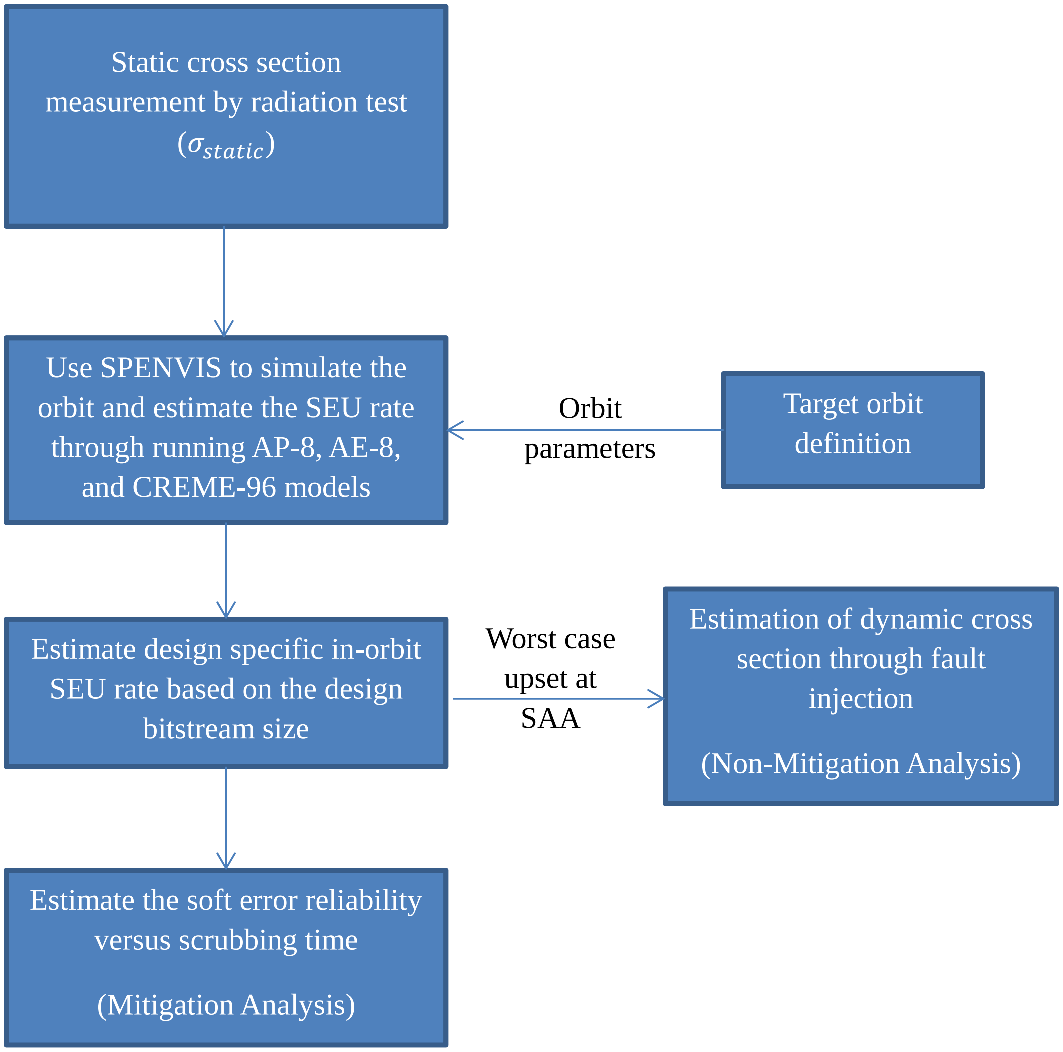

Static cross section measurement by radiation test (σ_{static})
Use SPENVIS to simulate the orbit and estimate the SEU rate through running AP-8, AE-8, and CREME-96 models
Orbit parameters
Target orbit definition
Estimate design specific in-orbit SEU rate based on the design bitstream size
Worst case upset at SAA
Estimation of dynamic cross section through fault injection
(Non-Mitigation Analysis)
Estimate the soft error reliability versus scrubbing time
(Mitigation Analysis)

Figure 7-9. Radiation Analysis Chart.

The static cross section σ_{static} is used by the models which run under SPENVIS to estimate the SEU rate based on the target orbit parameters. The AP-8 and AE-8 models are used to estimate the trapped protons and electrons energy spectra versus flux levels. Their output is used by the Cosmic Ray Effects on Micro-Electronics (CREME-96) model. The CREME-96 uses the rectangular parallelepiped method to estimate the accumulated charge and compare it with the critical charge at which the upset happens. The output of the CREME-96 is the SEU rate profile at the specified orbit. The design specific SEU rate is estimated based on the design bitstream size. The data is used to analyze the mitigated and non-mitigated cases. The mitigated case uses the configuration frames scrubbing method to wash out the configuration memory from SEU induced upsets. The soft error reliability is then calculated and estimated as a function in the time between scans. The non-mitigated case makes use of the fault injection concept to estimate the system dynamic cross section. The fault injection takes place at an upset rate which matches the worst upsets at the SAA region.

7.3 Fault Injection versus Radiation Tests

Fault injection is used in dynamic cross section estimation. It cannot be used in static cross section estimation. Radiation test can be used for both purposes. However, when using fault injection and/or radiation test, some points should be considered:

1) Fault injection implementation almost would need additional hardware and software to be able to inject the faults.
2) It might require stopping the operation of the design to inject the faults. However, if injection is on-the-fly as in the dynamic partial reconfiguration technique by Xilinx then there would be no need to stop the design operation.
3) Fault injection using SEU controller IP core covers injection in the FPGA configuration memory but what about the bits in the control and data registers of the design such as the processor registers?. Our main goal when we used the SEU controller IP core was to evaluate the fault injection effect in the configuration frames as they carry the hardware design. On the other side, system level faults such as register upsets are well studied and can be mitigated using variety of techniques such as TMR, EDAC, and SIHFT. We performed TMR effectiveness analysis using simulations as will be shown later. The system can still be recovered from SEUs in case of its complete failure using WDTs to reset it.
4) In the case of fault injection there is a non-simultaneous fault injection situation where faults are injected in the FPGA configuration on a one-by-one basis. Fault injection using SEU controller cannot inject faults in multiple frames simultaneously as in the real radiation case.
5) In radiation test we usually conduct the test at a specific energy level of the charged particles. However, in space, charged particles exist at all energies. Therefore, an extrapolation of the performance during the test should be considered to be able to plot the system failure rate versus the different upset rates. In radiation test we usually control the exposure time and flux as they are easier than changing the energy level. In fault injection test, we can easily test at

different upset rates as we just need to change the amount of injected faults per unit time.

6) In radiation test, if it was not at a particle accelerator, we need to perform de-capsulation. For example, Cf-252 test needs to de-capsulate the top cover of the package to get SEU events as shown in Figure 7-10.

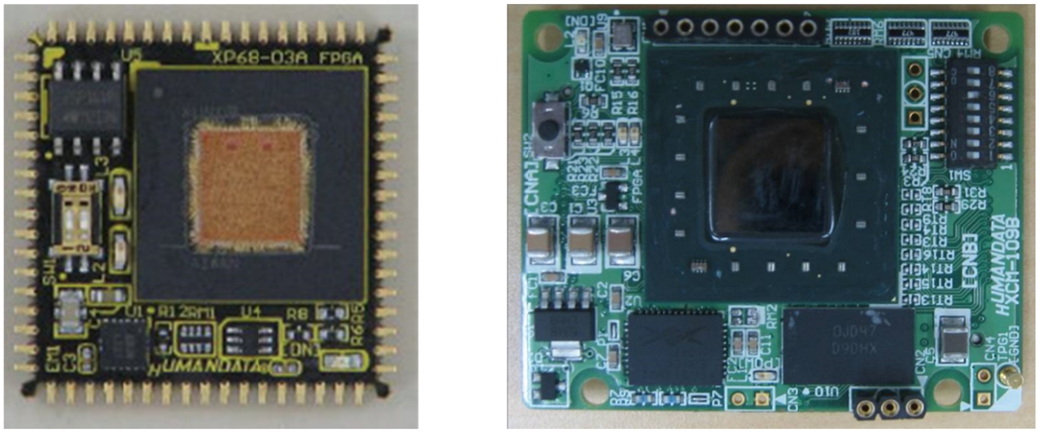


Figure 7-10. de-capsulated Spartan-6 and Virtex-5 FPGAs for Cf-252 radiation test.

7.4 Dynamic Cross Section through Fault Injection Test

Figure 7-11, shows the results of running the fault injection campaign on the MPSoC with two fault lists sizes: 1) 50 accumulated faults, and 2) 100 accumulated faults. Each fault list was generated randomly for 10 times and the results were collected for the accumulation of the faults at each time. In the MPSoC design there are three types of correction that can take place: 1) Full FPGA reconfiguration, 2) Partial reconfiguration and 3) Software resynchronization. The full reconfiguration takes place when the FPGA stops working due to fault injections. The entire bit-stream of the design should be reloaded to the internal SRAM in order to restore the correct operation. The partial reconfiguration takes place when one or two processor stops working but not the entire three processors. The system can be partially reconfigured without stopping the working processor(s) to restore the correct operation. Software resynchronization takes place when the software of the working processors need to be resynchronized to the same operation after one or two processors stopped working and then resumed again.

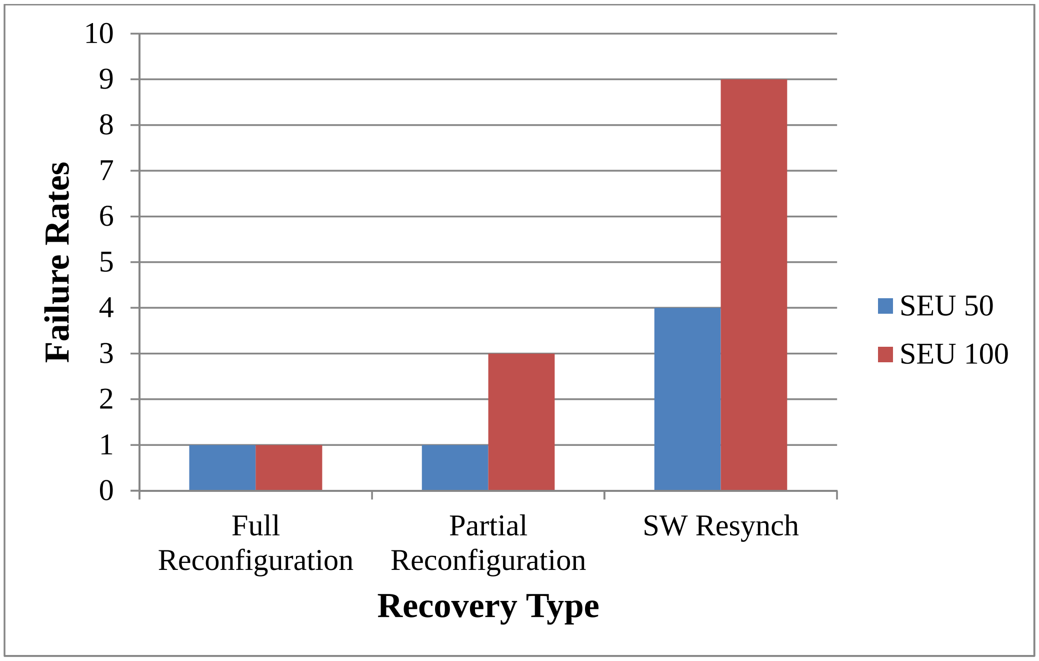


Figure 7-11. Fault Injection Results.

The results show that about 10% of the injected faults in the 50 faults batch and 10% of the 100 faults batch needed full reconfiguration. Another 10% of the faults in the 50 faults batch needed partial reconfiguration while 30% of the faults injected in the 100 faults batch needed partial reconfiguration. This means that in the 50 faults batch, only 80% of the injected faults where totally recovered through the ACM of the SEU controller without the need for partial or full reconfiguration. In the case of the 100 faults batch, 60% of the injected faults were fully recovered with no need of any reconfiguration. The software resynchronization takes place whenever a partial reconfiguration is initiated or a processor stops operation then resumes after the auto-correction mode has been enabled. The obtained results of fault injection coincides with the concepts mentioned by Xilinx that about 10% or less of the faults would lead to total system failure. This result can be viewed as a validation of the fault injector operation. Figure 7-12, verifies the fault injector function as it shows the plot of 50 injected faults versus the times between injections in seconds. The times between injections follow an exponential distribution. The number of faults themselves follows Poisson distribution.

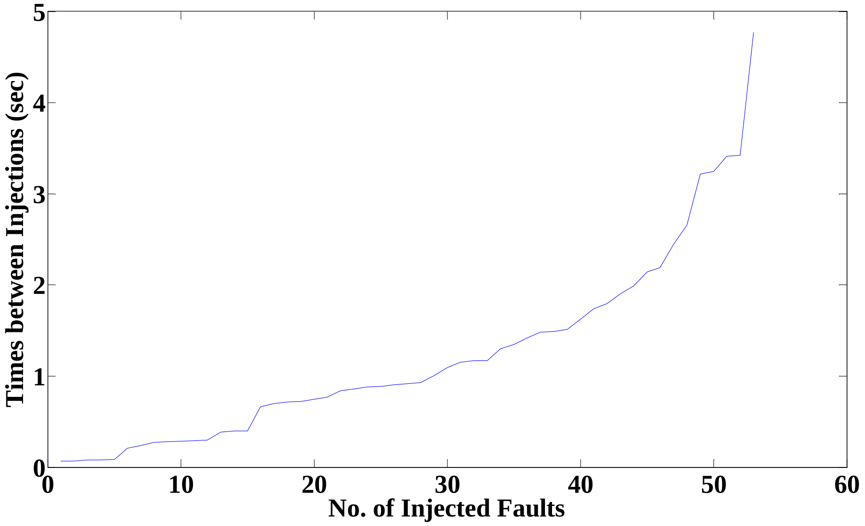


Figure 7-12. Time Distribution between Injected Faults.

The TMR system was simulated at different data vector sizes. We use Eq. **(7-1)** deduced in [85] [86], to evaluate the TMR simulator behavior.

$$R = \frac{1}{T_c} - \frac{1}{T_c}\prod_{i=1}^{M}[3e^{-2N_i r T_c} - 2e^{-3N_i r T_c}]$$

(7-1)

Where (R) is the TMR system failure rate, (T_c) is the cycle operation time during which faults are monitored also called the scrubbing time and we chose it as 1 sec, (N) is the number of bits in each TMR data vector and we set as 8 bits, (M) is the number of TMR groups for example we set it at 2048. Figure 7-13, shows the results of applying the fault injection over a packet size of 2048 bytes in a TMR operation. The packet contained a random vector of data and the vector is compared between three of the operating cores after faults were injected randomly in it. The vectors are compared value by value in a TMR operation through a voter. The upper curve (green) shows the failure rate as estimated by Eq. **(7-1)**, while the lower curve (red) shows the simulation results.

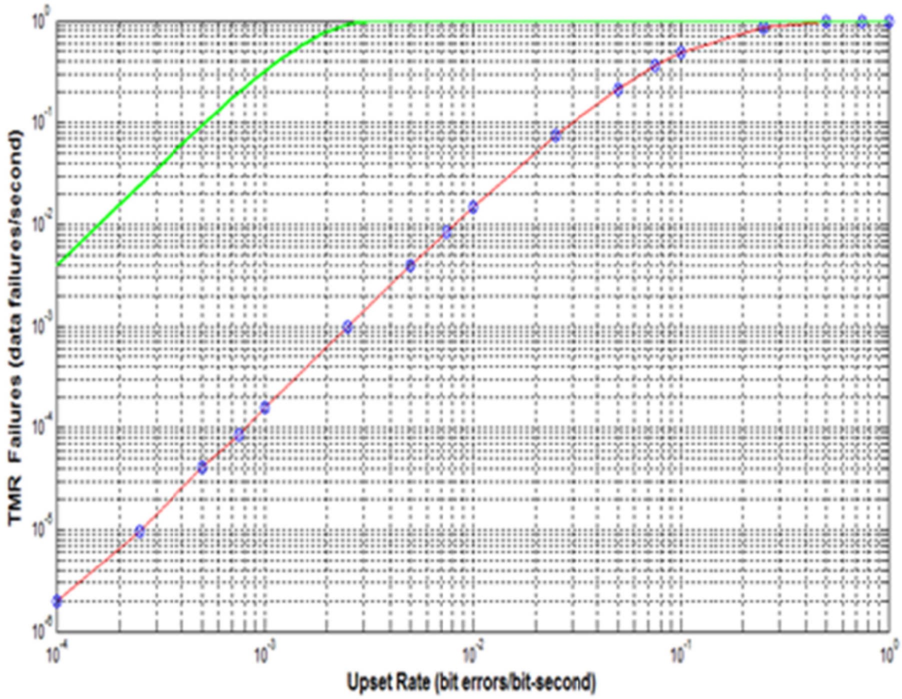


Figure 7-13. TMR Failure Rate at Data Vector of Size 2048 bytes.

From the results we conclude that fault injection can be used for injecting random faults in the FPGA bit-stream to simulate the effects of the space environment. About 10% of the injected faults in the hardware bit-stream needed full reconfiguration. In case of data fault injection at an upset rate of 0.1 upsets per bits per second, more than 50% of the data had residual failures.

To estimate the dynamic cross section we use Eq. (7-2) :

$$Dynamic\ Cross\ Section = \frac{\#\ of\ system\ failures}{\#\ of\ bit\ upsets} \tag{7-2}$$

We estimate the dynamic cross section in a pessimistic way. We consider that the system failure at any trial means the failure at all of the bits in that trial. Therefore, if the system fails when injecting 50 accumulated faults we would consider that this corresponds to 50 failures. The reason for that assumption is that we cannot identify the accurate number of bits after which the system stopped within the 50 injections. The system might have stopped after 10 or 20 or 40 errors were injected. For simplicity and as a worst case we consider that the total failures correspond to the trial size. By experimenting we found that 1 out of 10 trials had total failure in the 50 and 100 bits experiments. This corresponds to a dynamic cross section of (50/500) or (100/1000) which is 0.1 failures per bits upsets. So if there are 1000 upsets the system would fail by 100 times which is too much pessimistic consideration. To calculate the dynamic cross section per device we would divide by the number of bits in the device (0.1/11.37e6) = 8.79507e-09 bit^{-1}. As the injection elapsed about 60 sec for each trial. The Dynamic cross section per bit per sec can be calculated as = 8.79507e-09 bit^{-1} /60 = 1.46585e-10 sec^{-1} bit^{-1}. This value of the dynamic cross section is in the case of a fully configured FPGA. The expected failure rate depends on the actual design size. Therefore, the MicroBlaze system failure rate would be (0.26 ×11.37e06 × 1.46585e-10 = 4.33e-04 sec^{-1}). The simulation of the system failure rate in SPENVIS showed that each MicroBlaze system failure rate would be 0.00183517 sec^{-1} (at non-SAA)/3 = 6.117233e-04 sec^{-1}. The results show that SPENVIS calculations overestimated the failure rate by 30%. The actual in-orbit SEU rate as reported by a Virtex-4 flight experience was found to be four to five times less than the rate estimated by the CREME96 model which is used in SPENVIS [36]. This means that our estimation of the failure rate based on the fault injection approximation, complete 50 failures or 100 failures whenever a trial fails, has enough calculation margins.

7.5 Thermal Vacuum Test Results

Thermal vacuum test was repeated after the thermal vacuum screening test showed that the FPGA temperature reached about 105 ℃. In the repeated test we used the combination of the thermal control techniques mentioned in chapter 6. We thought of these techniques as a method that might reduce the FPGA core temperature through controlling the thermal power. At first we had to assure the auto protection feature of the Xilinx Virtex-5 FPGA. It is well designed to shut down when its core temperature exceeds 125 ˚C. It keeps off, until the silicon cools down and then it restarts again as shown in Figure 7-14. The FPGA cools to less than 114 ˚C. However, for reliability and long lifetime we should keep the FPGA around 85 ˚C as recommended by Xilinx for the commercial grade Virtex-5. After the auto protection test we performed a durability test at the 105℃ which was recorded during the initial temperature profile screening. The durability test, result shown in Figure 7-15, was a form of stress testing to make sure that the FPGA can withstand the over limit temperature stress even if it was not mitigated.

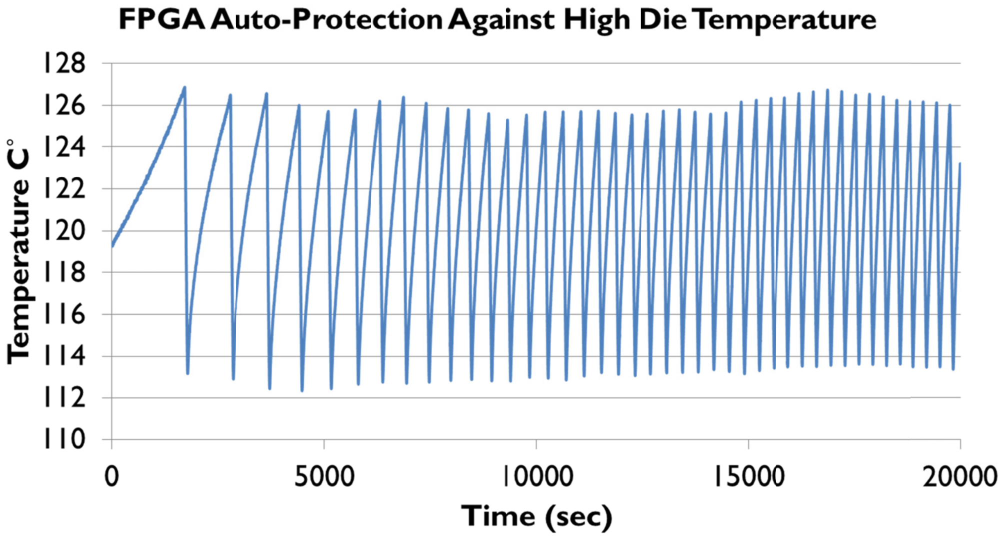


Figure 7-14. FPGA Automatic Thermal Protection.

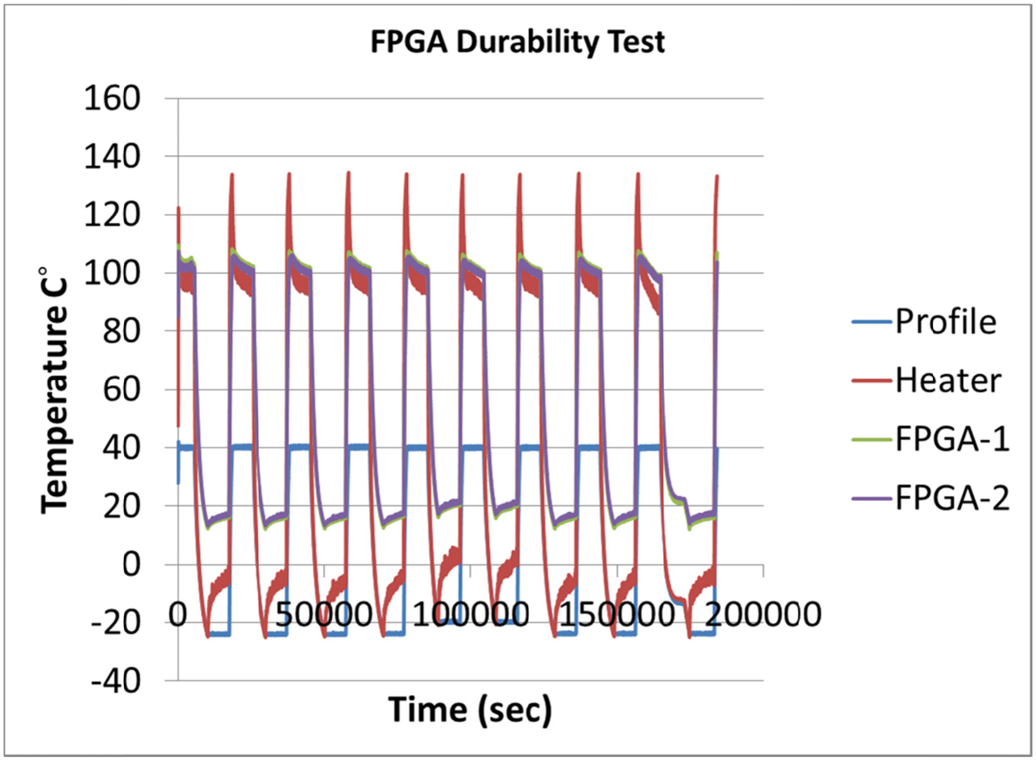


Figure 7-15. FPGA Durability Test.

Figure 7-16 shows the worst case temperature record from inside the FPGA die using the built thermal diode connected to an external temperature measurement IC as explained in chapter 6.

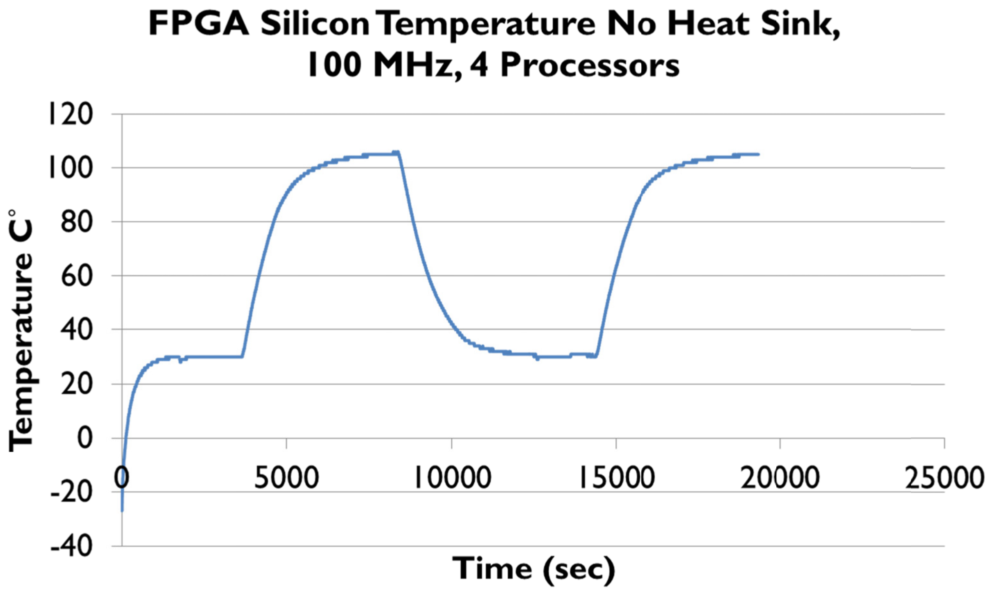


Figure 7-16. Thermal Vacuum Test, No-Heat Sink, 100 MHz, 4 Processors – FPGA Internal Die Temperature.

The external temperature sensors readings are shown in Figure 7-17, theses sensors are the thermocouples previously shown in Figure 6-10 and Figure 6-11.

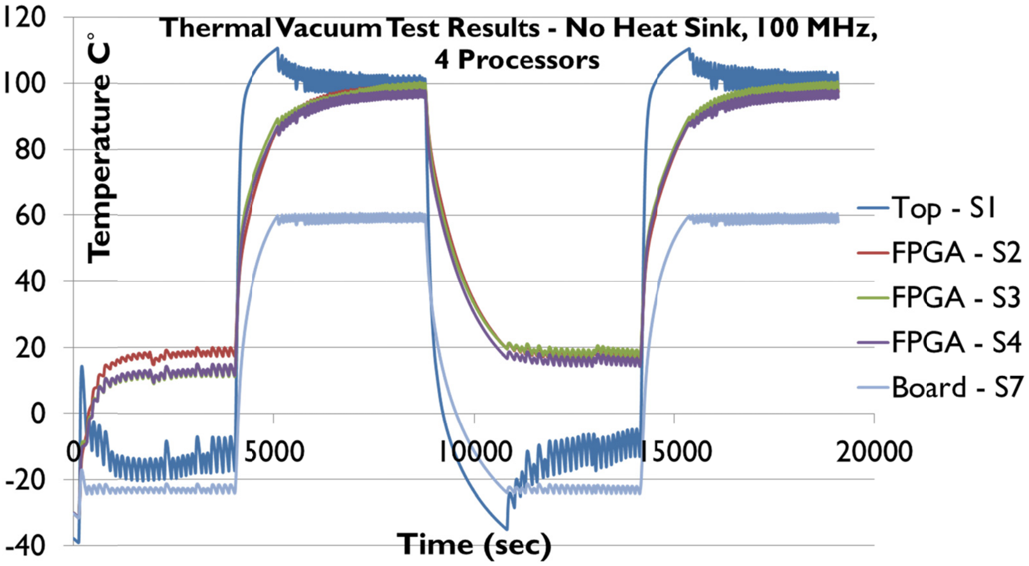


Figure 7-17. Thermal Vacuum Test, No-Heat Sink, 100 MHz, 4 Processors – Thermocouples.

Figure 7-18 shows the best case temperature record when applying all of the mitigation techgniques both passive and active. The measurement is from isnide the FPGA die.

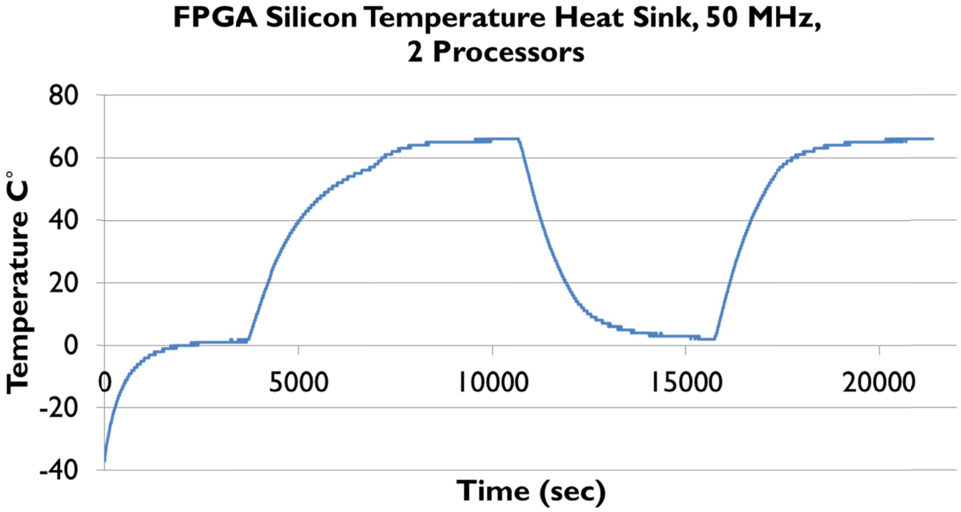


Figure 7-18. Thermal Vacuum Test, Heat Sink, 50 MHz, 2 Processors – FPGA Internal Die Temperature.

The external temperature sensors readings, external thermocouples, are shown in Figure 7-19. These are the thermocouples attached in Figure 6-11.and Figure 6-12.

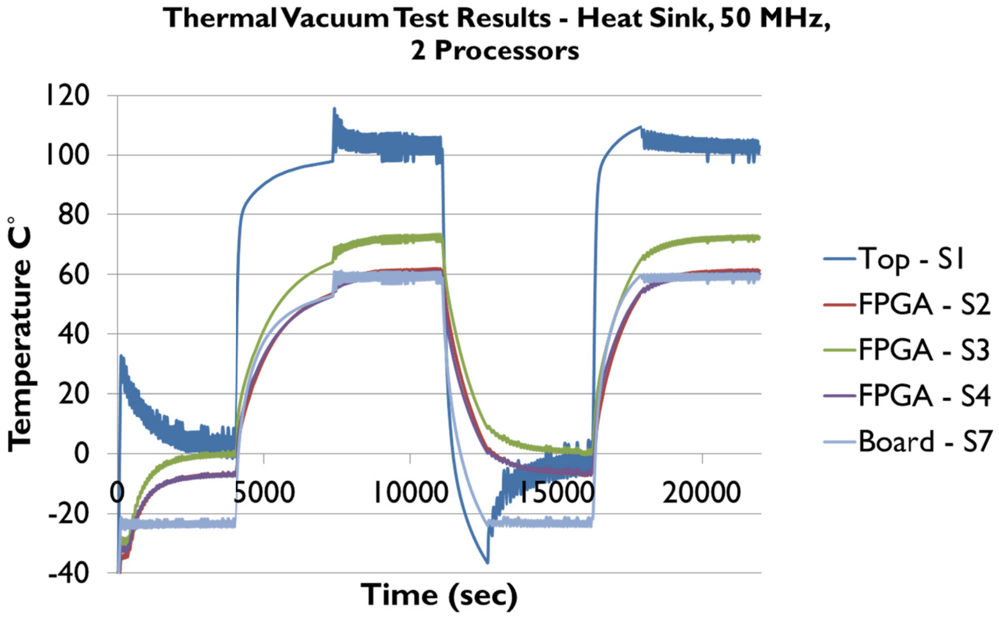
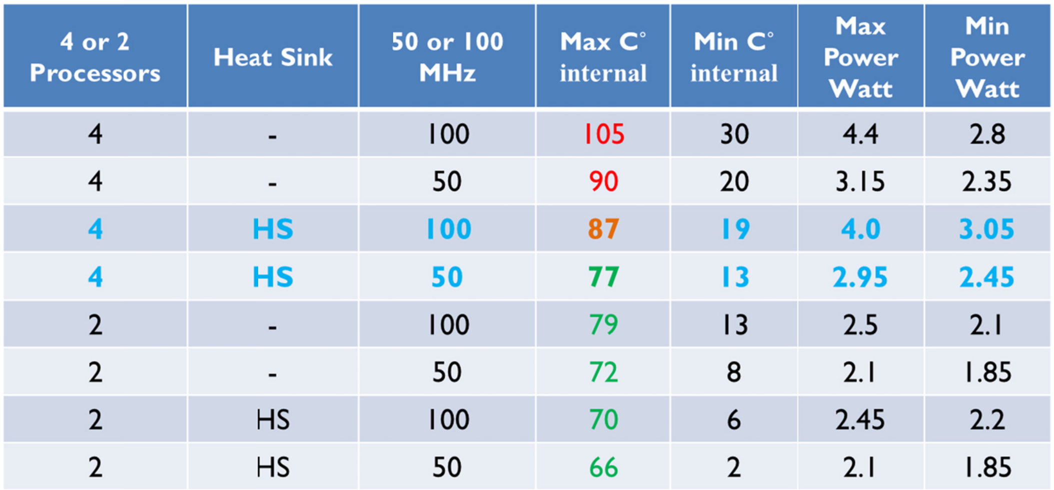


Figure 7-19. Thermal Vacuum Test, Heat Sink, 50 MHz, 2 Processors – Thermocouples.

Table 7-7. Thermal Vacuum Test Results with and without Mitigation Techniques



4 or 2 Processors	Heat Sink	50 or 100 MHz	Max C° internal	Min C° internal	Max Power Watt	Min Power Watt
4	-	100	105	30	4.4	2.8
4	-	50	90	20	3.15	2.35
4	**HS**	**100**	**87**	**19**	**4.0**	**3.05**
4	**HS**	**50**	**77**	**13**	**2.95**	**2.45**
2	-	100	79	13	2.5	2.1
2	-	50	72	8	2.1	1.85
2	HS	100	70	6	2.45	2.2
2	HS	50	66	2	2.1	1.85

Table 7-7 summarizes the mitigation techniques test results. The system without any mitigation technique would exceed the operating limits up to 105℃. It is clear that using heat sink is an effective passive solution to reduce the temperature and hence the static power. Reducing the operating frequency to reduce the dynamic power is an effective approach either alone or when combined with another mitigation technique. Using reduced number of processor systems would also reduce the number of nodes need to be toggled thus reducing the dynamic power. The maximum reduction in the operating temperature is achieved when all techniques are combined together thus reaching 66℃ which corresponds to -37% reduction from the maximum temperature without any mitigation. Figure 7-20 shows the effectiveness of the techniques when applied alone or combined together through the percentage of temperature reduction in reference to the maximum temperature in the non-mitigated case.

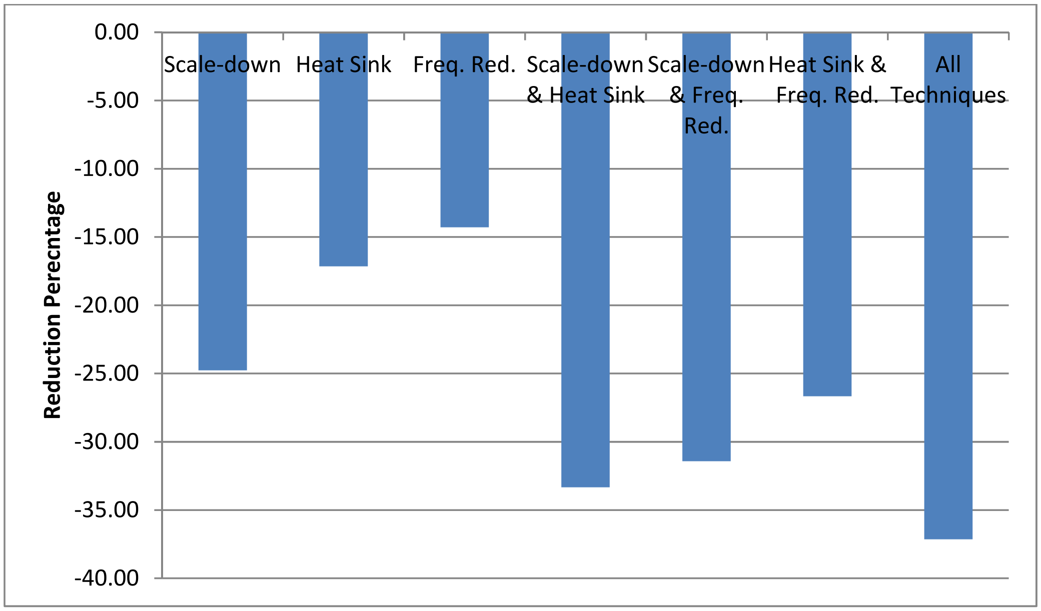


Figure 7-20. Thermal Mitigation Effectiveness.

In this chapter we presented the detailed results and discussion of the MPSoC OBC performance evaluation, time and space redundancy approaches effectiveness comparison, static event cross section estimation through proton beam radiation test, dynamic cross section estimation through fault injection and thermal vacuum mitigation test results. In the next chapter we present the conclusion and future work.

Chapter 8 : Conclusion and Future Perspectives

The thesis presented the design and qualification of an MPSoC-OBC that can be used in future space missions. The design used an SRAM-based FPGA in the implementation which is considered a state-of-art in small spacecraft technology by the time of writing the thesis as in [14]. In this chapter we cover the conclusive outcome of the thesis as well as future research perspectives.

8.1 Conclusion

In future space missions a lot of data would be collected through the on-board scientific instruments. This data would be analyzed and processed on-board rather than sending it to the ground. This technique of processing the data on-board the satellite means that the results are sent to the ground station which is known as in-orbit processing. As scientific satellites mostly fly in LEO, the communication session time with them is limited to 10 min at maximum and for 3 to 4 times per day. To download large volumes of scientific data and satellite telemetry, we need to have high bandwidth for higher bitrates and high power for correct transmission. This means we have to develop satellites with larger sizes and masses to suffice for the required communication system requirements. Building larger satellites means incurring higher recurring and non-recurring engineering costs which might not be afforded by scientific institutes. Therefore we thought of developing high performance processing computers that would process the data locally on-board the satellite without the need to download the raw data to the ground station. This on-board processing would reduce the required communication system characteristics and hence the satellite size, mass and cost. Also using COTS components reduces the cost drastically when compared to space approved components.

The research tried to provide the needed high processing capability platform at an affordable cost and reliability by making use of the state of art COTS technology. The deliverable was defined to be a low cost high performance MPSoC OBC that can be used in scientific satellites as well as commercial ones. The MPSoC OBC had to provide sufficient reliability suitable for small satellites missions as well as fitting within the tight power budget limits. As low cost does not mean that the system should be of low reliability, we had to propose mitigation techniques in the design and perform exhaustive testing on them to make sure that the system would work flawlessly in the space environment.

The design was performed using the Xilinx EDK tool. A total utilization of 98% of the FPGA resources was recorded. That utilization is occupied by 4 complete embedded systems. If there is no need to expand the design after deploying it by adding extra logic then the higher the utilization would be the better as long as the power constraints are respected.

Performance test was conducted to judge the performance of 65 nm FPGAs in the case of time and space redundancies. The execution time of the bubble sort algorithm on randomly generated data was almost three times larger in the case of time redundancy compared to space redundancy. Space redundancy execution time was slightly higher than the execution time of non-fault tolerant system. If more complex data communication protocol was used among the processors while exchanging their data for voting then the communication overhead would be higher. That would lead to significantly increasing the space redundancy execution time compared to the non-fault tolerant case. Therefore, we should be careful in choosing simple yet reliable data communication protocols between the processors in order to reduce the communication time overhead.

Reliability of the system was studied both in the case of time and space redundancies. It was found that the time between complete scans (scrubs) of the configuration memory is significant in determining the immunity of the configuration bitstream to the induced soft errors. The lower the time between the complete scans the faster the detection and hence correction of errors. Continuous scanning using the SEU controller was proven to assure a reliability of 0.999999. It should be noted that in case of using the TMR regime then the time between scans should not exceed a certain limit otherwise the reliability of the single core system would be higher than the TMR system. In TMR systems that limit is usually the 7 tenth (0.69) of the operating time.

Power consumption was noticed to be higher in the case of the triple core space redundancy when compared to the single core time redundancy. The dynamic power consumption was increased due to the extra logic used in the space redundancy by about 30% compared to the time redundancy case. By evaluating the four parameters of utilization, performance, power consumption and reliability, the space redundancy proved to be the best choice for 65 nm SRAM-based Virtex-5 FPGA. That result can be generalized to other FPGAs of the same technology feature size. It should be noted that the lower the technology feature size the higher the static power consumption and SEU induced soft errors would be.

Radiation testing was performed to judge the static and dynamic cross sections. The static cross section was evaluated using a new method. Conventionally two systems are used to evaluate the static cross section: 1) The DUT, and 2) Monitoring and control system. The monitoring and control system is responsible for controlling the test operation as well as scrubbing the FPGA for induced errors due to irradiation by the proton or heavy ion beam. In our case we used only one device, the DUT. We replaced the functions of the external monitoring and control system with the internal SEU controller. It performed the scrubbing function and reported the errors found through the UART interface to an external monitoring computer. We calculated the frame errors during exposure to radiation at 4 trials. The frame errors enabled us to calculate the event bit cross section which was found to be 6.34e-14 cm^2/bit and NASA's value is 6.37e-14 cm^2/bit at a proton beam energy level of 65 MeV.

The dynamic cross section was evaluated through fault injection. The performance of the system was monitored while injecting randomly generated 50 bit flips in the configuration frames for 10 times. The injection was repeated with randomly generated 100 bit flips for 10 times as well. The MicroBlaze system failure rate was estimated to be 4.33e-04 sec^{-1} using fault injection while it is estimated to be 6.117233e-04 sec^{-1} using SPENVIS at non-SAA. SPENVIS shows 30% higher estimation. The calculations of the static and dynamic cross sections are necessary to characterize the device and hence expect its performance under different radiation conditions.

Thermal vacuum testing was conducted to estimate the commercial grade FPGA ability to operate flawlessly within the expected in-orbit temperature ranges. The SMC standard, Air Force Space Command/Space and Missile Systems Center Standard - Test Requirements for Launch, Upper-Stage and Space Vehicles, was chosen to define the test limits.

Thermal vacuum testing was conducted in between -24 ˚C and + 60 ˚C. The FPGA temperature reached about 105 ˚C which is outside the recommended upper limit of 85 ˚C for the commercial grade. The FPGA was tested to check its auto-protection mechanism to switch off itself if the temperature reaches 125 ˚C. The mechanism worked well and then several mitigation techniques were proposed to put the operating temperature close to or less than the 85 ˚C limit. Heat sink, decreasing the operating frequency, and decreasing number of processors in the FPGA were tested. The temperature was proven to reach 66 ˚C when applying all of three techniques together. We recommend to use the black anodized heat sink with high emissivity (>0.8) at 50 MHz to keep the system temperature at about 77 ˚C. We should carefully consider that power consumption varies with the operating temperature. Static power consumption increases when temperature increases or core voltage changes. Dynamic power consumption varies with the change in the number of nodes being switched or switching frequency or core voltage. Reducing the operating temperature reduces the consumed power and reducing the power consumption reduces the total thermal power. This is a recursive relation that is used to achieve the required operating temperature and power limits.

This research is crucially important to the satellite development community as it would enable the satellite developers to build satellites with higher processing capabilities at much lower costs. Using COTS in space applications is an on-going state-of-art research that would open the horizon for developing new test techniques, algorithms and architectures for fault tolerant operation in the harsh space environment. Even in ground-based applications, neutrons induced SEUs were recorded. Therefore, mitigation against such effects is mandatory for safety critical applications and high quality services. The research also focused on the importance of providing effective thermal vacuum mitigation techniques for the FPGA. The results help in providing the best route to utilize the correct combination of mitigation techniques against soft errors and thermal vacuum conditions. These results can be applied to terrestrial based applications as well as space based applications.

The major contributions of this research can be summarized as : 1) The use of FPGA technology to develop a new reprogrammable design for low cost, high performance, moderate power and high reliability OBCs, 2) Formalization of the procedure to analyze in-orbit radiation effects and the efficiency of scrubbing to mitigate them, 3) New test method for static cross section calculation at accelerated proton radiation test, and 4) proposing effective thermal vacuum mitigation techniques for protecting commercial grade FPGAs while in-orbit as well as pinpointing the effect of the different techniques on power consumption.

8.2 Future Perspectives

In this research we introduced methods that can be generalized on different types of SRAM-based FPGAs with different technology feature sizes. Nowadays 28 nm FPGAs are available. It would be good to extend the research results on other types of SRAM-based FPGAs which provide higher performances with low power consumption and high reliability.

The essence of this research was to enable the commercial grade SRAM-based FPGAs to be used in serious space applications. The research should continue in the stream of enabling COTS SRAM-based FPGAs for variety of space applications. That would give higher access to universities, research institutes and developing countries to be actively participant in the dynamic and on-going progresses in space exploration.

Enabling modern FPGAs with feature sizes less than 65 nm to be used in space depends on two pillars: 1) developing new mitigation techniques, and 2) developing new test standards and procedures. As the feature size is reduced more challenges are introduced. Radiation effects, mainly SEUs, would be highly manifested as more soft errors are likely to be induced. This is due to the fact that the decrease in the node capacitances corresponds to decrease in the node critical charges needed to induce an upset. It is likely, then, to expect MBUs in the same configuration frame. Therefore, the current SECDED technique would not be suitable anymore. Dynamic partial reconfiguration through reading the full correct frame from an external golden flash memory and writing it back to the FPGA would solve the MBU problem.

As the technology feature decreases the device static power consumption increases as more current is likely to leak. Xilinx introduced the triple oxide concept to reduce that problem. However, on the system designer side, active thermal control might be needed to handle the increase in temperature due to the increase in static power consumption. Traditional passive thermal control might not be sufficient for achieving the required temperature operation limits.

Some tests are necessary to conduct for small feature sized FPGAs such as Total Ionization Dose and Displacement Damage Dose. X-ray scan before and after the thermal vacuum test might also be useful to study whether there are cracks in the Ball Grid Area (BGA) solder pads or not. Conducting thermal vacuum tests in the presence of infra-red camera would be useful to detect any hot-spots in the FPGA and/or PCB. Dynamic cross section evaluation through proton beam irradiation at fluxes close to the in-orbit case would be useful to assure the fault injection results. However, it would definitely need long exposure time which would increase the test costs.

Test procedures and setup are getting increasingly complex as more connections might be needed during the test. Fixation of the DUT becomes difficult as well. All of this introduces real challenges in performing correct testing which might give meaningful results [22]. Therefore, it would be useful to develop specific test procedures, communication protocols, test platforms, measurement and analysis software packages and even mechanical fixations with the concept of reusability in mind. That would reduce test costs and lead to faster and more accurate testing. It is always better to use a test device which we are sure of its accuracy than to use a new device for the first time during testing. Standards should be modified, or even a new standard should be developed, to fit with the nature of the small satellite missions. Accepting to accommodate more risk in small satellite means that test procedures would became less complex and less demanding compared to the currently available professional standards. The Nano-satellite Environment Test Standardization (NETS) project at Kyushu Institute of Technology is an example for the endeavors which are currently undertaken to develop more fitting standards [87].

A complete Satellite-on-Chip would be feasible to realize by now [88]. This idea is a fairly challenging task to implement the hardware and software of the different subsystems in a single FPGA package. However, implementing such a single chip satellite would be a revolutionary breakthrough in the satellite technology. As we were able to accommodate 4 complete embedded systems inside a single FPGA package then it would be feasible to assign the processors specific tasks to execute as if they represent the satellite subsystems' processors. Sophisticated failure management procedures as well as task distributions, dynamic resources allocation, interfaces and perhaps a larger size FPGA to accommodate specific digital design for communication modems are needed. The Satellite-on-Chip would mainly execute the digital signal processing as well as the algorithms of the different satellite subsystems. However, careful consideration for the problem of redundancy should be given.

Redundancy either in the form of cold backup or hot backup is a well-known technique in satellite engineering which is used to increase the system reliability. The use of redundancy adds a problem in terms of the mass, volume, and/or power consumption. If a cold backup subsystem is used then it would need its own volume in the satellite and it would add to the total mass. If hot backup redundancy is used then it would add to the consumed power, mass and volume. Nano-satellites are a domain to get rid of redundancy, taking the risk accompanying that decision. They do not have enough mass, power, and volume budgets to accommodate extra units.

By introducing miniaturized designs in modern FPGAs, redundancy can migrate from the outside to the inside. Thus it can move from being large bulk devices into small logic gates implemented inside a single FPGA. This means that redundant satellite-on-chips can be built on the same FPGA package to increase the overall system reliability. Moreover, redundancy can be selective. Modern SRAM-based FPGAs provide the facility to implement Dynamic Partial Reconfiguration of their bitstreams. This means that they can change some modules in design on-the-fly without stopping operation of the main static logic. Reconfigurable partitions are defined to carry reconfigurable modules. Whenever a redundancy is needed, its reconfigurable module is called and loaded in the reconfigurable partition to selectively provide high reliability depending on the operation conditions. If the redundancy is not needed, its reconfigurable partition is freed by loading a black box reconfigurable module. In this method power can be saved as well as area, mass, and volume as we call the function whenever needed in contrast to the previously permanent redundancy. We call that new concept dynamic redundancy.

One direct application of dynamic redundancy is to load TMR modules at the SAA region of the orbit while unloading them in the rest of the orbit. That loading and unloading could save power up to 60% in comparison to the permanent TMR system [58]. This means that hardware designs are no longer static. The design is dynamic and can be changed in-orbit either remotely by uploading new hardware or locally by loading hardware files from the memory. Also it means that the hardware became scalable in such a way that it can grow up to contain multiple modules or shrink down to the basic limits.

As the processing power of computational platforms is increasing tremendously, it would be a matter of time before we see a fully smart system which could cure itself. The concepts of evolution and adaptation are very important in the human lives. Biological inspirations in the field of fault tolerances and artificial intelligence are well established. However, there is the concept of evolvable hardware. That is the hardware which can evolve to adapt to changes in its environment in order not to die out. That adaptation might be through partially or fully changing its design. The change however need to be initiated based on well-established semantic to define what is good and what is bad for the system. Obviously, it would be some sort of an advanced silicon brain that would cross the limits of the current autonomous systems which can perform smart failure recovery and in-orbit reconfigurations in the absence of ground control. For example, IBM is building neurosynaptic chips at ultra-low power consumption. Being inspired with the human brain, IBM develops brain-inspired computer architecture chips. The recent chip built by IBM, by the time of writing this thesis, contains 1 million neuron and 256 million synapses with 5.4 billion transistors. It contains an on-chip network of 4096 neurosynaptic cores at 70 mWatt. This means that supercomputing and cognitive systems would ultimately be migrated to space borne devices, it is just a matter of time.

The difference between the current traditional and future evolutionary hardware designs is that the current systems are fully programmed with possible scenarios prior to their in-field operation. Even if they have the chance to implement some sort of smartness then it would be based on a predefined heuristic by the developers. While the evolutionary systems just have the starting point from which they start to develop just like human beings and then they can take decisions out of their own even without the discretion of their own developers. The merge between powerful design tools, computational platforms and advances in artificial intelligence would make satellites much smarter than we expect. Reaching that point might enforce the developers to negotiate with their evolving satellite-organisms for the best sake of both sides. Satellites might be able to decide what the best for their operation in a better way than the developers themselves.

Finally, we would like to stress on the fact that developing nano-satellites is an intrinsically challenging task as they do incur risks due to their design philosophy. Reiterating the previously accomplished processes is likely to occur during the development and qualification phases. More focus should be given to study the effects of those iterations on the schedule, cost and reliability of the nano-satellite projects. Appendix (G) introduces our study on that issue.

The future is out there and this thesis tried to push forward towards it. State of the art technology will always emerge with new products and outcomes. One way to push forward is to try to present the capable platforms to developers to realize their ideas and make them feasible. We tried to push the capabilities of the current satellites forward. There is still a lot to do and to achieve however it should always be considered that the value of the work is better appreciated whenever it touches real problems and open new doors for imagination and innovation.

References

[1] A. Siddiqi, "Sputnik 50 years later: New evidence on its origins," *Acta Astronautica,* vol. 63, no. 1-4, p. 529 – 539, July–August 2008.

[2] L. Summerer, "Thinking tomorrows' space – Research trends of the ESA advanced concepts team 2002–2012," *Acta Astronautica,* vol. 95, no. 0, p. 242 – 259, February–March 2014.

[3] E. E. Weeks and A. A. Faiyetole, "Science, technology and imaginable social and behavioral impacts as outer space develops," *Acta Astronautica,* vol. 95, no. 0, p. 166 – 173, February–March 2014.

[4] R. D. Launius, "Space stations for the United States: An idea whose time has come—and gone?," *Acta Astronautica,* vol. 62, no. 10–11, p. 539 – 555, May–June 2008.

[5] T. Thumm, J. A. Robinson, C. Alleyne, P. Hasbrook, S. Mayo, N. Buckley, P. Johnson-Green, G. Karabadzhak, S. Kamigaichi, S. Umemura, I. V. Sorokin, M. Zell, E. Istasse, J. Sabbagh and S. Pignataro, "International space station accomplishments update: Scientific discovery, advancing future exploration, and benefits brought home to earth," *Acta Astronautica,* vol. 103, no. 0, p. 235 – 242, October–November 2014.

[6] B. Detsis and E. Detsis, "The benefits brought by space—General public versus space agencies perspectives," *Acta Astronautica,* vol. 88, no. 0, p. 129 – 137, July–August 2013.

[7] S. P. Neeck, T. J. Magner and G. E. Paules, "NASA's small satellite missions for Earth observation," *Acta Astronautica,* vol. 56, no. 1–2, p. 187 – 192, January 2005.

[8] G. W. Ousley, "NASA/International small satellite program," *Acta Astronautica,* vol. 53, no. 4–10, p. 771 – 777, August–November 2003.

[9] L. Alkalai, "Advanced microelectronics technologies for future small satellite systems," *Acta Astronautica,* vol. 46, no. 2–6, p. 233 – 239, January–March 2000.

[10] J. Esper, P. V. Panetta, D. M. Ryschkewitsch, D. W. Wiscombe and S. Neeck, "NASA-GSFC nano-satellite technology for Earth science missions," *Acta Astronautica,* vol. 46, no. 2–6, p. 287 – 296, January–March 2000.

[11] P. A. Curto and R. S. Hornstein, "Injection of new technology into space systems," *Acta Astronautica,* vol. 57, no. 2–8, p. 490 – 497, July–October 2005.

[12] M. Sweeting, "25 Years of space at SURREY— Pioneering modern microsatellites," *Acta*

Astronautica, vol. 49, no. 12, pp. 681 - 691, December 2001.

[13] D. G. Gilmore, B. E. Hardt, R. C. Prager, E. W. Grob and W. Ousley, Space Mission Engineering: The New SMAD, vol. 28, J. R. Wertz, . D. F. Everett and J. J. Puschell, Eds., Microcosm Press 2011, Space Technology Library, 2011.

[14] Mission Design Division Staff Ames Research Center, "Small Spacecraft Technology - State of the Art," NASA/TP–2014–216648, pp. 1 - 208, Feb. 2014.

[15] D. Hastings and H. Garret, Spacecraft Environment Intercations, Cambridge University Press, 1996.

[16] K. L. Bedingfield and R. D. Leach, Spacecraft System Failures and Anomalies Attributed to the Natural Space Environment, M. B. Alexander, Ed., NASA Reference Publication 1390, pp. 1 - 51, 1996.

[17] M. O'Bryan, K. A. Label, R. A. Reed, J. Barth, C. Seidleck, P. Marshall, C. Marshall and M. Carts, "Single Event Effect and Radiation Damage Results For Candidate Spacecraft," in *Proceedings of IEEE NSREC'98*, pp. 1 - 12, 1998.

[18] R. Ecoffet, "Overview of In-Orbit Radiation Induced Sapcecarft Anomalies," *IEEE Transactions On Nuclear Science,* vol. 60, no. 3, pp. 1791-1815, June 2013.

[19] A. Holmes-Sidele and L. Adams, Eds., Handbook of Radiation Effects, 2nd edition, Oxford University Press, 2007.

[20] R. Velazco, P. Fouilat and R. Reis, Radiation Effects on Embedded Systems, Springer, 2007.

[21] T. R. Oldham, "Basic Mechanisms of TID and DDD Response in MOS and Bipolar Microelectronics," in *IEEE NSREC 2011 Short Course Notes, Section II*, pp. 1 - 95, 2011.

[22] H. Quinn, "Challenges in Testing Complex Systems," *IEEE Transactions On Nuclear Science,* vol. 61, no. 2, pp. 766 - 786, April 2014.

[23] R. Ecoffet, "On-Orbit Anomalies: Investigations and Root Cause Determination," in *IEEE NSREC 2011 Short Course Notes, Section IV*, pp. 1 - 25, 2011.

[24] G. C. Messenger and M. S. Ash, Single Event Phenomena, Springer US, 1997.

[25] P. P. Shirvani, "Fault-Tolerant Computing for Radiation Environments," Ph.D. Thesis, Center for Reliable Computing, Stanford University, pp. 1 - 86, June 2001.

[26] F. L. Kastensmidt, R. Reis and L. Carro, Fault Tolerance Techniques for SRAM-based FPGAs, Springer, 2006.

[27] I. Koren and M. Krishna, Fault Tolerant systems, ElSEVIER, 2007.

[28] N. Oh, "Software implemented hardware fault tolerance," Ph.D. Thesis, Center for Reliable Computing, Stanford University, pp. 1 - 54, Dec. 2000.

[29] O. Goloubeva, M. Rebaudengo, M. S. Reorda and M. Violante, Software Implemented Hardware Fault Tolerance, Springer, 2006.

[30] L. L. Pullum, Software Fault Tolerance Techniques and Implementation, London: Artech House, 2001.

[31] F. G. d. Lima, "Single Event Upset Mitigation Techniques for Programmable Devices," Ph.D. Thesis, INSTITUTO DE INFORMÁTICA, UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, pp. 1 - 102, Dec. 2000.

[32] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan and M. Wirthlin, "Fine-Grain SEU Mitigation for FPGAs Using Partial TMR," *IEEE Transactions On Nuclear Science,* vol. 55, no. 4, pp. 2274 - 2280, August 2008.

[33] I. Herrera-Alzu and M. López-Vallejo, "Design Techniques for Xilinx Virtex FPGA Configuration Memory Scrubbers," *IEEE Transactions On Nuclear Science,* vol. 60, no. 1, pp. 376 - 385, Feb. 2013.

[34] I. Herrera-Alzu and M. López-Vallejo, "System Design Framework and Methodology for Xilinx Virtex FPGA Configuration Scrubbers," *IEEE Transactions On Nuclear Science,* vol. 61, no. 1, pp. 619 - 629, Feb. 2014.

[35] H. Baig, J.-A. Lee and Z. A. Siddiqui, "A Low-Overhead Multiple-SEU Mitigation Approach for SRAM-based FPGAs with Increased Reliability," *IEEE Transactions On Nuclear Science,* vol. 61, no. 3, pp. 1389 - 1399, June 2014.

[36] H. Quinn, P. Graham, K. Morgan, Z. Baker, M. Caffrey, D. Smith, M. Wirthlin and R. Bell, "Flight Experience of the Xilinx Virtex-4," *IEEE Transactions On Nuclear Science,* vol. 60, no. 4, pp. 2682 - 2690, August 2013.

[37] "Virtex-5 FPGA Configuration User Guide, XILINX UG191 (v3.10)," XILINX, pp. 1 - 166, 2011.

[38] L. Alkali, "An overview of flight computer technologies for future NASA space exploration missions," *Acta Astronautica,* vol. 52, no. 9, pp. 857 - 867, 2003.

[39] M. PIGNOL, "COTS-based applications in space avionics," in *Proceedings of the Conference on Design, Automation and Test in Europe DATE ’10, pp. 1213 – 1219*, 2010.

[40] R. Ginosar, "A survey of processors for space," in *Proceedings of the Data Systems in Aerospace (DASIA), Eurospace, pp. 1 - 5*, May 2012.

[41] M. H. Thoby, "MYRIADE: CNES micro-satellite program," in *Proceedings of the 15th Annual/USU Conference on Small Satellites*, pp. 1 - 14, 2001.

[42] S. Montenegro and W. Barwald, "Powerbird-modern spacecraft bus controller," *Acta Astronautica,* vol. 52, no. 9 - 12, pp. 957 - 963, 2003.

[43] P. Behr, W. Bärwald, K. Brieß and S. Montenegro, "Fault tolerance and COTS: Next generation of high performance satellite computers," in *Proceedings of the DASIA (ESA SP-532)*, pp. 1 - 12, June 2003.

[44] C. Villalpando, D. Rennels, R. Some and M. Cabanas-Holmen, "Reliable multicore processors for nasa space missions," in *Proceedings of the 2011 IEEE Aerospace Conference*, pp. 1 – 12, 2011.

[45] M. Fayyaz, T. Vladimirova and J.-M. Caujolle, "Adaptive middleware design for satellite fault-tolerant distributed computing," in *Proceedings of the 2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2012)*, pp. 1 - 8, June 2012.

[46] I. V. McLoughlin and T. Bretschneider, "Achieving lowcost high-reliability computation through redundant parallel processing," in *Proceedings of the IEEE International Conference on Computing Informatics ICOCI 2006*, pp. 1 - 6, June 2006.

[47] P. Murray, T. Randolph, D. V. Buren, D. Anderson and I. Troxel, "High performance, high volume reconfigurable processor architecture," in *Proceedings of the 2012 IEEE Aerospace Conference*, pp. 1 - 8, 2012.

[48] T. Vladimirova, W. Xiaofeng and C. Bridges, "Development of a Satellite Sensor Network for Future Space Missions," in *Proceedings of the 2008 IEEE Aerospace Conference*, pp. 1 - 10, 2008.

[49] H. Yashiro, T. Takahashi, T. Fujiwara and K. Mori, "A High Assurance Space On-Board Distributed Computer System," in *Proceedings of the 2003 IEEE Aerospace Conference*, pp.5_2501 - 5_2509, 2003.

[50] J. Samson, G. Gardner, D. Lupia, M. Patel, P. Davis, V. Aggarwal, A. George, Z. Kalbarcyzk and R. Some, "High Performance Dependable Multiprocessor II," in *Proceedings of the 2007 IEEE Aerospace Conference*, pp. 1 - 22, 2007.

[51] J. Samson and E. Grobelny, "NMP ST8 Dependable Multiprocessor: TRL6 Validation – Preliminary Results," in *Proceedings of the 2009 IEEE Aerospace Conference*, pp. 1 - 23, 2009.

[52] J. J. Samson, "Update on Dependable Multiprocessor Cubesat Technology Development," in *Proceedings of the 2012 IEEE Aerospace Conference*, pp. 1 - 12, 2012.

[53] J. J. Samson, E. Grobelny, S. Driesse-Bunn, M. Clark and S. Van Portfliet, "Post-TRL6 Dependable Multiprocessor Technology Developments," in *Proceedings of the 2010*

IEEE Aerospace Conference, pp. 1 - 21, 2010.

[54] D. Lüdtke, K. Westerdorff, K. Stohlmann, A. Börner, O. Maibaum, T. Peng, B. Weps, G. Fey and A. Gerndt, "OBC-NG: Towards a Reconfigurable On-board Computing Architecture for Spacecraft," in *Proceedings of the 2014 IEEE Aerospace Conference*, pp. 1 - 13, 2014.

[55] "RAD750® radiation-hardened PowerPC microprocessor," BAE Systems, pp. 1 - 2, 2008.

[56] M. Ibrahim, K. Asami and M. Cho, "Fault Tolerant Architecture Alternatives for Developing Nano-Satellites Embedded Computers," in *Proceedings of the AIAA Space 2012 Conference & Exposition*, pp. 1 - 8, Sept. 2012.

[57] M. Ibrahim, K. Asami and M. Cho, "The Use of Reconfigurable FPGAs in Developing Reliable Satellite On-Board Computers," in *Proceedings of the AIAA Space 2012 Conference & Exposition*, pp. 1 - 8, 2012.

[58] M. Ibrahim, K. Asami and M. Cho, "Reconfigurable Fault Tolerant Avionics System," in *Proceedings of the 2013 IEEE Aerospace Conference*, pp. 1 - 12, 2013.

[59] M. M. Ibrahim, "Model Based Design of a Smart Embedded System for Space Applications," M.Sc. Thesis, Computer & Systems Engineering Department, AL AZHAR University, pp. 1 - 113, 2011.

[60] "Virtex-5 FPGA System Monitor - UG 192," XILINX, pp. 1 - 66, Feb. 2011.

[61] "MAX6655- Dual Remote/Local Temperature Sensors and Four-Channel Voltage Monitors," MAXIM, datasheet, pp. 1 - 18, 2006.

[62] M. Ibrahim, K. Asami and M. Cho, "Time and Space Redundancy Fault Tolerance Trade-offs for FPGA Based Single and Multicore Designs," *Transactions of the Japan Society for Aeronautical and Space Sciences,* vol. Article in Press, pp. 1 - 10, 2014.

[63] "SPENVIS: The Space Environment Information System," ESA, 1997-2014. [Online]. Available: https://www.spenvis.oma.be/.

[64] H. Quinn, K. Morgan, P. Graham, J. Krone and M. Caffrey, "Static Proton and Heavy Ion Testing of the Xilinx Virtex-5 Device," in *2007 IEEE Radiation Effects Data Workshop*, pp. 177 - 184, July 2007.

[65] M. Ibrahim, K. Asami and M. Cho, "LEO Single Event Upset Emulator for Validation of FPGA Based Avionics Systems," *Transactions of the Japan Society for Aeronautical and Space Sciences, Aerospace Technology Japan,* vol. 12, no. ists29, pp. Tf_19-Tf_25, 2014.

[66] M. Ibrahim, K. Asami and M. Cho, "Evaluation of SRAM Based FPGA Performance by Simulating SEU through Fault Injection," in *Proceedings of the 2013 6th International*

Conference on Recent Advances in Space Technologies (RAST), pp. 1 - 6, 2013.

[67] M. Ibrahim, K. Asami and M. Cho, "Thermal Vacuum Test Results for Virtex-5 FPGA Based Multi-core On-Board Computer," in *Proceedings of the 2014 IEEE Aerospace Conference*, pp. 1 - 9, 2014.

[68] D. Q. Dinh, "Thermal Modelling of Nanosat," M.Sc., Department of Mechanical and Aerospace Engineering, San José State University, pp. 1 - 47, 2012.

[69] G. Cook, "Satellite Drag Coefficients," *Planetary and Space Science ELSEVIER,* vol. 13, no. 10, pp. 929 - 946, 1965.

[70] F. P. Incropera, Fundamentals of Heat and Mass Transfer, New York: John Wiley & Sons, 2006.

[71] P. Fortescue, G. Swinerd and J. Stark, Spacecraft Systems Engineering, 4th Edition, Chichester, West Sussex: John Wiley & Sons, Ltd, 2011.

[72] C. B. VanOutryve, "A Thermal Analysis and Design Tool for Small Spacecraft," San Jose State University, 2008.

[73] D. F. Gluck and V. Baturkin, Spacecraft Thermal Control Handbook. Vol I: Fundamental Technologies., vol. I, D. G. Gilmore, Ed., El Segundo, California: The Aerospace Press, 2002, pp. 247 - 329, Mounting and Interfaces.

[74] Y. A. Cengel, Heat Transfer: A Practical Approach, San Francisco: McGraw Hill, 1998.

[75] D. G. Gilmore and R. L. Collins, Spacecraft Thermal Control Handbook. Vol I: Fundamental Technologies, D. G. Gilmore, Ed., El Segundo, CA: The Aerospace Press, 2002.

[76] J. R. Howell, R. Siegel and M. P. Menguc, Thermal Radiation Heat Transfer, 5th Edition, CRC Press, 2010.

[77] "Power Methodology Guide UG786 (v13.1)," XILINX, pp. 1 - 54, March 2011.

[78] "XPower Estimator User Guide UG440 (v13.4)," XILINX, pp. 1 - 64, Jan. 2012.

[79] P. Abusaidi, M. Klein and B. Philofsky, "Virtex-5 FPGA System Power Design Considerations WP285 (v1.0)," XILINX, pp. 1 - 23, Feb. 2008.

[80] M. Klein, "Static Power and the Importance of Realistic Junction Temperature Analysis WP221 (v1.0)," XILINX, pp. 1 - 10, March 2005.

[81] D. Curd, "Power Consumption in 65 nm FPGAs WP246 (v1.2)," XILINX, pp. 1 - 12, Feb. 2007.

[82] M. Cho, *Space Environment Testing,* Kyushu Institute of Technology, Lecture Notes, Thermal test part 2, pp. 1 - 64, 2014.

[83] "SMC Standard SMC-S-016 TEST REQUIREMENTS FOR LAUNCH, UPPER-STAGE AND SPACE VEHICLES," pp. 1 - 112, 13 June 2008.

[84] K. Chapman, "SEU Strategies for Virtex-5 Devices XAPP 864 (V2.0)," XILINX, pp. 1 - 16, April 2010.

[85] G. Allen, L. Edmonds, G. Swift, C. Carmichael, C. W. Tseng, K. Heldt, S. Anderson and M. Coe, "Single Event Test Methodologies and System Error Rate Analysis for Triple Modular Redundant Field Programmable Gate Arrays," *IEEE Transactions on Nuclear Science,* vol. 58, no. 3, pp. 1040 - 1046, June 2011.

[86] L. D. Edmonds, "Analysis of SEU Rates in TMR Devices," JPL Publication 09-6, pp. 1 - 18, Feb. 2009.

[87] "NETS Web," Center for Nanosatellite Testing - Kyushu Institute of Technology, [Online]. Available: http://cent.ele.kyutech.ac.jp/nets_web/nets_web.html .

[88] M. Peck, "Exploring Space with Chip-sized Satellites," July 2011. [Online]. Available: http://spectrum.ieee.org/aerospace/satellites/exploring-space-with-chipsized-satellites.

Appendix A

Microprocessor Hardware Specification of the MPSoC

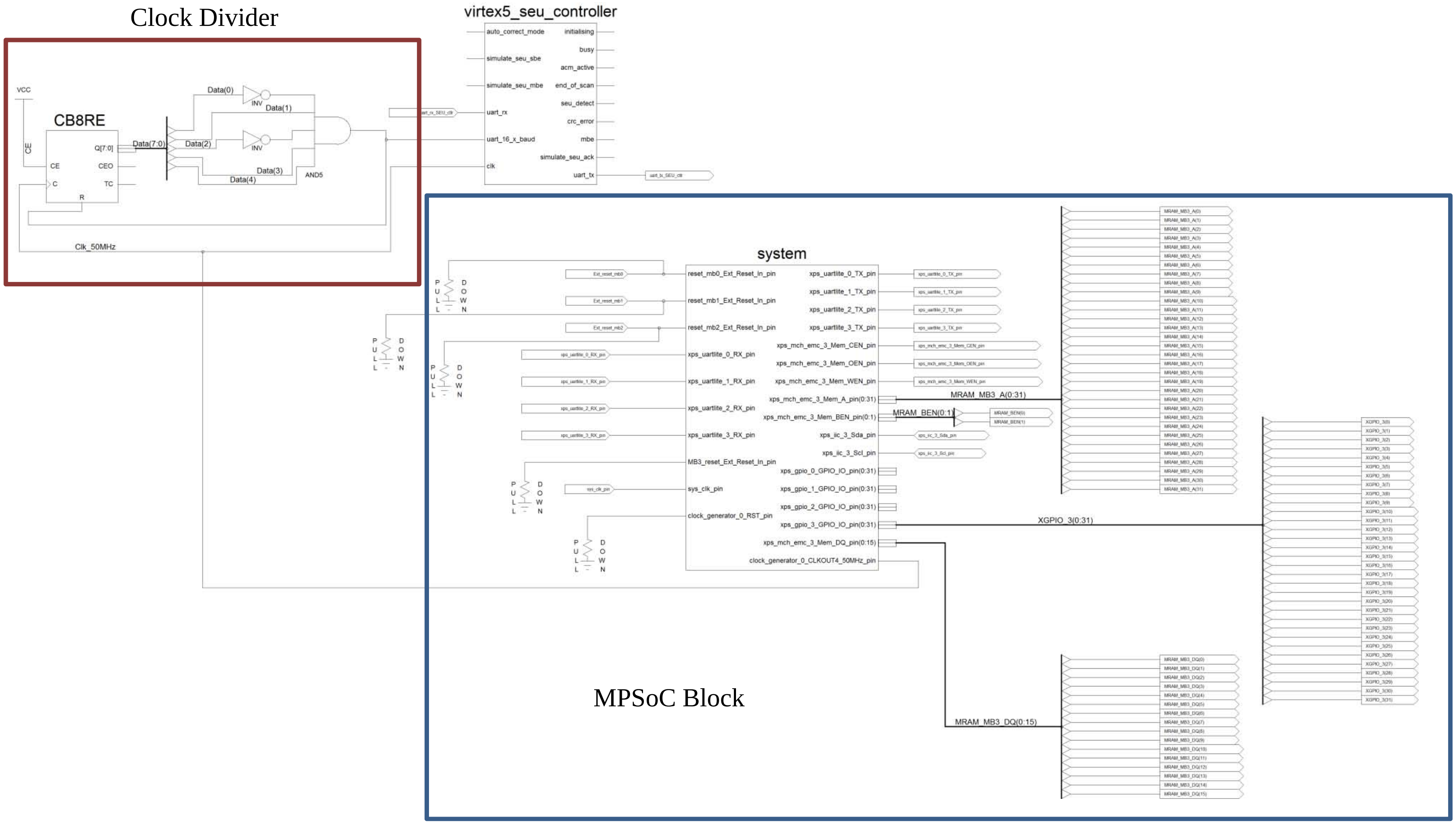
```
PARAMETER VERSION = 2.1.0

PORT reset_mb0_Ext_Reset_In_pin = net_reset_mb0_Ext_Reset_In_pin, DIR = I, SIGIS = RST
PORT reset_mb1_Ext_Reset_In_pin = net_reset_mb1_Ext_Reset_In_pin, DIR = I, SIGIS = RST
PORT reset_mb2_Ext_Reset_In_pin = net_reset_mb2_Ext_Reset_In_pin, DIR = I, SIGIS = RST
PORT xps_uartlite_0_RX_pin = net_xps_uartlite_0_RX_pin, DIR = I
PORT xps_uartlite_0_TX_pin = xps_uartlite_0_TX, DIR = O
PORT xps_uartlite_1_RX_pin = net_xps_uartlite_1_RX_pin, DIR = I
PORT xps_uartlite_1_TX_pin = xps_uartlite_1_TX, DIR = O
PORT xps_uartlite_2_RX_pin = net_xps_uartlite_2_RX_pin, DIR = I
PORT xps_uartlite_2_TX_pin = xps_uartlite_2_TX, DIR = O
PORT xps_uartlite_3_RX_pin = net_xps_uartlite_3_RX_pin, DIR = I
PORT xps_uartlite_3_TX_pin = xps_uartlite_3_TX, DIR = O
PORT xps_iic_3_Sda_pin = xps_iic_3_Sda, DIR = IO
PORT xps_iic_3_Scl_pin = xps_iic_3_Scl, DIR = IO
PORT xps_gpio_0_GPIO_IO_pin = xps_gpio_0_GPIO_IO, DIR = IO, VEC = [0:31]
PORT xps_gpio_1_GPIO_IO_pin = xps_gpio_1_GPIO_IO, DIR = IO, VEC = [0:31]
PORT xps_gpio_2_GPIO_IO_pin = xps_gpio_2_GPIO_IO, DIR = IO, VEC = [0:31]
PORT xps_gpio_3_GPIO_IO_pin = xps_gpio_3_GPIO_IO, DIR = IO, VEC = [0:31]
PORT xps_mch_emc_3_Mem_DQ_pin = xps_mch_emc_3_Mem_DQ, DIR = IO, VEC = [0:15]
PORT xps_mch_emc_3_Mem_A_pin = xps_mch_emc_3_Mem_A, DIR = O, VEC = [0:31]
PORT xps_mch_emc_3_Mem_CEN_pin = xps_mch_emc_3_Mem_CEN, DIR = O
PORT xps_mch_emc_3_Mem_OEN_pin = xps_mch_emc_3_Mem_OEN, DIR = O
PORT xps_mch_emc_3_Mem_WEN_pin = xps_mch_emc_3_Mem_WEN, DIR = O
PORT xps_mch_emc_3_Mem_BEN_pin = xps_mch_emc_3_Mem_BEN, DIR = O, VEC = [0:1]
PORT MB3_reset_Ext_Reset_In_pin = net_MB3_reset_Ext_Reset_In_pin, DIR = I, SIGIS = RST
PORT sys_clk_pin = CLK_S, DIR = I, SIGIS = CLK, CLK_FREQ = 50000000
PORT clock_generator_0_RST_pin = net_clock_generator_0_RST_pin, DIR = I, SIGIS = RST
PORT clock_generator_0_CLKOUT4_50MHz_pin = clock_generator_0_CLKOUT4_50MHz, DIR = O, SIGIS = CLK, CLK_FREQ = 50000000


BEGIN MicroBlaze
 PARAMETER INSTANCE = MicroBlaze_0
 PARAMETER HW_VER = 8.20.a
 PARAMETER C_FSL_LINKS = 3
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_FPU_EXCEPTION = 1
 PARAMETER C_DIV_ZERO_EXCEPTION = 1
 PARAMETER C_IPLB_BUS_EXCEPTION = 1
 PARAMETER C_DPLB_BUS_EXCEPTION = 1
 PARAMETER C_ILL_OPCODE_EXCEPTION = 1
 PARAMETER C_OPCODE_0x0_ILLEGAL = 1
 PARAMETER C_USE_STACK_PROTECTION = 1
 PARAMETER C_UNALIGNED_EXCEPTIONS = 1
 PARAMETER C_FSL_EXCEPTION = 1
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 2
 PARAMETER C_USE_DIV = 1
 PARAMETER C_USE_EXTENDED_FSL_INSTR = 1
 PARAMETER C_AREA_OPTIMIZED = 1
 PARAMETER C_NUMBER_OF_PC_BRK = 1
 PARAMETER C_NUMBER_OF_RD_ADDR_BRK = 0
 PARAMETER C_NUMBER_OF_WR_ADDR_BRK = 0
 BUS_INTERFACE DPLB = plb_v46_mb0
 BUS_INTERFACE IPLB = plb_v46_mb0
 BUS_INTERFACE DEBUG = debug_module_0_MBDEBUG_0
 BUS_INTERFACE SFSL1 = fsl_v20_mb1_mb0
 BUS_INTERFACE SFSL2 = fsl_v20_mb2_mb0
 BUS_INTERFACE DLMB = Data_lmb_v10_mb0
 BUS_INTERFACE ILMB = Inst_lmb_v10_mb0
 BUS_INTERFACE MFSL0 = fsl_v20_mb0
 BUS_INTERFACE MFSL1 = fsl_v20_mb0_mb1
 BUS_INTERFACE MFSL2 = fsl_v20_mb0_mb2
 PORT INTERRUPT = xps_intc_0_Irq
 PORT DBG_STOP = Ext_BRK
 PORT MB_RESET = reset_mb0_MB_Reset
END

BEGIN MicroBlaze
 PARAMETER INSTANCE = MicroBlaze_1
 PARAMETER HW_VER = 8.20.a
 PARAMETER C_FSL_LINKS = 3
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_FPU_EXCEPTION = 1
 PARAMETER C_DIV_ZERO_EXCEPTION = 1
 PARAMETER C_IPLB_BUS_EXCEPTION = 1
 PARAMETER C_DPLB_BUS_EXCEPTION = 1
 PARAMETER C_ILL_OPCODE_EXCEPTION = 1
 PARAMETER C_OPCODE_0x0_ILLEGAL = 1
 PARAMETER C_USE_STACK_PROTECTION = 1
 PARAMETER C_UNALIGNED_EXCEPTIONS = 1
 PARAMETER C_FSL_EXCEPTION = 1
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 2
 PARAMETER C_USE_DIV = 1
 PARAMETER C_USE_EXTENDED_FSL_INSTR = 1
 PARAMETER C_AREA_OPTIMIZED = 1
 BUS_INTERFACE DPLB = plb_v46_mb1
 BUS_INTERFACE IPLB = plb_v46_mb1
 BUS_INTERFACE DEBUG = debug_module_0_MBDEBUG_1
 BUS_INTERFACE SFSL1 = fsl_v20_mb0_mb1
 BUS_INTERFACE SFSL2 = fsl_v20_mb2_mb1
 BUS_INTERFACE DLMB = Data_lmb_v10_mb1
 BUS_INTERFACE ILMB = Inst_lmb_v10_mb1
 BUS_INTERFACE MFSL0 = fsl_v20_mb1
 BUS_INTERFACE MFSL1 = fsl_v20_mb1_mb0
 BUS_INTERFACE MFSL2 = fsl_v20_mb1_mb2
 PORT INTERRUPT = xps_intc_1_Irq
 PORT DBG_STOP = Ext_BRK
 PORT MB_RESET = reset_mb1_MB_Reset
END

BEGIN MicroBlaze
 PARAMETER INSTANCE = MicroBlaze_2
 PARAMETER HW_VER = 8.20.a
 PARAMETER C_FSL_LINKS = 3
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_FPU_EXCEPTION = 1
 PARAMETER C_DIV_ZERO_EXCEPTION = 1
 PARAMETER C_IPLB_BUS_EXCEPTION = 1
 PARAMETER C_DPLB_BUS_EXCEPTION = 1
 PARAMETER C_ILL_OPCODE_EXCEPTION = 1
 PARAMETER C_OPCODE_0x0_ILLEGAL = 1
 PARAMETER C_USE_STACK_PROTECTION = 1
 PARAMETER C_UNALIGNED_EXCEPTIONS = 1
 PARAMETER C_FSL_EXCEPTION = 1
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 2
 PARAMETER C_USE_DIV = 1
 PARAMETER C_USE_EXTENDED_FSL_INSTR = 1
 PARAMETER C_AREA_OPTIMIZED = 1
 BUS_INTERFACE DPLB = plb_v46_mb2
 BUS_INTERFACE IPLB = plb_v46_mb2
 BUS_INTERFACE DEBUG = debug_module_0_MBDEBUG_2
 BUS_INTERFACE SFSL1 = fsl_v20_mb0_mb2
 BUS_INTERFACE SFSL2 = fsl_v20_mb1_mb2
 BUS_INTERFACE DLMB = Data_lmb_v10_mb2
 BUS_INTERFACE ILMB = Inst_lmb_v10_mb2
 BUS_INTERFACE MFSL0 = fsl_v20_mb2
 BUS_INTERFACE MFSL1 = fsl_v20_mb2_mb0
 BUS_INTERFACE MFSL2 = fsl_v20_mb2_mb1
 PORT INTERRUPT = xps_intc_2_Irq
 PORT DBG_STOP = Ext_BRK
 PORT MB_RESET = reset_mb2_MB_Reset
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb0
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_FSL_DEPTH = 128
 PARAMETER C_ASYNC_CLKS = 0
 PARAMETER C_IMPL_STYLE = 1
 PORT FSL_CLK = clock_generator_0_CLKOUT2_100MHz
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb2
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_FSL_DEPTH = 128
 PARAMETER C_ASYNC_CLKS = 0
 PARAMETER C_IMPL_STYLE = 1
 PORT FSL_CLK = clock_generator_0_CLKOUT0_100MHz
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb1
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_FSL_DEPTH = 128
 PARAMETER C_ASYNC_CLKS = 0
 PARAMETER C_IMPL_STYLE = 1
 PORT FSL_CLK = clock_generator_0_CLKOUT3_100MHz
END

BEGIN lmb_v10
 PARAMETER INSTANCE = Data_lmb_v10_mb0
 PARAMETER HW_VER = 2.00.b
 PORT SYS_Rst = reset_mb0_Bus_Struct_Reset
 PORT LMB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN lmb_v10
 PARAMETER INSTANCE = Data_lmb_v10_mb1
 PARAMETER HW_VER = 2.00.b
 PORT SYS_Rst = reset_mb1_Bus_Struct_Reset
 PORT LMB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN lmb_v10
 PARAMETER INSTANCE = Data_lmb_v10_mb2
 PARAMETER HW_VER = 2.00.b
 PORT SYS_Rst = reset_mb2_Bus_Struct_Reset
 PORT LMB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN lmb_v10
 PARAMETER INSTANCE = Inst_lmb_v10_mb0
 PARAMETER HW_VER = 2.00.b
 PORT SYS_Rst = reset_mb0_Bus_Struct_Reset
 PORT LMB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN lmb_v10
 PARAMETER INSTANCE = Inst_lmb_v10_mb1
 PARAMETER HW_VER = 2.00.b
 PORT SYS_Rst = reset_mb1_Bus_Struct_Reset
 PORT LMB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN lmb_v10
 PARAMETER INSTANCE = Inst_lmb_v10_mb2
 PARAMETER HW_VER = 2.00.b
 PORT SYS_Rst = reset_mb2_Bus_Struct_Reset
 PORT LMB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN plb_v46
 PARAMETER INSTANCE = plb_v46_mb0
 PARAMETER HW_VER = 1.05.a
 PORT SYS_Rst = reset_mb0_Bus_Struct_Reset
 PORT PLB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN plb_v46
 PARAMETER INSTANCE = plb_v46_mb1
 PARAMETER HW_VER = 1.05.a
 PORT SYS_Rst = reset_mb1_Bus_Struct_Reset
 PORT PLB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN plb_v46
 PARAMETER INSTANCE = plb_v46_mb2
 PARAMETER HW_VER = 1.05.a
 PORT SYS_Rst = reset_mb2_Bus_Struct_Reset
 PORT PLB_Clk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = xps_uartlite_0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0x80440000
 PARAMETER C_HIGHADDR = 0x8044007f
 BUS_INTERFACE SPLB = plb_v46_mb0
 PORT RX = net_xps_uartlite_0_RX_pin
 PORT TX = xps_uartlite_0_TX
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = xps_uartlite_1
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0x80440000
 PARAMETER C_HIGHADDR = 0x8044007f
 BUS_INTERFACE SPLB = plb_v46_mb1
 PORT RX = net_xps_uartlite_1_RX_pin
 PORT TX = xps_uartlite_1_TX
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = xps_uartlite_2
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0x80440000
 PARAMETER C_HIGHADDR = 0x8044007f
 BUS_INTERFACE SPLB = plb_v46_mb2
 PORT RX = net_xps_uartlite_2_RX_pin
 PORT TX = xps_uartlite_2_TX
END

BEGIN xps_iic
 PARAMETER INSTANCE = xps_iic_0
 PARAMETER HW_VER = 2.03.a
 PARAMETER C_BASEADDR = 0x80500000
 PARAMETER C_HIGHADDR = 0x805001ff
 BUS_INTERFACE SPLB = plb_v46_mb0
 PORT IIC2INTC_Irpt = xps_iic_0_IIC2INTC_Irpt
END

BEGIN xps_iic
 PARAMETER INSTANCE = xps_iic_1
 PARAMETER HW_VER = 2.03.a
 PARAMETER C_BASEADDR = 0x80500000
 PARAMETER C_HIGHADDR = 0x805001ff
 BUS_INTERFACE SPLB = plb_v46_mb1
 PORT IIC2INTC_Irpt = xps_iic_1_IIC2INTC_Irpt
END

BEGIN xps_iic
 PARAMETER INSTANCE = xps_iic_2
 PARAMETER HW_VER = 2.03.a
 PARAMETER C_BASEADDR = 0x80500000
 PARAMETER C_HIGHADDR = 0x805001ff
 BUS_INTERFACE SPLB = plb_v46_mb2
 PORT IIC2INTC_Irpt = xps_iic_2_IIC2INTC_Irpt
END

BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x80580100
 PARAMETER C_HIGHADDR = 0x805801ff
 BUS_INTERFACE SPLB = plb_v46_mb0
 PORT Interrupt = xps_timer_0_Interrupt
END

BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_1
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x80580100
 PARAMETER C_HIGHADDR = 0x805801ff
 BUS_INTERFACE SPLB = plb_v46_mb1
 PORT Interrupt = xps_timer_1_Interrupt
END

BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_2
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x80580100
 PARAMETER C_HIGHADDR = 0x805801ff
 BUS_INTERFACE SPLB = plb_v46_mb2
 PORT Interrupt = xps_timer_2_Interrupt
END

BEGIN xps_timebase_wdt
 PARAMETER INSTANCE = xps_timebase_wdt_0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x80540180
 PARAMETER C_HIGHADDR = 0x805401ff
 PARAMETER C_WDT_INTERVAL = 29
 BUS_INTERFACE SPLB = plb_v46_mb0
 PORT WDT_Interrupt = xps_timebase_wdt_0_WDT_Interrupt
 PORT Timebase_Interrupt = xps_timebase_wdt_0_Timebase_Interrupt
 PORT WDT_Reset = xps_timebase_wdt_0_WDT_Reset
END

BEGIN xps_timebase_wdt
 PARAMETER INSTANCE = xps_timebase_wdt_1
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x80540180
 PARAMETER C_HIGHADDR = 0x805401ff
 PARAMETER C_WDT_INTERVAL = 29
 BUS_INTERFACE SPLB = plb_v46_mb1
 PORT WDT_Interrupt = xps_timebase_wdt_1_WDT_Interrupt
 PORT Timebase_Interrupt = xps_timebase_wdt_1_Timebase_Interrupt
 PORT WDT_Reset = xps_timebase_wdt_1_WDT_Reset
END

BEGIN xps_timebase_wdt
 PARAMETER INSTANCE = xps_timebase_wdt_2
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x80540180
 PARAMETER C_HIGHADDR = 0x805401ff
 PARAMETER C_WDT_INTERVAL = 29
 BUS_INTERFACE SPLB = plb_v46_mb2
 PORT WDT_Interrupt = xps_timebase_wdt_2_WDT_Interrupt
 PORT Timebase_Interrupt = xps_timebase_wdt_2_Timebase_Interrupt
 PORT WDT_Reset = xps_timebase_wdt_2_WDT_Reset
END
```

```
BEGIN mdm
 PARAMETER INSTANCE = debug_module_0
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_USE_UART = 0
 PARAMETER C_MB_DBG_PORTS = 4
 BUS_INTERFACE MBDEBUG_0 = debug_module_0_MBDEBUG_0
 BUS_INTERFACE MBDEBUG_1 = debug_module_0_MBDEBUG_1
 BUS_INTERFACE MBDEBUG_2 = debug_module_0_MBDEBUG_2
 BUS_INTERFACE MBDEBUG_3 = debug_module_0_MBDEBUG_3
 PORT Interrupt = debug_module_0_Interrupt
 PORT Debug_SYS_Rst = debug_module_0_Debug_SYS_Reset
END

BEGIN xps_gpio
 PARAMETER INSTANCE = xps_gpio_0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_INTERRUPT_PRESENT = 1
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_BASEADDR = 0x80480000
 PARAMETER C_HIGHADDR = 0x804801ff
 PARAMETER C_IS_DUAL = 1
 BUS_INTERFACE SPLB = plb_v46_mb0
 PORT IP2INTC_Irpt = xps_gpio_0_IP2INTC_Irpt
 PORT GPIO_IO = xps_gpio_0_GPIO_IO
END

BEGIN xps_gpio
 PARAMETER INSTANCE = xps_gpio_1
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_INTERRUPT_PRESENT = 1
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_BASEADDR = 0x80480000
 PARAMETER C_HIGHADDR = 0x804801ff
 PARAMETER C_IS_DUAL = 1
 BUS_INTERFACE SPLB = plb_v46_mb1
 PORT IP2INTC_Irpt = xps_gpio_1_IP2INTC_Irpt
 PORT GPIO_IO = xps_gpio_1_GPIO_IO
END

BEGIN xps_gpio
 PARAMETER INSTANCE = xps_gpio_2
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_INTERRUPT_PRESENT = 1
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_BASEADDR = 0x80480000
 PARAMETER C_HIGHADDR = 0x804801ff
 PARAMETER C_IS_DUAL = 1
 BUS_INTERFACE SPLB = plb_v46_mb2
 PORT IP2INTC_Irpt = xps_gpio_2_IP2INTC_Irpt
 PORT GPIO_IO = xps_gpio_2_GPIO_IO
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_IRQ_IS_LEVEL = 0
 PARAMETER C_BASEADDR = 0x804c0100
 PARAMETER C_HIGHADDR = 0x804c017f
 BUS_INTERFACE SPLB = plb_v46_mb0
 PORT Irq = xps_intc_0_Irq
 PORT Intr = xps_timebase_wdt_0_WDT_Interrupt & xps_timebase_wdt_0_Timebase_Interrupt &
xps_timer_0_Interrupt & xps_gpio_0_IP2INTC_Irpt & xps_iic_0_IIC2INTC_Irpt
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_1
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_IRQ_IS_LEVEL = 0
 PARAMETER C_BASEADDR = 0x804c0100
 PARAMETER C_HIGHADDR = 0x804c017f
 BUS_INTERFACE SPLB = plb_v46_mb1
 PORT Irq = xps_intc_1_Irq
 PORT Intr = xps_timebase_wdt_1_WDT_Interrupt & xps_timebase_wdt_1_Timebase_Interrupt &
xps_timer_1_Interrupt & xps_gpio_1_IP2INTC_Irpt & xps_iic_1_IIC2INTC_Irpt
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_2
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_IRQ_IS_LEVEL = 0
 PARAMETER C_BASEADDR = 0x804c0100
 PARAMETER C_HIGHADDR = 0x804c017f
 BUS_INTERFACE SPLB = plb_v46_mb2
 PORT Irq = xps_intc_2_Irq
 PORT Intr = xps_timebase_wdt_2_WDT_Interrupt & xps_timebase_wdt_2_Timebase_Interrupt &
xps_timer_2_Interrupt & xps_gpio_2_IP2INTC_Irpt & xps_iic_2_IIC2INTC_Irpt
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb0_mb1
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_FSL_DEPTH = 512
 PARAMETER C_ASYNC_CLKS = 1
 PORT FSL_M_CLK = clock_generator_0_CLKOUT2_100MHz
 PORT FSL_S_CLK = clock_generator_0_CLKOUT3_100MHz
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb0_mb2
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_FSL_DEPTH = 512
 PARAMETER C_ASYNC_CLKS = 1
 PORT FSL_M_CLK = clock_generator_0_CLKOUT2_100MHz
 PORT FSL_S_CLK = clock_generator_0_CLKOUT0_100MHz
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb1_mb0
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_FSL_DEPTH = 512
 PARAMETER C_ASYNC_CLKS = 1
 PORT FSL_M_CLK = clock_generator_0_CLKOUT3_100MHz
 PORT FSL_S_CLK = clock_generator_0_CLKOUT2_100MHz
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb1_mb2
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_FSL_DEPTH = 512
 PARAMETER C_ASYNC_CLKS = 1
 PORT FSL_M_CLK = clock_generator_0_CLKOUT3_100MHz
 PORT FSL_S_CLK = clock_generator_0_CLKOUT0_100MHz
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb2_mb0
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_FSL_DEPTH = 512
 PARAMETER C_ASYNC_CLKS = 1
 PORT FSL_M_CLK = clock_generator_0_CLKOUT0_100MHz
 PORT FSL_S_CLK = clock_generator_0_CLKOUT2_100MHz
END

BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_v20_mb2_mb1
 PARAMETER HW_VER = 2.11.e
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_FSL_DEPTH = 512
 PARAMETER C_ASYNC_CLKS = 1
 PORT FSL_M_CLK = clock_generator_0_CLKOUT0_100MHz
 PORT FSL_S_CLK = clock_generator_0_CLKOUT3_100MHz
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_mb0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_EXT_RST_WIDTH = 2
 PARAMETER C_AUX_RST_WIDTH = 2
 PORT Bus_Struct_Reset = reset_mb0_Bus_Struct_Reset
 PORT MB_Debug_Sys_Rst = debug_module_0_Debug_SYS_Reset
 PORT Ext_Reset_In = net_reset_mb0_Ext_Reset_In_pin
 PORT Aux_Reset_In = xps_timebase_wdt_0_WDT_Reset
 PORT MB_Reset = reset_mb0_MB_Reset
 PORT Slowest_sync_clk = clock_generator_0_CLKOUT3_100MHz
 PORT Dcm_locked = Dcm_all_locked
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_mb1
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_EXT_RST_WIDTH = 2
 PARAMETER C_AUX_RST_WIDTH = 2
 PORT Bus_Struct_Reset = reset_mb1_Bus_Struct_Reset
 PORT MB_Debug_Sys_Rst = debug_module_0_Debug_SYS_Reset
 PORT Ext_Reset_In = net_reset_mb1_Ext_Reset_In_pin
 PORT Aux_Reset_In = xps_timebase_wdt_1_WDT_Reset
 PORT MB_Reset = reset_mb1_MB_Reset
 PORT Slowest_sync_clk = clock_generator_0_CLKOUT3_100MHz
 PORT Dcm_locked = Dcm_all_locked
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_mb2
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_EXT_RST_WIDTH = 2
 PARAMETER C_AUX_RST_WIDTH = 2
 PORT Bus_Struct_Reset = reset_mb2_Bus_Struct_Reset
 PORT MB_Debug_Sys_Rst = debug_module_0_Debug_SYS_Reset
 PORT Ext_Reset_In = net_reset_mb2_Ext_Reset_In_pin
 PORT Aux_Reset_In = xps_timebase_wdt_2_WDT_Reset
 PORT MB_Reset = reset_mb2_MB_Reset
 PORT Slowest_sync_clk = clock_generator_0_CLKOUT3_100MHz
 PORT Dcm_locked = Dcm_all_locked
END

BEGIN bram_block
 PARAMETER INSTANCE = bram_block_mb0
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = bram_cntlr_Inst_mb0_BRAM_PORT
 BUS_INTERFACE PORTB = bram_cntlr_Data_mb0_BRAM_PORT
END

BEGIN bram_block
 PARAMETER INSTANCE = bram_block_mb1
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = bram_cntlr_Inst_mb1_BRAM_PORT
 BUS_INTERFACE PORTB = bram_cntlr_Data_mb1_BRAM_PORT
END

BEGIN bram_block
 PARAMETER INSTANCE = bram_block_mb2
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = bram_cntlr_Inst_mb2_BRAM_PORT
 BUS_INTERFACE PORTB = bram_cntlr_Data_mb2_BRAM_PORT
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = bram_cntlr_Inst_mb0
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00007fff
 BUS_INTERFACE SLMB = Inst_lmb_v10_mb0
 BUS_INTERFACE BRAM_PORT = bram_cntlr_Inst_mb0_BRAM_PORT
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = bram_cntlr_Inst_mb1
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00007fff
 BUS_INTERFACE SLMB = Inst_lmb_v10_mb1
 BUS_INTERFACE BRAM_PORT = bram_cntlr_Inst_mb1_BRAM_PORT
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = bram_cntlr_Inst_mb2
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00007fff
 BUS_INTERFACE SLMB = Inst_lmb_v10_mb2
 BUS_INTERFACE BRAM_PORT = bram_cntlr_Inst_mb2_BRAM_PORT
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = bram_cntlr_Data_mb0
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00007fff
 BUS_INTERFACE SLMB = Data_lmb_v10_mb0
 BUS_INTERFACE BRAM_PORT = bram_cntlr_Data_mb0_BRAM_PORT
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = bram_cntlr_Data_mb1
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00007fff
 BUS_INTERFACE SLMB = Data_lmb_v10_mb1
 BUS_INTERFACE BRAM_PORT = bram_cntlr_Data_mb1_BRAM_PORT
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = bram_cntlr_Data_mb2
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00007fff
 BUS_INTERFACE SLMB = Data_lmb_v10_mb2
 BUS_INTERFACE BRAM_PORT = bram_cntlr_Data_mb2_BRAM_PORT
END

BEGIN fsl_v20_voter
 PARAMETER INSTANCE = fsl_voter_0
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE SFSL = fsl_v20_mb0
END

BEGIN fsl_v20_voter
 PARAMETER INSTANCE = fsl_voter_1
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE SFSL = fsl_v20_mb1
END

BEGIN fsl_v20_voter
 PARAMETER INSTANCE = fsl_voter_2
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE SFSL = fsl_v20_mb2
END

BEGIN plb_v46
 PARAMETER INSTANCE = plb_v46_0_HWICAP
 PARAMETER HW_VER = 1.05.a
 PORT SYS_Rst = MB3_reset_Bus_Struct_Reset
 PORT PLB_Clk = clock_generator_0_CLKOUT1_100MHz
END

BEGIN MicroBlaze
 PARAMETER INSTANCE = MicroBlaze_3
 PARAMETER HW_VER = 8.20.a
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_AREA_OPTIMIZED = 1
 BUS_INTERFACE DPLB = plb_v46_0_HWICAP
 BUS_INTERFACE IPLB = plb_v46_0_HWICAP
 BUS_INTERFACE DEBUG = debug_module_0_MBDEBUG_3
 BUS_INTERFACE DLMB = MB3_lmb_v10_Data
 BUS_INTERFACE ILMB = MB3_lmb_v10_Inst
 PORT MB_RESET = MB3_reset_MB_Reset
 PORT INTERRUPT = xps_intc_3_Irq
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = MB3_reset
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_EXT_RST_WIDTH = 2
 PARAMETER C_AUX_RST_WIDTH = 2
 PORT Bus_Struct_Reset = MB3_reset_Bus_Struct_Reset
 PORT MB_Debug_Sys_Rst = debug_module_0_Debug_SYS_Reset
 PORT Ext_Reset_In = net_MB3_reset_Ext_Reset_In_pin
 PORT Aux_Reset_In = xps_timebase_wdt_3_WDT_Reset
 PORT MB_Reset = MB3_reset_MB_Reset
 PORT Slowest_sync_clk = clock_generator_0_CLKOUT1_100MHz
 PORT Dcm_locked = Dcm_all_locked
END

BEGIN bram_block
 PARAMETER INSTANCE = bram_block_0
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = MB3_Inst_BRAM_Cntlr_BRAM_PORT
 BUS_INTERFACE PORTB = MB3_Data_BRAM_Cntlr_BRAM_PORT
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = MB3_Inst_BRAM_Cntlr
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00007fff
 BUS_INTERFACE SLMB = MB3_lmb_v10_Inst
 BUS_INTERFACE BRAM_PORT = MB3_Inst_BRAM_Cntlr_BRAM_PORT
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = MB3_Data_BRAM_Cntlr
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00007fff
 BUS_INTERFACE SLMB = MB3_lmb_v10_Data
 BUS_INTERFACE BRAM_PORT = MB3_Data_BRAM_Cntlr_BRAM_PORT
END

BEGIN lmb_v10
 PARAMETER INSTANCE = MB3_lmb_v10_Data
 PARAMETER HW_VER = 2.00.b
 PORT SYS_Rst = MB3_reset_Bus_Struct_Reset
 PORT LMB_Clk = clock_generator_0_CLKOUT1_100MHz
END

BEGIN lmb_v10
 PARAMETER INSTANCE = MB3_lmb_v10_Inst
 PARAMETER HW_VER = 2.00.b
 PORT SYS_Rst = MB3_reset_Bus_Struct_Reset
 PORT LMB_Clk = clock_generator_0_CLKOUT1_100MHz
END

BEGIN xps_timebase_wdt
 PARAMETER INSTANCE = xps_timebase_wdt_3
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x80004800
 PARAMETER C_HIGHADDR = 0x8000487f
 PARAMETER C_WDT_INTERVAL = 29
 BUS_INTERFACE SPLB = plb_v46_0_HWICAP
 PORT Timebase_Interrupt = xps_timebase_wdt_3_Timebase_Interrupt
 PORT WDT_Interrupt = xps_timebase_wdt_3_WDT_Interrupt
 PORT WDT_Reset = xps_timebase_wdt_3_WDT_Reset
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_3
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_BASEADDR = 0x80004400
 PARAMETER C_HIGHADDR = 0x8000447f
 BUS_INTERFACE SPLB = plb_v46_0_HWICAP
 PORT Irq = xps_intc_3_Irq
 PORT Intr = xps_timebase_wdt_3_WDT_Interrupt & xps_timebase_wdt_3_Timebase_Interrupt &
xps_gpio_3_IP2INTC_Irpt & xps_iic_3_IIC2INTC_Irpt
END

BEGIN xps_mch_emc
 PARAMETER INSTANCE = xps_mch_emc_0
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_NUM_CHANNELS = 0
 PARAMETER C_MEM0_BASEADDR = 0x80600000
 PARAMETER C_MEM0_HIGHADDR = 0x807fffff
 BUS_INTERFACE SPLB = plb_v46_mb0
 PORT RdClk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN xps_mch_emc
 PARAMETER INSTANCE = xps_mch_emc_1
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_NUM_CHANNELS = 0
 PARAMETER C_MEM0_BASEADDR = 0x80600000
 PARAMETER C_MEM0_HIGHADDR = 0x807fffff
 BUS_INTERFACE SPLB = plb_v46_mb1
 PORT RdClk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN xps_mch_emc
 PARAMETER INSTANCE = xps_mch_emc_2
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_NUM_CHANNELS = 0
 PARAMETER C_MEM0_BASEADDR = 0x80600000
 PARAMETER C_MEM0_HIGHADDR = 0x807fffff
 BUS_INTERFACE SPLB = plb_v46_mb2
 PORT RdClk = clock_generator_0_CLKOUT0_100MHz
END

BEGIN xps_mch_emc
 PARAMETER INSTANCE = xps_mch_emc_3
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_NUM_CHANNELS = 0
 PARAMETER C_MEM0_BASEADDR = 0x80200000
 PARAMETER C_MEM0_HIGHADDR = 0x8023FFFF
 PARAMETER C_TCEDV_PS_MEM_0 = 35000
 PARAMETER C_TAVDV_PS_MEM_0 = 35000
 PARAMETER C_THZCE_PS_MEM_0 = 15000
 PARAMETER C_THZOE_PS_MEM_0 = 10000
 PARAMETER C_TWC_PS_MEM_0 = 35000
 PARAMETER C_TWP_PS_MEM_0 = 15000
 PARAMETER C_TLZWE_PS_MEM_0 = 3000
 PARAMETER C_MAX_MEM_WIDTH = 16
 PARAMETER C_MEM0_WIDTH = 16
 PARAMETER C_INCLUDE_DATAWIDTH_MATCHING_0 = 1
 BUS_INTERFACE SPLB = plb_v46_0_HWICAP
 PORT RdClk = clock_generator_0_CLKOUT1_100MHz
 PORT Mem_DQ = xps_mch_emc_3_Mem_DQ
 PORT Mem_CKEN = xps_mch_emc_3_Mem_CKEN
 PORT Mem_A = xps_mch_emc_3_Mem_A
 PORT Mem_RPN = xps_mch_emc_3_Mem_RPN
 PORT Mem_CEN = xps_mch_emc_3_Mem_CEN
 PORT Mem_OEN = xps_mch_emc_3_Mem_OEN
 PORT Mem_WEN = xps_mch_emc_3_Mem_WEN
 PORT Mem_BEN = xps_mch_emc_3_Mem_BEN
 PORT Mem_CE = xps_mch_emc_3_Mem_CE
 PORT Mem_ADV_LDN = xps_mch_emc_3_Mem_ADV_LDN
END

BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER HW_VER = 4.03.a
 PARAMETER C_CLKIN_FREQ = 50000000
 PARAMETER C_CLKOUT0_FREQ = 100000000
 PARAMETER C_CLKOUT0_PHASE = 0
 PARAMETER C_CLKOUT0_GROUP = NONE
 PARAMETER C_CLKOUT0_BUF = TRUE
 PARAMETER C_CLKOUT1_FREQ = 100000000
 PARAMETER C_CLKOUT1_PHASE = 0
 PARAMETER C_CLKOUT1_GROUP = NONE
 PARAMETER C_CLKOUT1_BUF = TRUE
 PARAMETER C_CLKOUT2_FREQ = 100000000
 PARAMETER C_CLKOUT2_PHASE = 0
 PARAMETER C_CLKOUT2_GROUP = NONE
 PARAMETER C_CLKOUT2_BUF = TRUE
 PARAMETER C_CLKOUT3_FREQ = 100000000
 PARAMETER C_CLKOUT3_PHASE = 0
 PARAMETER C_CLKOUT3_GROUP = NONE
 PARAMETER C_CLKOUT3_BUF = TRUE
 PARAMETER C_CLKOUT4_FREQ = 50000000
 PORT CLKIN = CLK_S
 PORT CLKOUT0 = clock_generator_0_CLKOUT0_100MHz
 PORT CLKOUT1 = clock_generator_0_CLKOUT1_100MHz
 PORT CLKOUT2 = clock_generator_0_CLKOUT2_100MHz
 PORT CLKOUT3 = clock_generator_0_CLKOUT3_100MHz
 PORT LOCKED = Dcm_all_locked
 PORT RST = net_clock_generator_0_RST_pin
 PORT CLKOUT4 = clock_generator_0_CLKOUT4_50MHz
END

BEGIN mailbox
 PARAMETER INSTANCE = mailbox_mb0_mb1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_ASYNC_CLKS = 1
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_MAILBOX_DEPTH = 512
 PARAMETER C_SPLB0_BASEADDR = 0x80440400
 PARAMETER C_SPLB0_HIGHADDR = 0x804407ff
 PARAMETER C_SPLB1_BASEADDR = 0x80440400
 PARAMETER C_SPLB1_HIGHADDR = 0x804407ff
 BUS_INTERFACE SPLB0 = plb_v46_mb0
 BUS_INTERFACE SPLB1 = plb_v46_mb1
END

BEGIN mailbox
 PARAMETER INSTANCE = mailbox_mb0_mb2
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_ASYNC_CLKS = 1
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_MAILBOX_DEPTH = 512
 PARAMETER C_SPLB0_BASEADDR = 0x80440800
 PARAMETER C_SPLB0_HIGHADDR = 0x80440bff
 PARAMETER C_SPLB1_BASEADDR = 0x80440400
 PARAMETER C_SPLB1_HIGHADDR = 0x804407ff
 BUS_INTERFACE SPLB0 = plb_v46_mb0
 BUS_INTERFACE SPLB1 = plb_v46_mb2
END

BEGIN mailbox
 PARAMETER INSTANCE = mailbox_mb0_mb3
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_ASYNC_CLKS = 1
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_MAILBOX_DEPTH = 512
 PARAMETER C_SPLB0_BASEADDR = 0x80440c00
 PARAMETER C_SPLB0_HIGHADDR = 0x80440fff
 PARAMETER C_SPLB1_BASEADDR = 0x80007c00
 PARAMETER C_SPLB1_HIGHADDR = 0x80007fff
 BUS_INTERFACE SPLB0 = plb_v46_mb0
 BUS_INTERFACE SPLB1 = plb_v46_0_HWICAP
END

BEGIN mailbox
 PARAMETER INSTANCE = mailbox_mb1_mb2
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_ASYNC_CLKS = 1
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_MAILBOX_DEPTH = 512
 PARAMETER C_SPLB0_BASEADDR = 0x80440800
 PARAMETER C_SPLB0_HIGHADDR = 0x80440bff
 PARAMETER C_SPLB1_BASEADDR = 0x80440800
 PARAMETER C_SPLB1_HIGHADDR = 0x80440bff
 BUS_INTERFACE SPLB0 = plb_v46_mb1
 BUS_INTERFACE SPLB1 = plb_v46_mb2
END

BEGIN mailbox
 PARAMETER INSTANCE = mailbox_mb1_mb3
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_ASYNC_CLKS = 1
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_MAILBOX_DEPTH = 512
 PARAMETER C_SPLB0_BASEADDR = 0x80440c00
 PARAMETER C_SPLB0_HIGHADDR = 0x80440fff
 PARAMETER C_SPLB1_BASEADDR = 0x80006800
 PARAMETER C_SPLB1_HIGHADDR = 0x80006bff
 BUS_INTERFACE SPLB0 = plb_v46_mb1
 BUS_INTERFACE SPLB1 = plb_v46_0_HWICAP
END
```

```
BEGIN mailbox
 PARAMETER INSTANCE = mailbox_mb2_mb3
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_ASYNC_CLKS = 1
 PARAMETER C_IMPL_STYLE = 1
 PARAMETER C_MAILBOX_DEPTH = 512
 PARAMETER C_SPLB0_BASEADDR = 0x80440c00
 PARAMETER C_SPLB0_HIGHADDR = 0x80440fff
 PARAMETER C_SPLB1_BASEADDR = 0x80007000
 PARAMETER C_SPLB1_HIGHADDR = 0x800073ff
 BUS_INTERFACE SPLB0 = plb_v46_mb2
 BUS_INTERFACE SPLB1 = plb_v46_0_HWICAP
END

BEGIN xps_gpio
 PARAMETER INSTANCE = xps_gpio_3
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_INTERRUPT_PRESENT = 1
 PARAMETER C_BASEADDR = 0x80004a00
 PARAMETER C_HIGHADDR = 0x80004bff
 BUS_INTERFACE SPLB = plb_v46_0_HWICAP
 PORT IP2INTC_Irpt = xps_gpio_3_IP2INTC_Irpt
 PORT GPIO_IO = xps_gpio_3_GPIO_IO[0:31]
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = xps_uartlite_3
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0x80001800
 PARAMETER C_HIGHADDR = 0x8000187f
 BUS_INTERFACE SPLB = plb_v46_0_HWICAP
 PORT RX = net_xps_uartlite_3_RX_pin
 PORT TX = xps_uartlite_3_TX
END

BEGIN xps_iic
 PARAMETER INSTANCE = xps_iic_3
 PARAMETER HW_VER = 2.03.a
 PARAMETER C_BASEADDR = 0x81600000
 PARAMETER C_HIGHADDR = 0x8160ffff
 PARAMETER C_IIC_FREQ = 50000
 PARAMETER C_SCL_INERTIAL_DELAY = 0
 PARAMETER C_SDA_INERTIAL_DELAY = 0
 BUS_INTERFACE SPLB = plb_v46_0_HWICAP
 PORT Sda = xps_iic_3_Sda
 PORT Scl = xps_iic_3_Scl
 PORT IIC2INTC_Irpt = xps_iic_3_IIC2INTC_Irpt
END
```

Appendix B

SEU Controller Connection Schematic



Appendix C

SEU Screening of Spartan-6 FPGA Using Cf-252

C.1 Introduction

Building satellites for scientific missions at Universities requires the use of low cost and high performance components to maintain the mission budget and performance. This requirement drives the researchers to make use of the available state-of-art COTS components in developing satellite subsystems. COTS have higher performance and extremely low cost when compared to the space approved components however they are not guaranteed for use in space applications. Therefore, researchers are working on the development of new design techniques to make COTS components suitable for use in space. At Kyushu Institute of Technology in Japan, we have been carrying research on the fitness of commercial grade SRAM based FPGAs for use in scientific satellite missions. SRAM based FPGAs are widely used in developing digital systems for different applications. However, their use in space applications is still under investigation due to three reasons: 1) their configuration bitstream can be corrupted by SEUs, 2) they consume high power and 3) their cost is high when compared to discrete components. FPGA manufacturers managed to solve the problems of power consumption and cost. They introduced low cost and low power consumption FPGAs such as the Spartan-6 with 45nm technology from Xilinx. To solve the problem of corrupting the configuration memory by the space radiation, Xilinx developed a technique named DPR. In that technique, the configuration memory can be read and checked for errors in a frame by frame basis. If a single bit error is found then the frame would be corrected and written back to the configuration memory. Xilinx introduced an IP core, the Single Event Mitigation (SEM) core, to scrub the configuration memory of its devices using the DPR technique. The purpose of this research is to evaluate the performance of using the SEM with DPR in mitigating soft errors in the configuration memory of the Sparatn-6 FPGA using Cf-252. The radiation tests were carried out at the Kyoto University/Research Reactor Institute using a decapsulated Sapatn-6 FPGA. The use of Cf-252 in assessing the radiation effects is easier and has lower cost when compared to the proton beam and heavy ion tests. Nevertheless, when using Cf-252 we cannot control the flux and energy of the radiated ions. Therefore, the Cf-252 is suitable to give an estimate of the design and fabric performance under radiation without having detailed accurate measurements of the dynamic and static cross sections as in the proton and heavy ion tests. In fact for developing low cost missions with acceptable reliability, Cf-252 test results are adequate. Cf-252 is a very suitable alternative to the expensive and complicated variant of proton and heavy ion tests. We do believe that if more research is carried on qualifying COTS SRAM based FPGAs for use in space then more advanced missions would be launched and more capabilities would be acquired. In the next sections we present the tested system design, the test setup, the test results and the conclusion.

C.2 System Design and Test Setup

An embedded processor system was developed using the EDK package from Xilinx.. Figure 1 shows the architecture and external interface of the used design during the Cf-252 tests. The system consisted of a MicroBlaze processor interfaced to an Interrupt Controller (INTC), Timer, WDT and Reset Module through the PLB. The MicroBlaze run the code form its Block RAM (BRAM). MDM IP core was used for debugging through JTAG interface. Xilinx General Peripheral Input/Output (XGPIO) core was used for external Input/Output from/to status leds and switches. A UART is used for data logging from the MicroBlaze to an external PC. The SEM core continuously scrubbed the configuration memory which contained 11,939,296 configuration bits [1]. It sent the results of scrubbing indicating any single or multiple bit upsets found in the entire device bitstream to the external data logging PC. During the test, software ISR was initiated based on the interrupt received from the WDT via the INTC. The WDT issued the interrupt whenever it reached its time out value (every 5.34 sec). Upon reception of the interrupt and initiation of the ISR, the MicroBalze reset the WDT. If the MicroBlaze was in a hang up situation then the WDT would not be reset. At that moment the WDT sends a reset signal to the reset module to reset the whole system. An external WDT with longer time out interval (every 15sec) resets and reconfigures the whole FPGA whenever the MicroBlaze fails to restart after the internal WDT reset. The Sparatn-6 was decapsualted as shown in figure 2. During the test, The Cf-252 was placed at different distances from the FPGA top. The number of system failures and configuration frames bit upsets were recorded. The test was conducted in two cases: with enabling the SEM core and without enabling the SEM core. Static cross section evaluation was done at 0.5 cm and 1cm distances. System Failure rate tests were conducted at three distances: 1 cm, 2 cm and 3 cm. Each test was run for 20 min. All the tests were carried out in a vacuumed chamber.

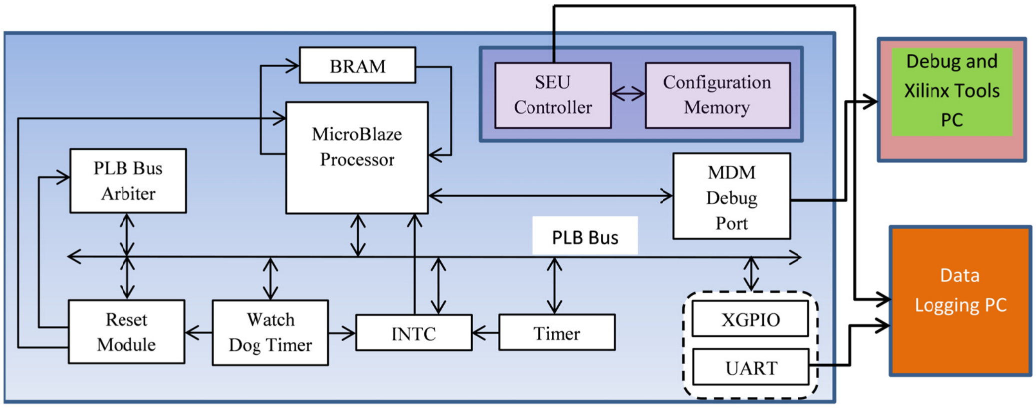


Figure 1 Design Under Test Architecture and External Interfaces

C.3 Results and Discussion

The SEU and MBU count during 20 min of tests at 1cm, 2 cm and 3 cm distances between the Cf-252 and the FPGA top is shown in figure 3. The Cf-252 has a mean LET of 43 (MeV cm^2/mg). The flux is about 22 ions/cm^2/sec [2][3]. The energy changes with the distance from the FPGA. The total number of upsets decreases as the distance increases. MBU ratio to the total upsets decreases as the distance increases because the stopping energy causing the adjacent bit upsets reduces as the distance to the FPGA silicon increases.

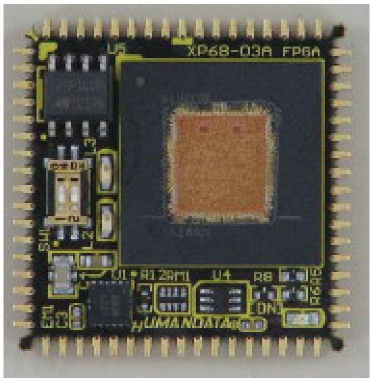


Figure 2 Decapsulated Spartan 6 FPGA.

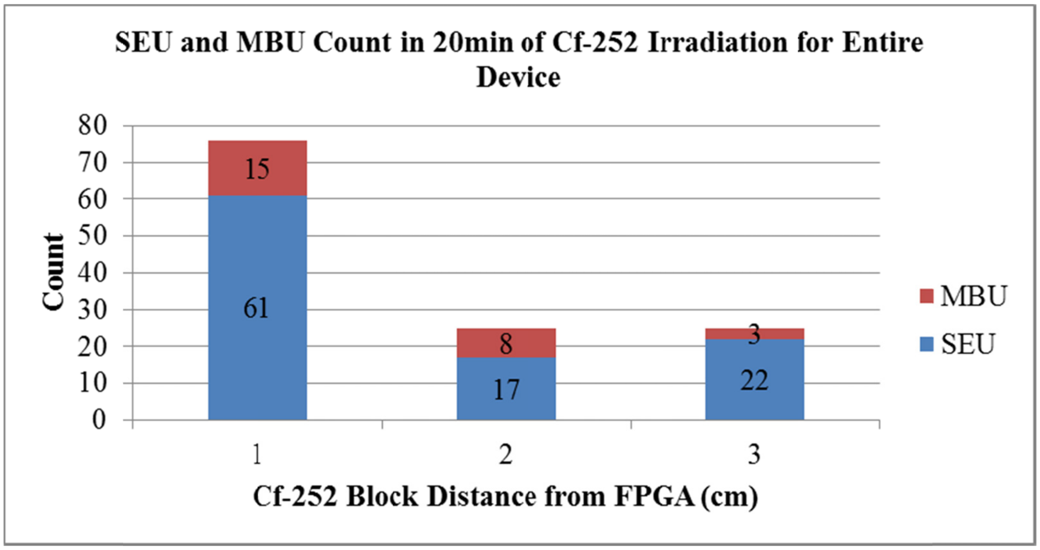


Figure 3 SEU and MBU count Vs. Distance.

To estimate the bit upset rate and static cross section for the entire device we consider that the device has 11,939,296 configuration bits and that the flux of the Cf-252 is 22 (ions/cm^2/sec).The MBUs are considered as double. We calculate the bit upsets according to Eq.1:

$$\text{Total Bit Upsets Count} = \text{MBU Count} \times 2 + \text{SEU Count} \qquad (1)$$

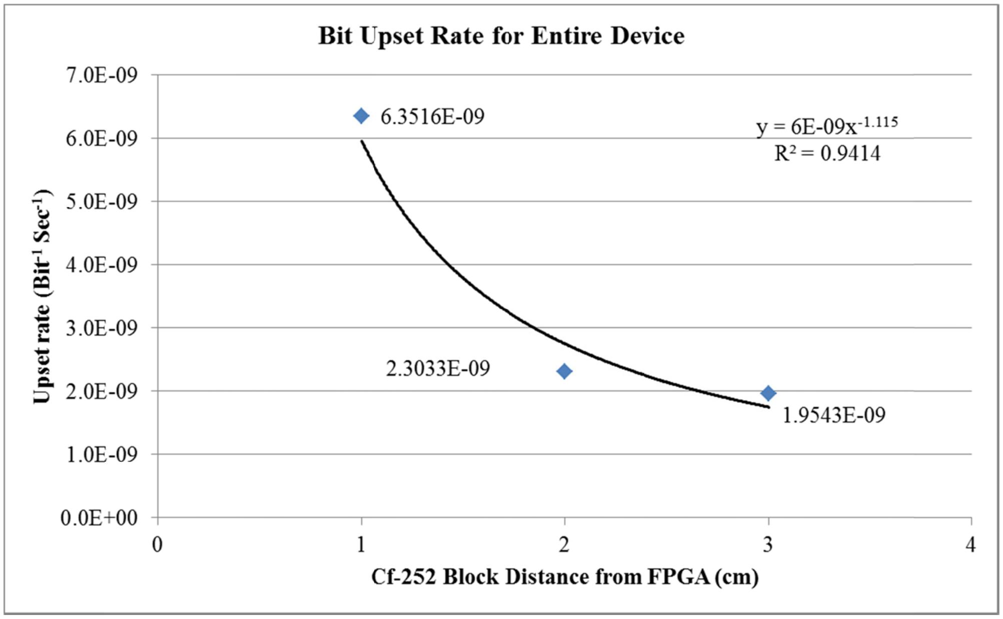


Figure 4 Bit Upset Rate for the Entire Spartan 6 Device.

The data is fit in figure 4using a power equation, the R-Squared value for the fit curve is about 0.94. The cross section is calculated from Eq.2 and shown in figure5 :

$$\sigma = \frac{number\ of\ upsets}{fluence\ (particle/cm^2)} \tag{2}$$

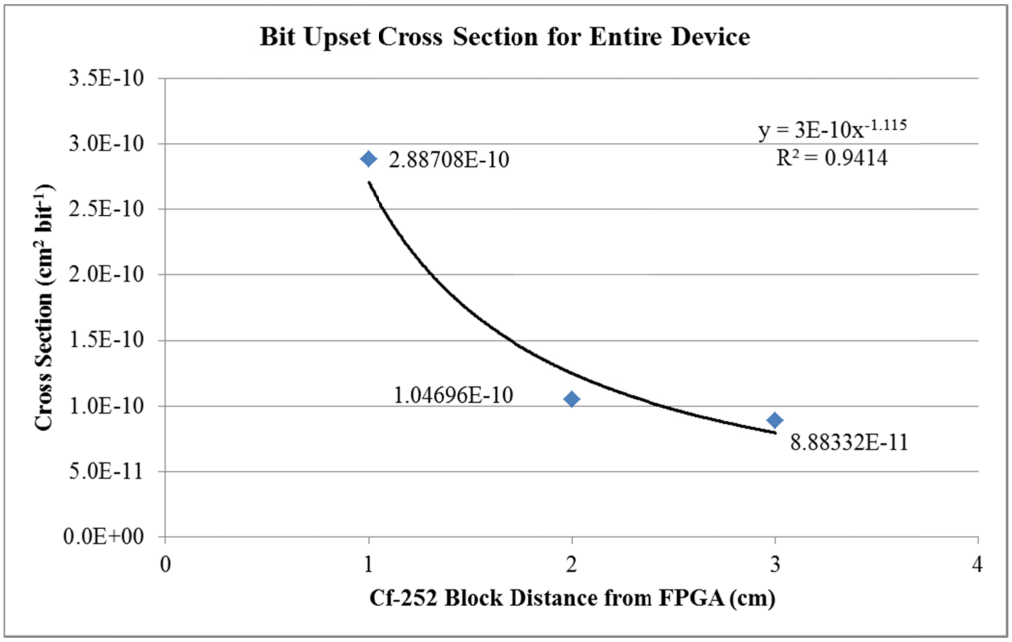


Figure 5 Bit Upset Cross Section for the Sparatn 6 Device.

The system failure rate was estimated based on full system failure and FPGA reloading. It is shown in two cases when the SEM core is enabled and when it is disabled. When the SEM core is used for scrubbing the configuration memory, the system failures were reduced by about 32% at 0.5 cm and 52.9% at 1cm. Actual failure rates depend on the selected orbit.

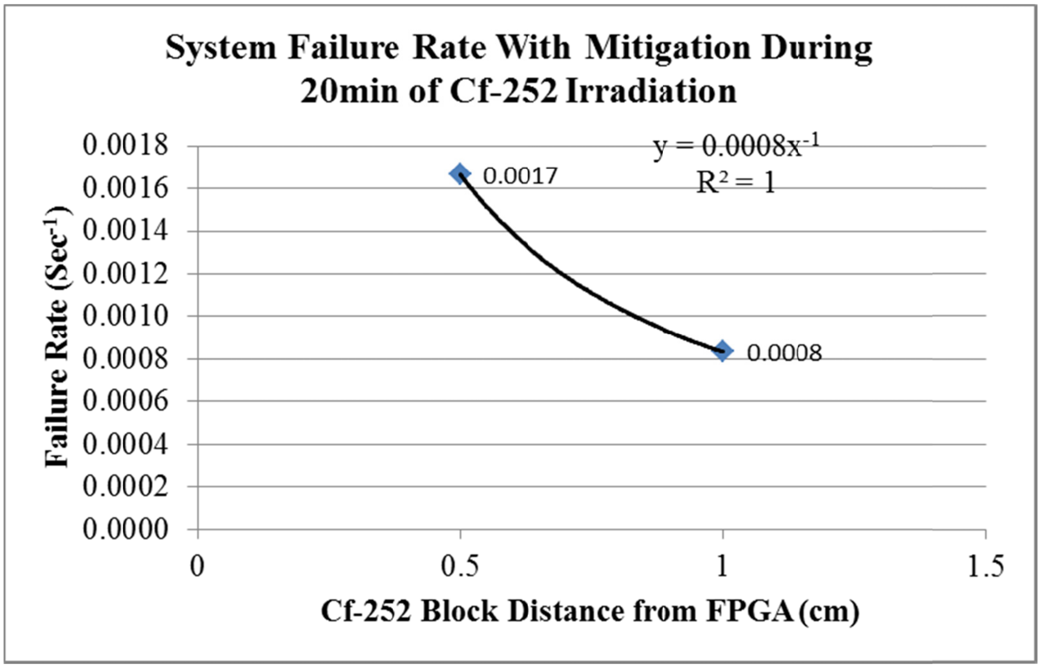


Figure 6 System Failure Rate with Mitigation.

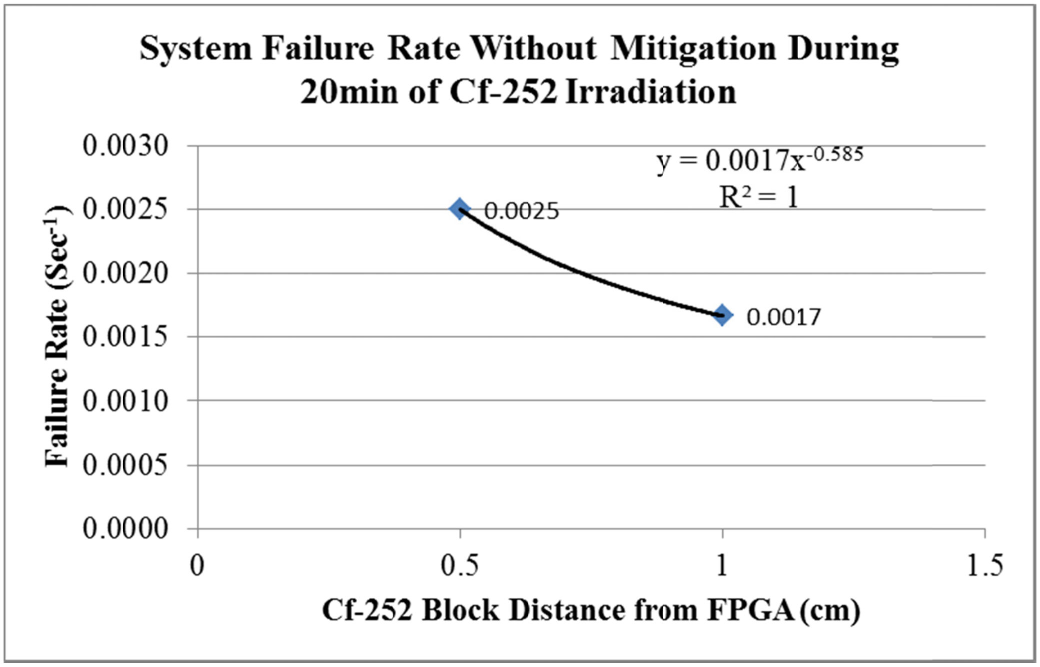


Figure 7 System Failure Rate without Mitigation.

C.4 Conclusion

Decapsulated Spartan 6 was tested using Cf-252 irradiation. The FPGA showed better performance in terms of reduced failure rates when SEM core was used. The cross section curve is presented.

References

[1] Xilinx, Spartan-6 FPGA Configuration User Guide, UG380 (v2.5) January 23, 2013.

[2] Gaisler Research, LEON3-FT-RTAX SEU test results, Issue 1, December 7, 2005.

[3] Stephen, J. H.; Sanderson, T. K.; Mapper, D.; Farren, J.; Harboe-Sorensen, R.; Adams, L., "A Comparison of Heavy Ion Sources Used in Cosmic Ray Simulation Studies of VLSI Circuits," Nuclear Science, IEEE Transactions on , vol.31, no.6, pp.1069,1072, Dec. 1984.

Appendix D

Sample Code – MPSoC Embedded Software in Thermal Vacuum and Radiation Tests

D.1 MicroBlaze – 0 Code

```
#include <stdio.h>
#include "xparameters.h"
#include "xuartlite.h"
#include "xuartlite_l.h"
#include "platform.h"
#include "fsl.h"
#include "xwdttb.h"
#include "xintc.h"
#include "xmbox.h"
#include "xtmrctr.h"



//comment this line to stop resetting the Microblaze through its WDT
//#define reset_WDT

void print(char *str);

XWdtTb WDT;
XIntc INTC;
XMbox MB0_MB3;
XMbox_Config CFG1;
XTmrCtr Timer;


union ACK_mes
{
    char ACK_char[4];
    unsigned long ACK_long;
};

union ACK_mes DATA;


//volatile unsigned long int i = 0;

#define reset_WDT
static unsigned long Time=0;

unsigned char Timer_start_flag=0;
unsigned long Timer_Value;


void WDT_INT_Handler(void* base_p)
{
```

```
//Interrupt Handling Routine, restart watch dog timer
    //unsigned int long j;

#ifdef reset_WDT
    *(&Time) = *(&Time) + 1;

    //xil_printf("\r\n WDT ISR started....resetting watch dog
timer...\r\n");

    XMbox_WriteBlocking(&MB0_MB3, &DATA.ACK_long, 4);


    if (Timer_start_flag==1)
        {
         XTmrCtr_Stop  (&Timer, 0);
         Timer_start_flag=0;
         Timer_Value = XTmrCtr_GetValue (&Timer, 0);
         xil_printf("\r\n Timer is Stopped... Value = %d \r\n",
Timer_Value);
        }
    if (Timer_start_flag==0)
    {
        Timer_start_flag=1;
        XTmrCtr_Reset (&Timer, 0);
        XTmrCtr_Start  (&Timer, 0);
        xil_printf("\r\n Timer is Started... \r\n");
    }


    xil_printf("\r\n MB0\t Time Step: %d\tStatus: OK \r\n", Time);
    xil_printf("\r\n ACk sent to MB3 \r\n");

    //for (j=0;j<100000;j++);


    XWdtTb_RestartWdt  (&WDT) ;
#else
        //if WDT interrupt is set in hardware as level then you will
see this message until system reset, make it edge to enter ISR once
        xil_printf("\r\n WDT ISR started....watch dog timer will not
be reset...\r\n");

        for (j=0;j<100000;j++);


#endif


    //  return;

}


int main()
{
    DATA.ACK_char[0] = 'A';
    DATA.ACK_char[1] = 'C';
    DATA.ACK_char[2] = 'K';
    DATA.ACK_char[3] = '1';

    init_platform();
    XUartLite Com_Port;

    //Initialize UART
    XUartLite_Initialize  ( &Com_Port,
XPAR_XPS_UARTLITE_0_DEVICE_ID);

    xil_printf("\r\n WELCOME to MB0\r\n");

    //Initialize Mailbox
     XMbox_CfgInitialize (&MB0_MB3, &CFG1,
XPAR_MAILBOX_MB0_MB3_IF_0_BASEADDR);


     //initialize counter
      XTmrCtr_Initialize ( &Timer, XPAR_XPS_TIMER_0_DEVICE_ID);

     //reset value
     XTmrCtr_SetResetValue  ( &Timer, 0,  0);


     //Initialize Watch Dog Timer

    XWdtTb_Initialize  (&WDT, XPAR_XPS_TIMEBASE_WDT_0_DEVICE_ID  );


     //check if there was a previous reset...
     if (XWdtTb_IsWdtExpired  (&WDT))
         xil_printf("\r\n MB0 was reset...\r\n");
     else
     xil_printf("\r\n System is power recycled...\r\n");

     xil_printf("\r\n MB0\t Time Step: %d\tStatus: OK \r\n", Time);

    /* Enable MicroBlaze interrupts */
    microblaze_enable_interrupts();

    /* Register Watch Dog Timer interrupt handler */
    XIntc_RegisterHandler(XPAR_XPS_INTC_0_BASEADDR ,
XPAR_XPS_INTC_0_XPS_TIMEBASE_WDT_0_WDT_INTERRUPT_INTR,
WDT_INT_Handler, (void*)XPAR_XPS_TIMEBASE_WDT_0_BASEADDR);

    /* Start the interrupt controller */
    XIntc_MasterEnable (XPAR_XPS_INTC_0_BASEADDR);

    // very important, if you remove it then interrupt will not be
caught....enable interrupt for the WDT

    XIntc_EnableIntr  (XPAR_XPS_INTC_0_BASEADDR,
XPAR_XPS_TIMEBASE_WDT_0_WDT_INTERRUPT_MASK);
```

```
////////////////////////////////////////////////////////////////////
////////////////////////////////////////

    //Start Watch Dog Timer

     XWdtTb_Start  (&WDT);

     for(;;)
     {
         //xil_printf("\r\n MB0 is working");
         //for (i = 0; i<1000000; i++);
     }

    cleanup_platform();

    return 0;
}
```

D.2 MIcroBlaze 1 – Code

```
#include <stdio.h>
#include "xparameters.h"
#include "xuartlite.h"
#include "xuartlite_l.h"
#include "platform.h"
#include "fsl.h"
#include "xwdttb.h"
#include "xintc.h"
#include "xmbox.h"
#include "xtmrctr.h"

//comment this line to stop resetting the Microblaze through its WDT
//#define reset_WDT

void print(char *str);

XWdtTb WDT;
XIntc INTC;
XMbox MB1_MB3;
XMbox_Config CFG1;
XTmrCtr Timer;

union ACK_mes
{
    char ACK_char[4];
    unsigned long ACK_long;
};

union ACK_mes DATA;
//volatile unsigned long int i = 0;

#define reset_WDT
static unsigned long Time=0;
```

```
unsigned char Timer_start_flag=0;
unsigned long Timer_Value;



void WDT_INT_Handler(void* base_p)
{

//Interrupt Handling Routine, restart watch dog timer
//  unsigned int long j;

#ifdef reset_WDT
    *(&Time) = *(&Time) + 1;
    //xil_printf("\r\n WDT ISR started....resetting watch dog
timer...\r\n");

    XMbox_WriteBlocking(&MB1_MB3, &DATA.ACK_long, 4);

    if (Timer_start_flag==1)
            {
             XTmrCtr_Stop  (&Timer, 0);
             Timer_start_flag=0;
             Timer_Value = XTmrCtr_GetValue (&Timer, 0);
             xil_printf("\r\n Timer is Stopped... Value = %d \r\n",
Timer_Value);
            }
        if (Timer_start_flag==0)
        {
            Timer_start_flag=1;
            XTmrCtr_Reset (&Timer, 0);
            XTmrCtr_Start  (&Timer, 0);
            xil_printf("\r\n Timer is Started... \r\n");
        }


    xil_printf("\r\n MB1\t Time Step: %d\tStatus: OK \r\n", Time);
    xil_printf("\r\n ACk sent to MB3 \r\n");
    //for (j=0;j<100000;j++);


    XWdtTb_RestartWdt  (&WDT) ;
#else
        //if WDT interrupt is set in hardware as level then you will
see this message until system reset, make it edge to enter ISR once
        xil_printf("\r\n WDT ISR started....watch dog timer will not
be reset...\r\n");

        for (j=0;j<100000;j++);


#endif



    //  return;

}
```

```
int main()
{
    DATA.ACK_char[0] = 'A';
    DATA.ACK_char[1] = 'C';
    DATA.ACK_char[2] = 'K';
    DATA.ACK_char[3] = '1';

    init_platform();
    XUartLite Com_Port;
    XUartLite_Initialize  ( &Com_Port,
XPAR_XPS_UARTLITE_1_DEVICE_ID);

    xil_printf("\r\n WELCOME to MB1\r\n");

    //Initialize Mailbox
     XMbox_CfgInitialize (&MB1_MB3, &CFG1,
XPAR_MAILBOX_MB1_MB3_IF_0_BASEADDR);

     //initialize counter
     XTmrCtr_Initialize ( &Timer, XPAR_XPS_TIMER_1_DEVICE_ID);

    //reset value
    XTmrCtr_SetResetValue  ( &Timer, 0,  0);

     //Initialize Watch Dog Timer

    XWdtTb_Initialize  (&WDT, XPAR_XPS_TIMEBASE_WDT_1_DEVICE_ID  );


     //check if there was a previous reset...
     if (XWdtTb_IsWdtExpired  (&WDT))
         xil_printf("\r\n MB1 was reset...\r\n");
     else
     xil_printf("\r\n System is power recycled...\r\n");
     xil_printf("\r\n MB1\t Time Step: %d\tStatus: OK \r\n", Time);

    /* Enable MicroBlaze interrupts */
    microblaze_enable_interrupts();

    /* Register Watch Dog Timer interrupt handler */
    XIntc_RegisterHandler(XPAR_XPS_INTC_1_BASEADDR ,
XPAR_XPS_INTC_1_XPS_TIMEBASE_WDT_1_WDT_INTERRUPT_INTR,
WDT_INT_Handler, (void*)XPAR_XPS_TIMEBASE_WDT_1_BASEADDR);

    /* Start the interrupt controller */
    XIntc_MasterEnable (XPAR_INTC_0_BASEADDR);

    // very important, if you remove it then interrupt will not be
caught....enable interrupt for the WDT

    XIntc_EnableIntr
(XPAR_XPS_INTC_1_BASEADDR,XPAR_XPS_TIMEBASE_WDT_1_WDT_INTERRUPT_MASK
);



////////////////////////////////////////////////////////////////////
////////////////////////////////////////
```

```
    //Start Watch Dog Timer

     XWdtTb_Start  (&WDT);

     for(;;)
     {
         //xil_printf("\r\n MB1 is working");
         //for (i = 0; i<1000000; i++);
     }

    cleanup_platform();

    return 0;
}
```

D.3 MicroBlaze 2 – Code

```
#include <stdio.h>
#include "xparameters.h"
#include "xuartlite.h"
#include "xuartlite_l.h"
#include "platform.h"
#include "fsl.h"
#include "xwdttb.h"
#include "xintc.h"
#include "xmbox.h"
#include "xtmrctr.h"


//comment this line to stop resetting the Microblaze through its WDT
//#define reset_WDT

void print(char *str);

XWdtTb WDT;
XIntc INTC;
XMbox MB2_MB3;
XMbox_Config CFG1;
XTmrCtr Timer;

union ACK_mes
{
    char ACK_char[4];
    unsigned long ACK_long;
};

union ACK_mes DATA;

//volatile unsigned long int i = 0;


#define reset_WDT
static unsigned long Time=0;
```

```
unsigned char Timer_start_flag=0;
unsigned long Timer_Value;



void WDT_INT_Handler(void* base_p)
{

//Interrupt Handling Routine, restart watch dog timer
//  unsigned int long j;

#ifdef reset_WDT
    *(&Time) = *(&Time) + 1;
    //xil_printf("\r\n WDT ISR started....resetting watch dog
timer...\r\n");

    XMbox_WriteBlocking(&MB2_MB3, &DATA.ACK_long, 4);

    if (Timer_start_flag==1)
                {
                 XTmrCtr_Stop  (&Timer, 0);
                 Timer_start_flag=0;
                 Timer_Value = XTmrCtr_GetValue (&Timer, 0);
                 xil_printf("\r\n Timer is Stopped... Value = %d
\r\n",  Timer_Value);
                }
            if (Timer_start_flag==0)
            {
                Timer_start_flag=1;
                XTmrCtr_Reset (&Timer, 0);
                XTmrCtr_Start  (&Timer, 0);
                xil_printf("\r\n Timer is Started... \r\n");
            }


    xil_printf("\r\n MB2\t Time Step: %d\tStatus: OK \r\n", Time);
    xil_printf("\r\n ACk sent to MB3 \r\n");
    //for (j=0;j<100000;j++);

    XWdtTb_RestartWdt  (&WDT) ;
#else
        //if WDT interrupt is set in hardware as level then you will
see this message until system reset, make it edge to enter ISR once
        xil_printf("\r\n WDT ISR started....watch dog timer will not
be reset...\r\n");

        for (j=0;j<100000;j++);


#endif


    //  return;

}
```

```
int main()
{
    DATA.ACK_char[0] = 'A';
    DATA.ACK_char[1] = 'C';
    DATA.ACK_char[2] = 'K';
    DATA.ACK_char[3] = '1';

    init_platform();
    XUartLite Com_Port;
    XUartLite_Initialize  ( &Com_Port,
XPAR_XPS_UARTLITE_2_DEVICE_ID);

    xil_printf("\r\n WELCOME to MB2\r\n");


    //Initialize Mailbox
    XMbox_CfgInitialize (&MB2_MB3, &CFG1,
XPAR_MAILBOX_MB2_MB3_IF_0_BASEADDR);

     //initialize counter
         XTmrCtr_Initialize ( &Timer, XPAR_XPS_TIMER_2_DEVICE_ID);

        //reset value
        XTmrCtr_SetResetValue  ( &Timer, 0,  0);


     //Initialize Watch Dog Timer

    XWdtTb_Initialize  (&WDT, XPAR_XPS_TIMEBASE_WDT_2_DEVICE_ID  );


     //check if there was a previous reset...
     if (XWdtTb_IsWdtExpired  (&WDT))
         xil_printf("\r\n MB2 was reset...\r\n");
     else
     xil_printf("\r\n System is power recycled...\r\n");
     xil_printf("\r\n MB2\t Time Step: %d\tStatus: OK \r\n", Time);

    /* Enable MicroBlaze interrupts */
    microblaze_enable_interrupts();

    /* Register Watch Dog Timer interrupt handler */
    XIntc_RegisterHandler(XPAR_XPS_INTC_2_BASEADDR ,
XPAR_XPS_INTC_2_XPS_TIMEBASE_WDT_2_WDT_INTERRUPT_INTR,
WDT_INT_Handler, (void*)XPAR_XPS_TIMEBASE_WDT_2_BASEADDR);

    /* Start the interrupt controller */
    XIntc_MasterEnable (XPAR_XPS_INTC_2_BASEADDR);

    // very important, if you remove it then interrupt will not be
caught....enable interrupt for the WDT

    XIntc_EnableIntr
(XPAR_XPS_INTC_2_BASEADDR,XPAR_XPS_TIMEBASE_WDT_2_WDT_INTERRUPT_MASK
);
////////////////////////////////////////////////////////////////////
////////////////////////////////////////

    //Start Watch Dog Timer

     XWdtTb_Start  (&WDT);

     for(;;)
     {

        // xil_printf("\r\n MB2 is working");
        // for (i = 0; i<1000000; i++);

     }

    cleanup_platform();

    return 0;
}
```

D.4 MicroBlaze 3 (MSCP) – Code

```
#include <stdio.h>
#include "xparameters.h"
#include "xuartlite.h"
#include "xuartlite_l.h"
#include "platform.h"
#include "fsl.h"
#include "xwdttb.h"
#include "xintc.h"
#include "xiic_l.h"
#include "xiic.h"
#include "xmbox.h"
#include "xgpio.h"
#include "MAX6656.h"
#include "xutil.h"

#define MEM_UNDER_TEST_BASE_ADDR  XPAR_EMC_0_MEM0_BASEADDR
#define print_MAX6656_messages
#define reset_WDT
#define thermal_data
#define Board_MRAM //This is for the 256K MRAM

//comment this line to stop resetting the Microblaze through its WDT
//#define reset_WDT

int initialize_MAX6656(void);
void print(char *str);

int STATUS;
XStatus status;
unsigned long Time=0;
```

```
int flag = 1;
int flag2 = 0;
int FPGA_temp;
int sensor_temp;
//volatile unsigned long int i = 0;

XIic IIC;
XWdtTb WDT;
XIntc INTC;
XUartLite Com_Port;
XMbox MB3_MB0;
XMbox_Config CFG0;
XMbox MB3_MB1;
XMbox_Config CFG1;
XMbox MB3_MB2;
XMbox_Config CFG2;
XGpio gpio;




union ACK_mes
{
    char ACK_char[4];
    unsigned long ACK_long;
};

union ACK_mes DATA;

unsigned long r0;
unsigned long r1;
unsigned long r2;

unsigned long WDI;

u8 TEMP_SENSOR_ADDRESS = 0x18; //originally it is 0x30 but alligned
for 7 bit addressing


void WDT_INT_Handler(void* base_p)
{

    //Interrupt Handling Routine, restart watch dog timer
    //  unsigned int long j;

#ifdef reset_WDT
    //*(&Time) = *(&Time) + 1;
    Time++;
    //xil_printf("\r\n WDT ISR started....resetting watch dog
timer...\r\n");

    XMbox_ReadBlocking(&MB3_MB0, &r0, 4);
    XMbox_ReadBlocking(&MB3_MB1, &r1, 4);
    XMbox_ReadBlocking(&MB3_MB2, &r2, 4);

    WDI = WDI^0x80000000;
    XGpio_DiscreteWrite (&gpio,1,WDI);
    flag = 1;

    if ((r0 == DATA.ACK_long) && (r1 == DATA.ACK_long) && (r2 ==
DATA.ACK_long))
    {
        XWdtTb_RestartWdt  (&WDT) ;
        flag2 = 1;
    }


    //for (j=0;j<100000;j++);


////////////////////////////////////////////////////////////////////
//////////////////////////
        xil_printf("\r\n Starting MemoryTest for EMC_MRAM:\r\n");

        xil_printf("\r\n Running 32-bit test...\r\n");
        status =
XUtil_MemoryTest32((Xuint32*)MEM_UNDER_TEST_BASE_ADDR, 65536,
0xAAAA5555, XUT_ALLMEMTESTS);
        if (status == XST_SUCCESS)
        {
           print("\r\n PASSED!\r\n");
        }
        else
        {
           print("\r\n FAILED!\r\n");
        }
        xil_printf("\r\n Running 16-bit MRAM test...\r\n");
        status =
XUtil_MemoryTest16((Xuint16*)MEM_UNDER_TEST_BASE_ADDR, 131072,
0xAA55, XUT_ALLMEMTESTS);
        if (status == XST_SUCCESS)
        {
           xil_printf("\r\n PASSED!\r\n");
        }
        else
        {
           xil_printf("\r\n FAILED!\r\n");
        }

////////////////////////////////////////////////////////////////////
/////////////////////////


#else
    //if WDT interrupt is set in hardware as level then you will see
this message until system reset, make it edge to enter ISR once
    xil_printf("\r\n WDT ISR started....watch dog timer will not be
reset...\r\n");

    for (j=0;j<100000;j++);


#endif


    //  return;
```

```
}


int main()
{

    DATA.ACK_char[0] = 'A';
    DATA.ACK_char[1] = 'C';
    DATA.ACK_char[2] = 'K';
    DATA.ACK_char[3] = '1';

    u8 temp_u8;
    int temp_int;


    u8 reg_add;
    //unsigned long i;

    init_platform();
    XUartLite Com_Port;
    XUartLite_Initialize  ( &Com_Port,
XPAR_XPS_UARTLITE_3_DEVICE_ID);

    //Initialize Mailbox
    XMbox_CfgInitialize (&MB3_MB0, &CFG0,
XPAR_MAILBOX_MB0_MB3_IF_1_BASEADDR);
    XMbox_CfgInitialize (&MB3_MB1, &CFG1,
XPAR_MAILBOX_MB1_MB3_IF_1_BASEADDR);
    XMbox_CfgInitialize (&MB3_MB2, &CFG2,
XPAR_MAILBOX_MB2_MB3_IF_1_BASEADDR);

    //Initialize XGpio
    XGpio_Initialize(&gpio, XPAR_XPS_GPIO_3_DEVICE_ID);
    XGpio_SetDataDirection  (&gpio,1, 0x55ff5500);
    WDI = 0x80000000;
    XGpio_DiscreteWrite (&gpio,1,WDI);

    //initialize IIC and MAX6656
    XIic_Initialize(&IIC,XPAR_INTC_0_DEVICE_ID);
    XIic_Start(&IIC);
    // if the MAX6656 is not working then this will lead to stopping
the system!!
    //comment it to get the system to work...
    initialize_MAX6656();


    xil_printf("\r\n WELCOME to MB3\r\n");



    //Initialize Watch Dog Timer

    XWdtTb_Initialize  (&WDT, XPAR_XPS_TIMEBASE_WDT_3_DEVICE_ID  );


    //check if there was a previous reset...
    if (XWdtTb_IsWdtExpired  (&WDT))
```

```
        xil_printf("\r\n MB3 was reset...\r\n");
    else
        xil_printf("\r\n System is power recycled...\r\n");

    //xil_printf("\r\n MB3\t Time Step: %d\tStatus: OK FPGA
Temp: %d\tSensor Temp: %d\r\n", Time, FPGA_temp, sensor_temp);

    /* Enable MicroBlaze interrupts */
    microblaze_enable_interrupts();

    /* Register Watch Dog Timer interrupt handler */
    XIntc_RegisterHandler(XPAR_XPS_INTC_3_BASEADDR ,
XPAR_XPS_INTC_3_XPS_TIMEBASE_WDT_3_WDT_INTERRUPT_INTR,
WDT_INT_Handler, (void*)XPAR_XPS_TIMEBASE_WDT_3_BASEADDR);

    /* Start the interrupt controller */
    XIntc_MasterEnable (XPAR_XPS_INTC_3_BASEADDR);

    // very important, if you remove it then interrupt will not be
caught....enable interrupt for the WDT

    XIntc_EnableIntr
(XPAR_XPS_INTC_3_BASEADDR,XPAR_XPS_TIMEBASE_WDT_3_WDT_INTERRUPT_MASK
);




////////////////////////////////////////////////////////////////////
////////////////////////////////////////


    //Start Watch Dog Timer

    XWdtTb_Start  (&WDT);


    for(;;)
    {

        //reading external temperature
        //reading the unsigned byte part
        if (flag == 1)
        {
            reg_add = (u8) RRTE;

            XIic_DynSend(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS,
&reg_add, 1, XIIC_REPEATED_START);

            XIic_DynRecv(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS,
&temp_u8, 1);

            temp_int = (temp_u8 & 0x80)*(-1);

            temp_int = temp_int + temp_u8;
            //I disabled the fraction measurement part as it leads
to stopping the loop


            FPGA_temp = temp_int;

#ifdef thermal_test
            xil_printf("\r\n MB3\t Time Step: %d\tStatus: OK\tFPGA
Temp: %d\t  WDI: %d\r\n", Time, FPGA_temp, WDI>>31);
#endif

#ifdef thermal_data
            //to display the values read we have to switch on the
system from the board I made...
            xil_printf("\r\n%d\t%d\t%d", Time, FPGA_temp, WDI>>31);
#endif
            if (flag2 == 1)
                                    {
#ifdef thermal_test
                xil_printf("\r\n ACK received from all processors
\r\n");
#endif
                flag2 = 0;
                                    }



            flag = 0;
        } //if



    } //for
    cleanup_platform();

    return 0;
}





////////////////////////////////////////////////////////////////////
///


int initialize_MAX6656(void)
{

    u8 tx_data[2]; //register address[0] and transmit data [1]
    u8 rx_data;

    //write Data to conversion rate register

    tx_data[0] = WCRW;
    tx_data[1] = 0x01;  //measurements every 0.5 sec


    XIic_DynSend(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, tx_data,
2, XIIC_REPEATED_START);
```

```
    //readback to verify

    //read Data from conversion rate register
    tx_data[0] = RCRA;

    XIic_DynSend(XPAR_IIC_0_BASEADDR,  TEMP_SENSOR_ADDRESS, tx_data,
1, XIIC_REPEATED_START);

    XIic_DynRecv(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, &rx_data ,
1);

#ifdef print_MAX6656_messages
    xil_printf("\r\n Value written to register address [%x] is
[%x] ... value read from register address [%x] is [%x] \r\n", WCRW,
tx_data[1], RCRA,rx_data);
#endif
    if (rx_data != tx_data[1])
    {
#ifdef print_MAX6656_messages
        xil_printf("\r\n Readback error: Value written to register
address [%x] is different from value read \r\n", tx_data[0]);
#endif
        return(-3);
    }


    //write Data to internal ALERT high limit

    tx_data[0] = WLHO;
    tx_data[1] = 0x55;  //+80 C

    XIic_DynSend(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, tx_data,
2, XIIC_REPEATED_START);

    //readback to verify

    //read Data from internal ALERT high limit
    tx_data[0] = RLHN;

    XIic_DynSend(XPAR_IIC_0_BASEADDR,  TEMP_SENSOR_ADDRESS, tx_data,
1, XIIC_REPEATED_START);

    XIic_DynRecv(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, &rx_data ,
1);

#ifdef print_MAX6656_messages
    xil_printf("\r\n Value written to register address [%x] is
[%x] ... value read from register address [%x] is [%x] \r\n", WLHO,
tx_data[1], RLHN,rx_data);
#endif
    if (rx_data != tx_data[1])
    {
#ifdef print_MAX6656_messages
        xil_printf("\r\n Readback error: Value written to register
address [%x] is different from value read \r\n", tx_data[0]);
#endif
        return(-4);
    }
    //write Data to internal ALERT low limit
    tx_data[0] = WLLM;
    tx_data[1] = 0xEC;  //-20 C

    XIic_DynSend(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, tx_data,
2, XIIC_REPEATED_START);

    //readback to verify

    //read Data from internal ALERT low limit
    tx_data[0] = RLLI;

    XIic_DynSend(XPAR_IIC_0_BASEADDR,  TEMP_SENSOR_ADDRESS, tx_data,
1, XIIC_REPEATED_START);

    XIic_DynRecv(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, &rx_data ,
1);

#ifdef print_MAX6656_messages
    xil_printf("\r\n Value written to register address [%x] is
[%x] ... value read from register address [%x] is [%x] \r\n", WLLM,
tx_data[1], RLLI,rx_data);
#endif

    if (rx_data != tx_data[1])
    {
#ifdef print_MAX6656_messages
        xil_printf("\r\n Readback error: Value written to register
address [%x] is different from value read \r\n", tx_data[0]);
#endif
        return(-5);
    }


    //write Data to external ALERT high limit
    tx_data[0] = WRHA;
    tx_data[1] = 0x55;  //+80 C

    XIic_DynSend(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, tx_data,
2, XIIC_REPEATED_START);

    //readback to verify

    //read Data from external ALERT high limit
    tx_data[0] = RRHI;

    XIic_DynSend(XPAR_IIC_0_BASEADDR,  TEMP_SENSOR_ADDRESS, tx_data,
1, XIIC_REPEATED_START);

    XIic_DynRecv(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, &rx_data ,
1);

#ifdef print_MAX6656_messages
    xil_printf("\r\n Value written to register address [%x] is
[%x] ... value read from register address [%x] is [%x] \r\n", WRHA,
tx_data[1], RRHI,rx_data);
#endif
```

```
    if (rx_data != tx_data[1])
    {
#ifdef print_MAX6656_messages
        xil_printf("\r\n Readback error: Value written to register
address [%x] is different from value read \r\n", tx_data[0]);
#endif
        return(-6);
    }

    //write Data to external ALERT low limit
    tx_data[0] = WRLN;
    tx_data[1] = 0xEC;  //-20 C

    XIic_DynSend(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, tx_data,
2, XIIC_REPEATED_START);

    //readback to verify

    //read Data from external ALERT low limit
    tx_data[0] = RRLS;

    XIic_DynSend(XPAR_IIC_0_BASEADDR,  TEMP_SENSOR_ADDRESS, tx_data,
1, XIIC_REPEATED_START);

    XIic_DynRecv(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, &rx_data ,
1);

#ifdef print_MAX6656_messages
    xil_printf("\r\n Value written to register address [%x] is
[%x] ... value read from register address [%x] is [%x] \r\n", WRLN,
tx_data[1], RRLS,rx_data);
#endif
    if (rx_data != tx_data[1])
    {
#ifdef print_MAX6656_messages
        xil_printf("\r\n Readback error: Value written to register
address [%x] is different from value read \r\n", tx_data[0]);
#endif
        return(-7);
    }


    //write Data to internal OVERT limit
    tx_data[0] = WLOL;
    tx_data[1] = 0x64;  //+100 C

    XIic_DynSend(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, tx_data,
2, XIIC_REPEATED_START);

    //readback to verify

    //read Data from internal OVERT limit
    tx_data[0] = RLOL;

    XIic_DynSend(XPAR_IIC_0_BASEADDR,  TEMP_SENSOR_ADDRESS, tx_data,
1, XIIC_REPEATED_START);

    XIic_DynRecv(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, &rx_data ,
1);
```

```
#ifdef print_MAX6656_messages
    xil_printf("\r\n Value written to register address [%x] is
[%x] ... value read from register address [%x] is [%x] \r\n", WLOL,
tx_data[1], RLOL,rx_data);
#endif

    if (rx_data != tx_data[1])
    {
#ifdef print_MAX6656_messages
        xil_printf("\r\n Readback error: Value written to register
address [%x] is different from value read \r\n", tx_data[0]);
#endif
        return(-8);
    }

    //write Data to external OVERT limit
    tx_data[0] = WROL1;
    tx_data[1] = 0x64;  //+100 C

    XIic_DynSend(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, tx_data,
2, XIIC_REPEATED_START);

    //readback to verify

    //read Data from external OVERT limit
    tx_data[0] = RLOL1;

    XIic_DynSend(XPAR_IIC_0_BASEADDR,  TEMP_SENSOR_ADDRESS, tx_data,
1, XIIC_REPEATED_START);

    XIic_DynRecv(XPAR_IIC_0_BASEADDR, TEMP_SENSOR_ADDRESS, &rx_data ,
1);

#ifdef print_MAX6656_messages
    xil_printf("\r\n Value written to register address [%x] is
[%x] ... value read from register address [%x] is [%x] \r\n", WROL1,
tx_data[1], RLOL1,rx_data);
#endif

    if (rx_data != tx_data[1])
    {
#ifdef print_MAX6656_messages
        xil_printf("\r\n Readback error: Value written to register
address [%x] is different from value read \r\n", tx_data[0]);
#endif
        return(-9);
    }

    xil_printf("\r\n CONFIGURATION ENDED SUCCESSFULLY \r\n");
    return 1;
}
```

D.5 Temperature IC Addresses

```
/*
 * MAX6656.h
 *
 *  Created on: Oct 16, 2013
 *      Author: eslab
 */

#ifndef MAX6656_H_
#define MAX6656_H_


#endif /* MAX6656_H_ */

/*
 * MAX6656.h
 *
 *  Created on: Sep 6, 2013
 *      Author: eslab
 */

#ifndef MAX6656_H_
#define MAX6656_H_


#endif /* MAX6656_H_ */

#define RLTS 0x00 //0000 0000 Read Internal Temperature
#define RRTE 0x01 //0000 0000 Read External Temperature 1
#define RSL 0x02 //0000 0000 Read Status Byte; Note 1
#define RCL 0x03 //0000 0000 Read Configuration Byte
#define RCRA 0x04 //0000 0010 Read Conversion Rate Byte
#define RLHN 0x05 //0111 1111 Read Internal ALERT High Limit
#define RLLI 0x06 //1100 1001 Read Internal ALERT Low Limit
#define RRHI 0x07 //0111 1111 Read External Temperature 1 ALERT High
Limit
#define RRLS 0x08 //1100 1001 Read External Temperature 1 ALERT Low
Limit
#define WCA 0x09 //N/A Write Configuration Byte
#define WCRW 0x0A //N/A Write Conversion Rate Control Byte
#define WLHO 0x0B //N/A Write Internal ALERT High Limit
#define WLLM 0x0C //N/A Write Internal ALERT Low Limit
#define WRHA 0x0D //N/A Write External Temperature 1 ALERT High
Limit
#define WRLN 0x0E //N/A Write External Temperature 1 ALERT Low Limit
#define RRET1 0x10 //0000 0000 Read External 1 Extended Temperature
#define RRET2 0x11 //0000 0000 Read External 2 Extended Temperature
#define RLET 0x12 //0000 0000 Read Internal Extended Temperature
#define RRT2 0x13 //0000 0000 Read External Temperature 2
#define RRHL2 0x14 //0111 1111 Read External Temperature 2 ALERT
High Limit
#define RRLL2 0x15 //1100 1001 Read External Temperature 2 ALERT Low
Limit
#define RLOL 0x16 //0111 1111 Read Internal OVERT Limit
#define RLOL1 0x17 //0111 1111 Read External 1 OVERT Limit
#define RLOL2 0x18 //0111 1111 Read External 2 OVERT Limit
#define WLOL 0x19 //N/A Write Internal OVERT Limit
#define WROL1 0x1A //N/A Write External 1 OVERT Limit
#define WROL2 0x1B //N/A Write External 2 OVERT Limit
```

```
#define WRH2 0x1C //N/A Write External Temperature 2 ALERT High
Limit
#define WRL2 0x1D //N/A Write External Temperature 2 ALERT Low Limit
#define WV0HL 0x1E //N/A Write VCC (VIN2) ALERT High Limit for
MAX6655 (MAX6656)
#define WV0LL 0x1F //N/A Write VCC (VIN2) ALERT Low Limit for
MAX6655 (MAX6656)
#define WV1HL 0x20 //N/A Write VIN1 ALERT High Limit
#define WV1LL 0x21 //N/A Write VIN1 ALERT Low Limit
#define WV2HL 0x22 //N/A Write VIN2 (VCC) ALERT High Limit for
MAX6655 (MAX6656)
#define WV2LL 0x23 //N/A Write VIN2 (VCC) ALERT Low Limit for
MAX6655 (MAX6656)
#define WV3HL 0x24 //N/A Write VIN3 ALERT High Limit
#define WV3LL 0x25 //N/A Write VIN3 ALERT Low Limit
#define RV0HL 0x26 //1101 0011 Read VCC (VIN2) ALERT High Limit for
MAX6655 (MAX6656)
#define RV0LL 0x27 //1010 1101 Read VCC (VIN2) ALERT Low Limit for
MAX6655 (MAX6656)
#define RV1HL 0x28 //1101 0011 Read VIN1 ALERT High Limit
#define RV1LL 0x29 //1010 1101 Read VIN1 ALERT Low Limit
#define RV2HL 0x2A //1101 0011 Read VIN2 (VCC) ALERT High Limit for
MAX6655 (MAX6656)
#define RV2LL 0x2B //1010 1101 Read VIN2 (VCC) ALERT Low Limit for
MAX6655 (MAX6656)
#define RV3HL 0x2C //1101 0011 Read VIN3 ALERT High Limit
#define RV3LL 0x2D //1010 1101 Read VIN3 ALERT Low Limit
#define RV0 0x2E //0000 0000 Read VCC (VIN2) for MAX6655 (MAX6656)
#define RV1 0x2F //0000 0000 Read VIN1
#define RV2 0x30 //0000 0000 Read VIN2 (VCC) for MAX6655 (MAX6656)
#define RV3 0x31 //0000 0000 Read VIN3
#define RSL2 0x32 //0000 0000 Read Status Byte 2
#define RCL2 0x33 //0000 0000 Read Configuration Byte 2
#define WCA2 0x34 //N/A Write Configuration Byte 2
#define RDID 0xFE //0000 1010 Read Device ID
#define RDRV 0xFF //0100 1101 Read Manufacture ID
```

Appendix E

Frame Addresses Generation in the Virtex-5 FPGA

```
%this file is used to query all the configuration frame addresses of
the
%Virtex5Lx50. It depends on issuing the (q)command to the embedded
SEU
%controller macro for each of the(21d6)frames and receiving the
%corresponding 6 digit hex address. The linear addresses together
with the
%converted addresses are stored in a file for later retrieval by the
random
%fault injector.

%file for storing the addresses after conversion

fid = fopen('C:\Users\mohamed
mahmoud\Documents\MATLAB\SEU_MODEL\frame_add.txt', 'w');

%Serial Port Settings
s1 = serial('COM2');

%Serial Port specific properties:
s1.BaudRate = 115200;
s1.DataBits = 8;
s1.DataTerminalReady = 'off';
s1.FlowControl = 'none';
s1.Parity = 'none';
s1.ReadAsyncMode = 'continuous';
s1.StopBits = 1;
s1.Terminator = 'CR';

% Serial Port (additional) Properties
s1.ByteOrder = 'littleEndian';
%s1.BytesAvailableFcn = {'dispcallback'};
%s1.BytesAvailableFcnCount = 16;
%s1.BytesAvailableFcnMode = 'bytes';
%s1.ErrorFcn =
s1.InputBufferSize = 512;
s1.ObjectVisibility = 'on';
s1.OutputBufferSize = 512;
%s1.OutputEmptyFcn =
%s1.RecordDetail = 'compact';
%s1.RecordMode = 'overwrite';
%s1.RecordName = 'record.txt';
%s1.RecordStatus = 'off';

s1.Timeout = 10;
%s1.TimerFcn =
%s1.TimerPeriod = 1;

%response to * command when sent for second time
res_ast_2 = [42;13;63;13;13;62;13;13;63;13;13;62];

total_no_of_frames = 8663; %21D7h
```

```
%frame numbers vector
Frame_No_Hex_Converter = zeros(total_no_of_frames-1,1);

fopen(s1);
%normally input and output buffers are flushed after fopen, flushing
here
%is for more robustness.

flushinput(s1);
flushoutput(s1);

%Set the system in UART control mode

fprintf(s1,'*');
pause(0.5);
flushinput(s1);

fprintf(s1,'*');
pause(0.5);
len = s1.bytesavailable;
input_buffer = fread(s1,len);

count=0;
for i = 1:len

    if(input_buffer(i) == res_ast_2(i))
        count = count +1;
    end
end

if(count == 12)
    fprintf('response received for Asterik command\r\n');
else
    fprintf('Error - incorrect response to Asterik command\r\n');
    exit();
end

%flush input buffer for receiving new data, it should be clean even
without
%it.
flushinput(s1);
flushoutput(s1);

%query of frame numbers
%for j = 2602:total_no_of_frames-1, I made like that because it
stopped
%working in the middle of operation at a20 hex, which is 2601
decimal so I
%decided to resume from the next frame in order not to waste time.

for j = 1:total_no_of_frames-1
    %convert the frame index into hex then string then concatenate
it to
    %the q command then send it to the SEU controller
    frame_no_hex = dec2hex(j,4);
    frame_no_str = num2str(frame_no_hex,'%x');

    %same as the above two lines
    %frame_no_str = num2str(frame_no_hex,'%.4x')
    q_command = strcat('q',frame_no_str);

    %flush output buffer for safety, it should be clean even without
it.
    flushoutput(s1);
    fprintf(s1, q_command);
    pause(0.5);
    %read response to issuing the q-command
    q_command_response = fread(s1,s1.bytesavailable);

    %read the address and convert it into characters, hex string
    frame_add_hex = sscanf(char(q_command_response(21:26)),'%x');

    %store it in the vector that will be indexed later to retrive
the
    %converted frame addresses.
    Frame_No_Hex_Converter(j) =  frame_add_hex;

    %write data in the text file
    %important to keep the format specifiers for the width of the
printed
    %word to be 4 and 6 as this is needed by the controller, 6
digits of
    %hex values corresponding to 24 bits address.
    fprintf(fid,'%.4x\t%.6x\r\n',j,Frame_No_Hex_Converter(j));
    fprintf('%.4x\t%.6x\r\n',j,Frame_No_Hex_Converter(j));
    flushinput(s1);
end



%close file
fclose(fid);
%close, delete, clear object
fclose(s1);
delete(s1);
clear s1;
```

Appendix F

Sample Code - Fault Injection of Single Bit Upsets

```
%random seed

RandStream.setDefaultStream(RandStream('mt19937ar','seed',sum(100*cl
ock)));

%file to read the addresses of FPGA frames.

fid = fopen('C:\Users\mohamed
mahmoud\Documents\MATLAB\SEU_MODEL\frame_add.txt','r');

%constants

no_of_SEU = 100;
test_cycles = 10;
total_no_of_frames = 8663;%this is 21D7 in hex, but we can generate
from 1 to 21D6 as in the SEU controller data. frame 0 is unused.
total_no_of_bits = 1312;%we can generate from 0 up to 1311 this is
in Dec, in hex it is from 0 till 51F.

%response to * command when sent for second time
res_ast_2 = [42;13;63;13;13;62;13;13;63;13;13;62];

%response to 'd' command
d_command_res = [100;13;68;79;77;13;13;62;13;13;63;13;13;62];

%read the farmes addresses from the addresses file

frames_addresses = fscanf(fid,'%4c%*c%6c\r\n',[10 8662]);%8663-1 =
8662 frames. frame numbers start from 1 till 21D6h.reading is
%in column order that is why we swap columns and rows indeices then
we
%transpoase the resulting array
frames_addresses = frames_addresses';
%when accessing the farmes_addresses array to get the farme hex
address we
%can have to index it with the frame linear address then obtain the
hex
%address from column 5 till 10, i.e. frame_addresses(8662, 5:10) =
710100

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%


%1- start serial communication to put the SEU controller in the uart
mode
%and start commanding it

%Serial Port Settings
%serial port can be different than com2
s1 = serial('COM1');
```

```
%Serial Port specific properties:
s1.BaudRate = 115200;
s1.DataBits = 8;
s1.DataTerminalReady = 'off';
s1.FlowControl = 'none';
s1.Parity = 'none';
s1.ReadAsyncMode = 'continuous';
s1.StopBits = 1;
s1.Terminator = 'CR';

% Serial Port (additional) Properties
s1.ByteOrder = 'littleEndian';
%s1.BytesAvailableFcn = {'dispcallback'};
%s1.BytesAvailableFcnCount = 16;
%s1.BytesAvailableFcnMode = 'bytes';
%s1.ErrorFcn =
s1.InputBufferSize = 512;
s1.ObjectVisibility = 'on';
s1.OutputBufferSize = 512;
%s1.OutputEmptyFcn =
%s1.RecordDetail = 'compact';
%s1.RecordMode = 'overwrite';
%s1.RecordName = 'record.txt';
%s1.RecordStatus = 'off';

s1.Timeout = 10;
%s1.TimerFcn =
%s1.TimerPeriod = 1;


fopen(s1);

flushinput(s1);
flushoutput(s1);

%reading functions are blocking the Matlab command line.
%fread for binary read. the read operation ends when
% timeout occurs or teh specificed number of values is read (a vlaue
might
%consist of more than one byte)or the buffer is filled.
%fscanf read data and format it as character. it blocks the command
line
%until one of the following conditions takes place (terminator is
received,
%input buffer is filled, time out reached, specified number of
values is read)
%scantext
%fgetl read one line of text data without terminator. read operation
blocks
%the Matlab command until one of teh following conditions takes
place: the
%terminator is read, timeout is reached, the buffer is filled.
%fgets read one line of text data with terminator. same conditions
as fgetl
%to stop the blocking by the reading process.

%in the buffer full call back function: query the number of bytes
available
%and then read them if they are more than 0.
```

```
%writing text data through fprintf, by default the stream will be
followed
%by the terminator. also '\n' will be replaced with the terminator
%character. The binary write can be realized by using the fwrite
function.
%The write operation finsihes when the all the data is written or
the timer
%timeout condition is reached.

%in the empty call back function: query the number of bytes to
output and
%then write them if more they are more than 0.

for test_loop = 1: test_cycles


    fprintf('Test Loop No. = %d\r\n',test_loop);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
%just in case of reprogamming the * should be issued again other
wise the
%SEU will not respond.

%normally input and output buffers are flushed after fopen, flushing
here
%is for more robustness.

flushinput(s1);
flushoutput(s1);

%Set the system in UART control mode

fprintf(s1,'*');
pause(0.5);
flushinput(s1);

fprintf(s1,'*');
pause(0.5);


len = s1.bytesavailable;
input_buffer = fread(s1,len);

count=0;
for i = 1:len

    if(input_buffer(i) == res_ast_2(i))
        count = count +1;
    end
end

if(count == 12)
    fprintf('response received for Asterik command, SEU controller
in UART control mode\r\n');
else
```

```
    fprintf('Error - incorrect response to Asterik command\r\n');
    exit();
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%





%flush input buffer for receiving new data, it should be clean even
without
%FLUSHING but we do it for safety.
flushinput(s1);
flushoutput(s1);

  %Always work in detection mode and not auto correction mode to
    %accumulate errors...
    %to work in auto correction mode then this block down to the
marker
    %should be commented

    %first we send the 'd' command to put the system in detection
only mode
    %in order to test the effects of error accumulation.

fprintf(s1,'d');
pause(1);

%test the detection only mode response
len = s1.bytesavailable;
input_buffer = fread(s1,len);

count=0;
for i = 1:len

    if(input_buffer(i) == d_command_res(i))
        count = count +1;
    end
end

if(count == 14)
    fprintf('response received for detection_mode command\r\n');
else
    fprintf('Error - incorrect response for detection_mode
command\r\n');
    exit();
end


flushinput(s1);
flushoutput(s1);

%marker of autodetection mode
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    %Matlab said use it once, but I use it at each test cycle to
generate
    %as much as possible random values..


RandStream.setDefaultStream(RandStream('mt19937ar','seed',sum(100*cl
ock)));


    %Fault Injection

%Generation of random locations

%2- start by generating random values for the frame numbers and bit
numbres
%number of frames = (21D7)H, (8663)D, No of bits in each frame =
(41words x 32 bits = 1312bit) (520)h

loop_flag = 1;

while loop_flag %enable this loop to make sure that all generated
errors are SBE only without MBE.

SBE_count = 0;
MBE_count = 0;

Frame_No = unidrnd(total_no_of_frames-1,1,no_of_SEU);

[value F H]=unique(Frame_No);
rep=diff(find(diff([-Inf sort(H) Inf])));

%Test if Frame_No contains repeated bits or not.. if it contains
repeated bits
%then this means we will have to reset the FPGA at the end of this
%simulation run due to having MBE (more than one SBE in the same
frame)

len =  length(unique(Frame_No));

if ( len < no_of_SEU)
    disp('The Random Frame Numbers contain a repetition, MBE is
expected, Reset is needed at end of simulation');
    disp('repeating random number generation to get only SBEs');

    for i = 1:size(rep,2)

        if(rep(1,i) > 1)
            fprintf('Frame Value = %d  Numbers of Repetition = %d\n',
value(1,i), rep(1,i));
            MBE_count = MBE_count + 1;
        else

            SBE_count = SBE_count + 1;
        end
    end
    fprintf('Total number of error injected Frames = %d\n Number of
frames having SBE = %d\n Number of Frames having MBE = %d\n', len,
SBE_count, MBE_count);


else
    disp('The Random Frame Numbers contain no repetition, SBE is
expected');
end

%indicate which frames have repetitions, if any. (unique) return a
vector
%without repetitions, (diff) calculate diff between successive
elements,
%(find)gets the indices and values of non-zero elements, (all)
determines
%if all elements of array are non-zero ot true.

if MBE_count > 0
    loop_flag = 1;
else
    loop_flag = 0;
end

end %while loop

% plotting of random generated farmes histogram
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% colormap(hsv);
%
% x = 1:total_no_of_frames;
% hist(Frame_No,x);
%
% figure;
% scatter(value,rep,10,rep);
%
% [freq_count, bin_loc] = hist(Frame_No,x);
% figure;
% bar(bin_loc, freq_count);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%consider plotting a bar plot that represents the repetition in
groups of
%frames, to show number of 1 bit error trials, 2 bit error trials...
etc
%also use a color map


%3- generate the random bit to be inverted for each frame occurance
in the frames addresses vector.
Bit_No = unidrnd(total_no_of_bits-1,1,no_of_SEU);

%4- actual Injection

for i = 1 : no_of_SEU

    %4.1 form strings to be sent to the port "t frame address
bit_no"
```

```
t_command =
strcat('t',frames_addresses(Frame_No(i),5:10),num2str(Bit_No(i),'%.3
x'));
fprintf('Simulation Step = %d  Command = %s\r\n', i, t_command);


%4.2 writing the toggle command
fprintf(s1,t_command);
%random delay between events in the interval 0.1 till 1 sec. this is
a
%uniform random distribution should be poisson...
random_t = 0.1 + (1-0.1)*rand(1);
%pause(1);
pause(random_t);
flushoutput(s1);

end

%flush the input buffer
%issue the status command
%check mode, total no of SBE that have been corrected since
initialize,
%number of frames errors during last scan, CRC error flag, 24 bit
count of
%completed scans, record all this to a file.

%ask if system is still working?

%issue acm mode command

%issue status command

%analyze status data and notice the difference

%ask run simulation again?



fprintf('press a button before starting ACM \r\n');

    h = waitforbuttonpress;

flushinput(s1);
flushoutput(s1);
%fprintf(s1,'s');
%pause(1);
fprintf(s1,'a');
pause(1);
%6- analyze port response after each insertion and the total system
%response at the end. how many cores stopped? did system stop or
not?

%redo simulation

%7- calculate SEU Rate versus .. System Failure Rate and draw the
curve of
%system failure rate versus SEU rate
```

```
    fprintf('press a button when ready to start the next cycle, you
might need to reset the FPGA \r\n');

    w = waitforbuttonpress;



end %test loop
%close, delete, clear object
fclose(fid);
fclose(s1);
delete(s1);
clear s1;
```

Appendix G

Iterations effect on the Cost, Schedule and Reliability of Nano-Satellites Projects

G.1 Introduction

In the process of developing complex engineering projects, which consist of many cross-dependent processes, iterations might take place as the work progresses. The project complexity is mainly associated with two dimensions: the number of processes and the effort level entitled within the project phases. A third dimension that adds to the project complexity is the level of dependency among the different processes. Dependency can be classified into control and data. Control dependency takes place when a specific process controls the initiation and termination of another process. Data dependency is a form of exchanging information between an information-generating process and an information-consuming process at certain point(s) in the execution flow of both processes. In modern engineering systems, complexity is increasing tremendously with the increment in the number of diversified modules used in building larger systems. The processes used in developing, qualifying and integrating these modules usually encompass multitude of control and data interdependencies among them, which means that high level of complexity would exist.

We introduce a typical nano-satellite project as a case study of a complex engineering project with many control and data interdependencies among its processes. The case study is used to develop an estimation procedure for the three crucial parameters in any project: The schedule, the cost, and the expected product reliability. Currently, nano-satellite development activities spread all over the world. Unlike traditional large/medium satellites, nano-satellites try to achieve low-cost and fast-delivery with less emphasis on the reliability. However, this does not mean that the reliability can be ignored for nano-satellites. As long as nano-satellites are utilized for commercial purpose, a certain amount of reliability must be secured. It is a very important and difficult problem as how to balance the low-cost and fast-delivery requirements of nano-satellite projects while securing accepted reliability. Discussion on such a topic is often based on individuals experiences gained through their involvement in satellite development projects. This is due to the fact that the number of satellites, especially nano-satellites, produced so far, is few when compared to other industrial products, such as automobiles, airplanes, consumer electronics. Therefore, quantitative discussion based on the field data is extremely difficult.

The purpose of this study is to stimulate the discussion on nano-satellite cost, schedule and reliability estimations by providing the basis of quantitative discussion. We run Monte-Carlo simulations to estimate the probabilistic values of the cost and schedule of the processes which are found in a real nano-satellite project. The reliability of the satellite is assumed to be affected by the number of iterations performed. We estimate the reliability as a function in processes rework. In this way, we can evaluate cost, schedule and reliability of many hypothetic projects.

Another purpose is to develop a complexity management tool for satellite projects, not only for nano-satellites but also for larger classes as well. The motivation behind this work is driven by the fact that the satellite developers want to predict, beforehand, a reasonable estimate of the project budget, duration and product quality. The more accurate the prediction is, the more feasible the project control would be. Our proposed procedure helps in facilitating that prediction as well as providing a better chance for informative decision making. It enables the project manager to decide when to cut certain loops of iterations and estimate the risk on the total cost, schedule and reliability.

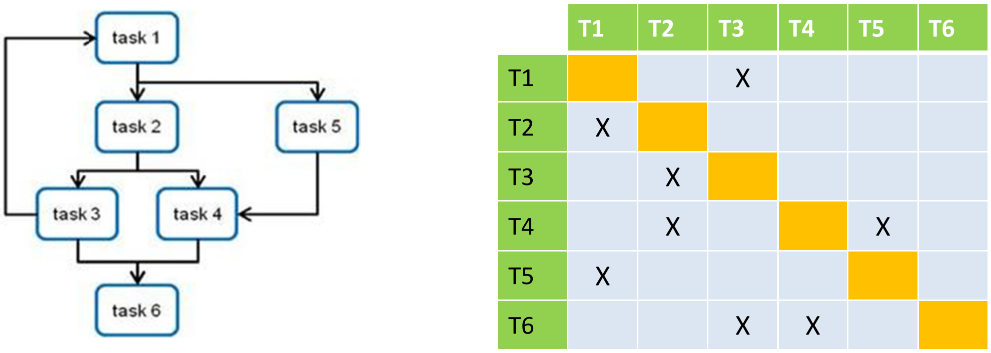
In this study, the project processes are represented using the Dependency Structure Matrix (DSM), a technique used for representing interdependent processes in complex projects. It facilitates visualizing the interdependency among processes and the modeling of project cost and schedule. This work builds upon the case study and technique that was adopted in [1] as it presented an Unmanned Aerial Vehicle (UAV) manufacturing project and calculated the cost and schedule estimates based on Monte-Carlo simulations. In this work we extend the problem studied in [1] by covering nano-satellite projects as a case study. We propose a new procedure to study the effect of the project processes on the reliability through defining the concept of Process Importance Factor (PIF). Also we handle the calculations of the expected reliability based on an empirical function in the number of processes iterations.

The study is organized as follows: Section 2 reviews the background of the DSM concept. Section 3 handles the nano-satellite case study through defining the concept and typical architecture of a nano-satellite and illustrating the phases of a nano-satellite development project followed by the DSM representation of the project processes. Section 4 shows the simulation algorithm details. Section 5 presents the results and the discussion followed by the study conclusion in section 6.

G.2 Dependency Structure Matrix Background

The DSM is used to represent the work flow among the project processes. Researchers expanded its use in modeling complex systems and handling complexity management. It was applied in many industries and services such as automotive, electronics, construction, and marketing [2][3]. The DSM has three application domains: processes interactions, product architectures, and organization structures. It is possible to represent a combination of these domains in a multi-domain DSM where the relations among the teams, products and processes can be illustrated in the same matrix.

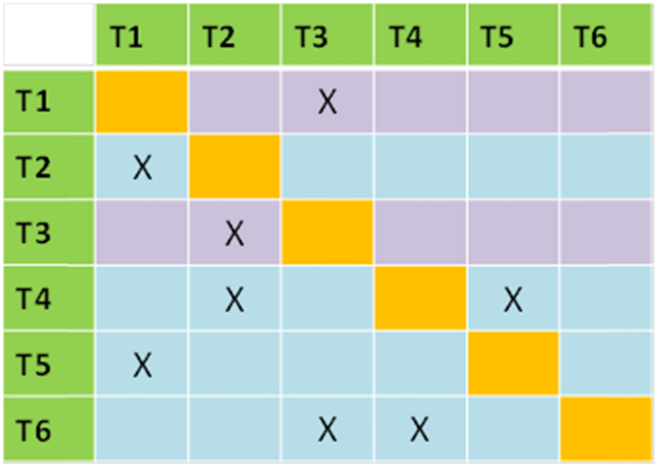
The basic idea of the DSM depends on representing the project processes in the form of an N^2 matrix. The matrix consists of rows and columns representing the project processes according to their sequence of occurrence. As shown in figure 1, a sample project consisting of 6 tasks is represented in its DSM form. The activities listed in the rows represent the processes of the project which receive inputs from other processes. The activities listed in the columns represent the processes of the project which give outputs to other processes. Reading along a row tells the sources of inputs to that specific row process. While reading along a column tells the processes to which the output of the current column process will be directed to. The feedback from any current process, such as process 5 in figure 1, to an earlier process, such as process 4, is represented with a cross mark at the location above the intersection of the current process row and column in the matrix. This form of the DSM is called Inputs in Rows\Feedback Above the Diagonal (IR\FAD). Another form of the DSM is the Inputs In Columns with Feedback Below the Diagonal (IC\FBD). The DSM can be analyzed using several algorithms such as clustering, tearing, banding and partitioning. In the case of project processes representation, we use the banding algorithm. The processes listed in the DSM are ordered according to their sequence of execution which denotes the control flow of the processes execution in the project. The data exchanged, at the interface from one process output to another process input, is represented by the cross-marks. The DSM can be of binary form or multi-valued form. The binary DSM contains 0s and 1s which denote the presence or absence of information flow. In a multivalued DSM several values can be used to represent different logic notions.



	T1	T2	T3	T4	T5	T6
T1	🟧		X			
T2	X	🟧				
T3		X	🟧			
T4		X		🟧	X	
T5	X				🟧	
T6			X	X		🟧

Figure 1, DSM representing a sample project consisting of 6 tasks, inputs are in rows while feedback is above the diagonal.

In sequential processes simulation using the DSM, the banding algorithm is used to identify the possibilities of having processes that can run in parallel. If a process waits for an input from its predecessor process then it cannot run in parallel with its predecessor. For example, in figure 1, process number 3 takes an input from process number 2. This means that process number 3 should wait until process number 2 finishes execution to provide the needed information to process number 3. These two processes cannot run in parallel. If we consider process number 4 which takes a feed-forward input from process number 2 and a feed-backward input from process number 5, we would discover that it is independent from process number 3. As process number 4 takes no inputs from process number 3 then at the time step where process number 3 would be executed, process number 4 can start execution as well. This parallel execution is limited by the ability of the system to dispatch the needed resources among the parallel running processes. Resources can be of different forms such as human resources, financial resources, and equipment. For mathematical simulations we consider the resources as being unlimited therefore parallel processes can start execution together and at once. In reality this assumption should be limited by the constrained number of available resources. Figure 2, shows the categorization of the matrix shown in Figure 1 into bands of parallel processes. Processes 4, 5 and 6 are located in the same band as they can start running in parallel at the same time step after finishing process 3. However, process 2 should wait for process 1 to finish and process 3 should wait for process 2 to finish before any of them can start running. If an earlier process, such as process 1, receives a feedback from a later process, such as process 3, then it does not mean that the earlier process has to postpone its execution until the feedback is ready. Feedback is probabilistic which means that it might or might not lead to rework of the earlier process. If rework takes place then the earlier process will be re-executed again after the later process finishes.



	T1	T2	T3	T4	T5	T6
T1			X			
T2	X					
T3		X				
T4		X			X	
T5	X					
T6			X	X		

Figure 2, applying the banding algorithm to the DSM matrix.

G.3 Nano-satellite Project Case Study

In the simulations that we run to present the results in this study we used a simple case study of a typical nano-satellite. It is a 50 kg satellite in a (50cm × 50cm × 50cm) volume. Usually nano-satellites are characterized as low cost and high risk satellites. Typically they have no redundant subsystems as they are produced with an expectation of failure that is much higher than professional larger satellites. The typical missions for nano-satellites include technology demonstration for new subsystems and\or technologies, earth observation with moderate spectral and spatial resolutions, store and forward systems, and scientific missions.

Figure 3, shows the architecture of a generic non-redundant nano-satellite configuration. The On-Board Computer (OBC) controls the switch on/off of subsystems and change of operation modes. It also sends the received command stream to the relevant subsystem after decoding it. The telemetry information is collected by the Telemetry Module (TM) and transferred to the OBC for encoding and further transfer to the ground station via the Communication Subsystem (CSS). The CSS is the interface between the satellite bus and the ground station and it receives the encoded command stream that is forwarded to the OBC for decoding then executing or distributing it to the relevant subsystems. The Power Subsystem (PSS) is responsible for controlling the charging of the batteries through the solar panels and distributing the power over the power bus to the satellite subsystems. Power can be distributed in regulated or non-regulated form. The Attitude Determination and Control Subsystem (ADCS) is responsible for detecting the current satellite orientation and controlling it to reach a target orientation and stabilize within certain angular velocities ranges as required by the mission. It needs to be updated with the timing information from the navigation subsystem such as the Global Positioning System receiver (GPS) for the attitude prediction algorithms to work properly. The GPS also provides time scale corrections to the OBC.

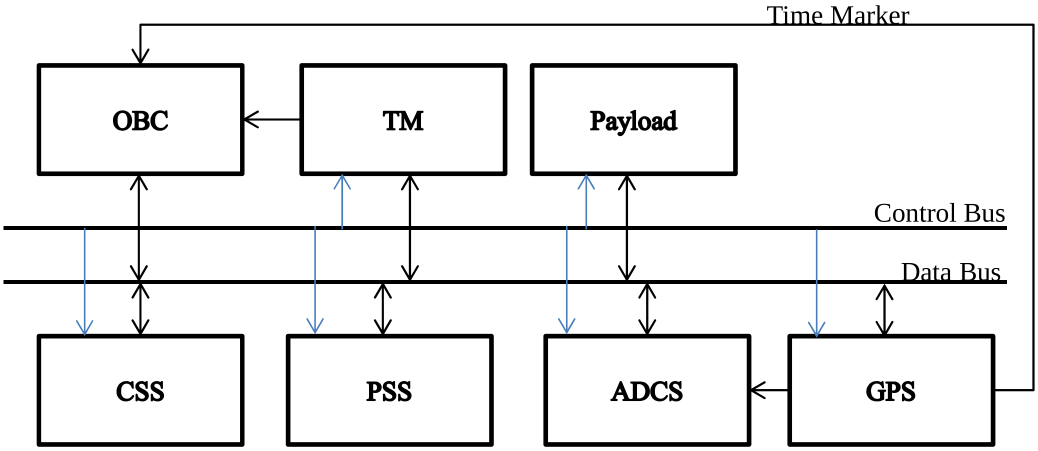


Figure3, Simple Nano-Satellite Architecture.

a) Project Life Cycle

Building a satellite follows a lifecycle that is defined by the project plan. The lifecycle is divided into group of activities which are carried out along the project phases. The activities define the inputs, outputs and processes which take place within the phases. Figure 4, shows the main phases used in developing a nano-satellite project. The technical requirements and specification phase starts with a general statement of the required mission. In that phase, detailed technical requirements and specifications on the system and subsystem levels are specified. The Structure and Thermal Model (STM) phase handles the activities related to producing the nano-satellite model which is used in vibration and thermal vacuum tests. The Qualification Test Model (QTM) phase handles the activities related to producing the functional nano-satellite subsystems and performing the integrated environmental tests. The Flight Model (FM) phase handles the production of the flight approved model of the nano-satellite and performing the end-to-end (E2E) communication tests. In the following section, the DSM of the project phases and activities is described in details.

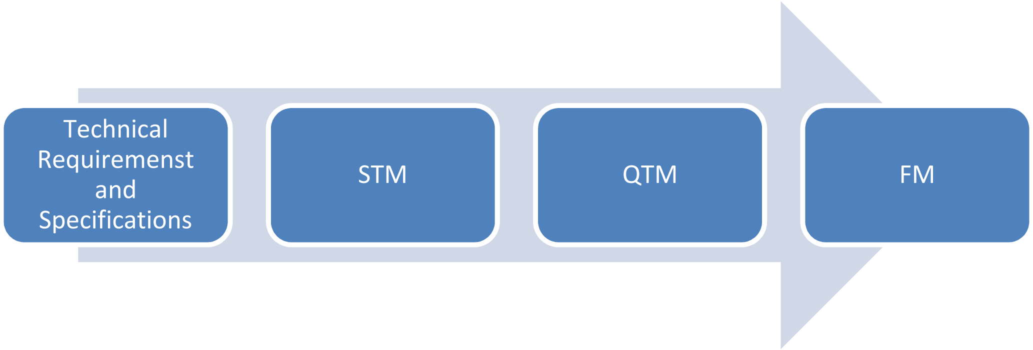


Figure 4, Main phases in a nano-satellite project.

b) Project DSM Model

The nano-satellite project full DSM model is shown in Figure. 5. The complete set of processes used in developing the nano-satellite is illustrated. A total of 177 processes are presented in this matrix. The color bands are used to group the processes into phases of related activities. Each phase ends by a review process which forms a project milestone. The black bounded boxes on the matrix diagonal represent the processes within each phase.

If we consider any of the black bounded boxes and look at its left then this would denote the inputs this box of processes is taking from predecessor processes in a feed-forward fashion. If we look at the right of the box then this denotes the inputs from successor processes in a feed- backward fashion. If we look above the box then this denotes the feed-backward outputs which are given by the box processes to the upstream, predecessor, processes. If we look below the box then these are the feed-forward outputs given by the box processes to the subsequent, coming, processes. So, for each black bounded box of processes in the DSM matrix in figure 5, group of collected processes, we have four cases: Left which denotes input from/feed-forward, right which denotes input from/feed-backward, up which denotes output to/feed-backward and down which denotes output to/feed-forward.

The DSM in figure 6, allows us to understand the complexity of the satellite development processes. We can spot the locations which enjoy high level of feed-back. The yellow marks represent the information flow. Feedback is mainly located at three locations in the matrix: the beginning, near the middle and at the end of the project.

At the beginning of the project where the specifications of the satellite system, subsystems and components are being settled, many iterations take place as the parameters needed to be specified are interdependent. Usually we start with a set of parameters for the mission and then we deduce the rest of the parameters iteratively by fine tuning until reaching the final set of parameters and specifications of the satellite.

As the work in the project progresses, subsystems start to be produced. Subsystems are developed separately based on the specifications which were set at the beginning and then integrated together for mutual functional testing. At the integration phase, electrical, mechanical and thermal interfacing problems usually start to show up. Iterations are performed to correct the interfaces, and in some cases the functionality, of each subsystem to perform as it should be. This step is affected by the level of accuracy of the interface control documents used to relay the information between the subsystem developers. It is also affected by the design that is adopted to develop the subsystems and the components quality. Therefore most of the feedback in this band, iterations, is affected by the previous two bands which indicate the design, unit testing and components procurement of the satellite subsystems.

The flight model testing is the final step in developing a satellite. This step requires great focus on reviewing the details of all specifications to verify that they were accurately satisfied. Iterations might take place due to the design, manufacturing, components quality and acceptance tests which covers the previous two steps.

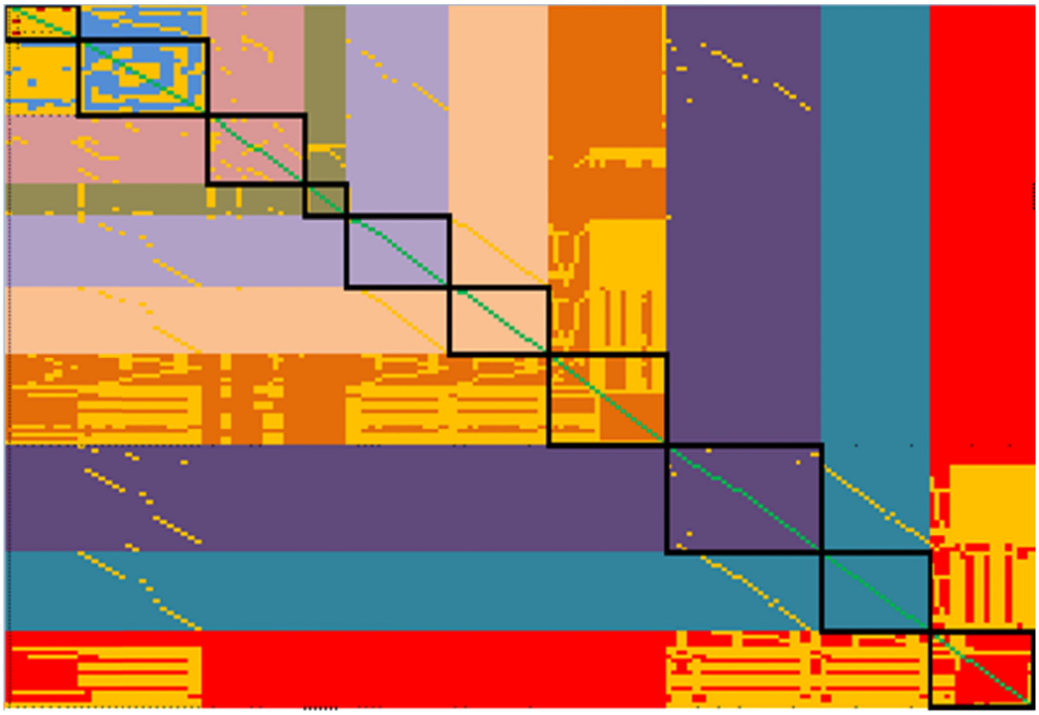
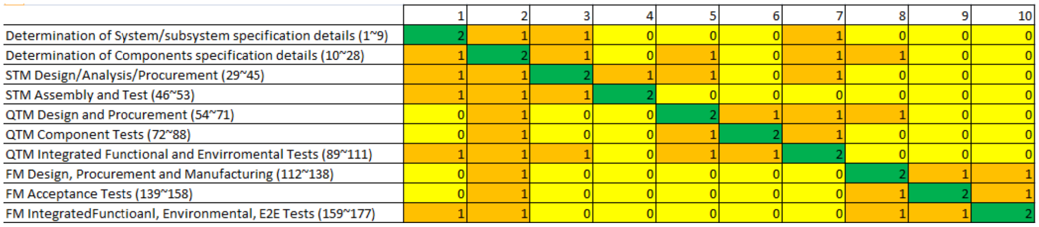


Figure 5, Full activities - DSM of the nano-satellite project.

The processes in figure 6, represent the 10 color bands, project phases, shown in figure 5. The groups of processes in each of the bands shown in figure 5 are represented by the numbers in the parentheses in figure 6. For illustration, the first two processes groups in figure 6, which correspond to the processes from 1 to 9 and from 10 to 28 in figure 5, are presented in details in figure 7. We will use the simple DSM matrix in figure 6 to prove the validity of the model results as we run the simulations on the full and simple (scaled down) DSM.



	1	2	3	4	5	6	7	8	9	10
Determination of System/subsystem specification details (1~9)	2	1	1	0	0	0	1	0	0	0
Determination of Components specification details (10~28)	1	2	1	0	1	0	1	1	0	0
STM Design/Analysis/Procurement (29~45)	1	1	2	1	1	0	1	0	0	0
STM Assembly and Test (46~53)	1	1	1	2	0	0	0	0	0	0
QTM Design and Procurement (54~71)	0	1	0	0	2	1	1	1	0	0
QTM Component Tests (72~88)	0	1	0	0	1	2	1	0	0	0
QTM Integrated Functional and Envirromental Tests (89~111)	1	1	1	0	1	1	2	0	0	0
FM Design, Procurement and Manufacturing (112~138)	0	1	0	0	0	0	0	2	1	1
FM Acceptance Tests (139~158)	0	1	0	0	0	0	0	1	2	1
FM IntegratedFunctioanl, Environmental, E2E Tests (159~177)	1	1	0	0	0	0	0	1	1	2

Figure 6, Main project phases - DSM of the nano-satellite project.

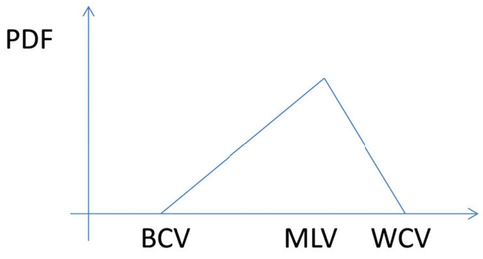
1	Determination of system specification details		■																											
2	Determination of sub-system specification details	Structure	○	■	○	○		○	○	○	○																			
3		Thermal	○	○	■	○	○	○	○		○																			
4		Communication	○	○		■	○	○	○	○	○																			
5		C&DH	○			○	■	○	○	○	○																			
6		AOCS	○	○		○	○	■	○	○	○																			
7		Power	○	○	○	○	○	○	■	○	○																			
8		Harness	○		○	○	○	○	○	■	○																			
9		Mission	○	○		○	○	○	○	○	■																			
10	Determination of component specification details	Structural body		○								■	○	○	○	○	○	○		○	○	○	○	○	○	○	○	○	○	
11		Sep. system		○								○	■																	
12		Radiator /heater			○							○		■										○	○	○				
13		TM/Com TX/RX			○	○						○			■		○	○	○							○				
14		Downlink TX			○	○						○				■	○	○	○							○				
15		Antenna				○		○				○			○	○	■													
16		OBC hardware			○		○	○				○		○	○	○		■	○	○	○	○	○			○			○	
17		C&DH SW					○							○	○	○		○	■	○	○	○	○			○			○	
18		AOCS SW			○			○				○						○		■	○	○	○			○				
19		Att. actuator			○			○				○						○		○	■	○				○				
20		Att. Sensor			○			○				○						○		○	○	■				○				
21		GPS			○			○				○						○		○			■			○				
22		Solar panel			○				○			○		○										■	○	○				
23		Battery			○				○			○		○										○	■	○				
24		PCU+PDU			○				○			○		○	○	○		○	○	○	○	○	○	○	○	■			○	
25		Cables/connectors								○		○		○	○	○	○	○		○	○	○	○	○	○	○	■		○	
26		Lens			○						○	○																■	○	
27		Camera			○						○	○						○	○							○		○	■	
28	Review		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	■

Figure 7, First 28 processes concerned with specification details IR/FAD.

G.4 The Simulation Algorithm

The Nano-satellite project processes are represented using DSM. The order of execution of the processes matches their order in the DSM. For example, the process located at row number 3 in the DSM would be executed after the process in row number 2 and before the process in row number 4. The processes can execute in parallel if they have no interdependency in exchanging outputs as described in section 2. The exchange of information between the processes is presented at the intersections of a row and a column in the DSM.

For each process there is a variable cost and schedule range. The range specifies the worst, most likely and best values for each process. This notation is called the TRIPDF function [1]. The same concept is used in the Program Evaluation and Review Technique (PERT) when developing project schedules. The likelihood to obtain a specific value for the cost and schedule of any process depends on the TRIPDF distribution. The model starts by generating random variables that follow the TRIPDF distribution for each process. The random variables represent the cost and schedule pairs for the different process. Figure 8, shows the TRIPDF distribution.



$$\text{TriPDF Area} = b\,h\,/\,2 = (\text{WCV - BCV})\ \text{P(MLV)}\,/\,2 = 1$$

$$\text{P(MLV)} = 2\,/\,(\text{WCV - BCV})$$

BCV: Best Case value
MLV: Most Likely Value
WCV: Worst Case Value

Figure 8, TriPDF distribution.

The mathematical model is developed using the C language. The model simulates the execution of the project processes in the order of their appearance in the DSM. Whenever a process is executed for some time ΔT, its duration is reduced with the same amount. This indicates that the process already run and would finish when its duration equals to (0). The cost is increased in a linear relation with the elapsed duration. As the process spends more time in execution its duration is decreased and its cost is increased.

The DSM matrix is analyzed by the banding algorithm to detect the set of processes that can run in parallel during the same simulation step. The step itself is set to a value that is lower than the lowest duration change in any process. For example, if the lowest duration for any activity is 1 day then the simulation step should be set to less than 1 day. The smaller the simulation step the higher the accuracy of the model outputs. Simulation can run by progressing the time in terms of small steps of ΔT and calculating the change in the processes durations and costs based on that change or through discrete events of change [1].

The DSM representation shows the possible iterations in the form of feedback marks above the diagonal in the (IR/FAD) form. This feedback from the downstream activities to the upstream might lead to probable rework of downstream activities in between the process that received the feedback and the process that issued the feedback [1].

This rework is not certain as it depends on finding a necessity to go back and modify something in a previously iterated process. In some cases of projects, feedback represents an input that must be taken into consideration but it is not available at the moment of running the activity. The feedback value is then estimated as a preliminary value which would be modified when output from the upstream activity becomes available. The rework in that situation is certain and not probable as it represents an input that was roughly estimated beforehand.

The rework probability is estimated in the model based on the fact that it is less probable to rework earlier upstream processes due to the change that takes place in later downstream processes. Thus the rework probability decreases as we go towards the end of the project when we traverse through rows which represent the affected processes. To compensate for the probable modification of results that might happen later due to the availability of information in a more accurate form, the rework probability increases slightly as we traverse through column processes. This represents an increase in the probability of rework for the affecting processes. These two conditions seem to be contradictory, increase as we traverse through columns and decrease as we traverse through rows, however this is the physical conditions that do really exist. Figure.9, shows the rework probability over the rows and columns of the DSM matrix.

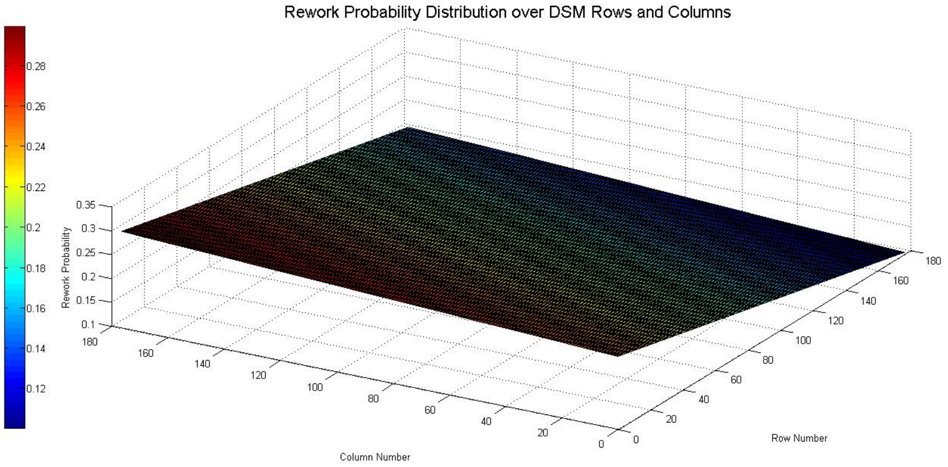


Figure 9, Rework probability over the rows and columns of the DSM.

The satellite subsystems depend in their development on some of the processes that are listed in the DSM. The processes are relevant to each subsystem depending on the project phase and the subsystem itself. The effect of each process on the successfulness of developing the subsystem and its ability to meet the design requirements is represented by a weight function. It can be seen as a function which defines the importance of each process to each subsystem. Three weight functions are being studied: Equal weight, increasing weight and decreasing weight. Figure 10, shows the different weight functions plots. In equal weight all the processes have the same importance all over the project phases. In increasing weight processes have more importance as we move towards the end of the project. In decreasing weight processes have less importance as we move towards the end of the project. The importance of a process is a heuristic for the effect that the process might have on the project in case its outputs did change.

The importance of the processes and their effect on the project outcomes is affected by the management approach. Some process managers tend to focus on the project at its beginning to start correctly and build upon the success of each stage. Others like to focus on the test stage and think that testing is sufficient to overcome the mistakes of others as long as cost of change will not blow up. The majority tend to deal with the project processes as equi-important.

The weight function is a simple first degree polynomial which relates the process number to a weighting factor. In figure 10, the area under the weight curve is equal to 1. We use simple linear functions to represent the weight effects. Nevertheless, a quadratic weight function might also be used to have smoother transitions. Sometimes processes have higher importance at the beginning then saturate then decreases importance as in the shape of a trapezoid function. Another weight function might be the saw tooth or even n-degree polynomial. The weight function tries to mimic the behavior of process managers and executers when leading\executing projects. Usually people tend to start building enthusiasm towards doing the tasks gradually then they saturate then start to lose it gradually. The peak of work is mostly near the middle of the project. The quadratic function might be useful in representing that situation. It can be skewed in either direction to reveal the tendency to focus on the first half (before the peak) or the second half of the project (after the peak).

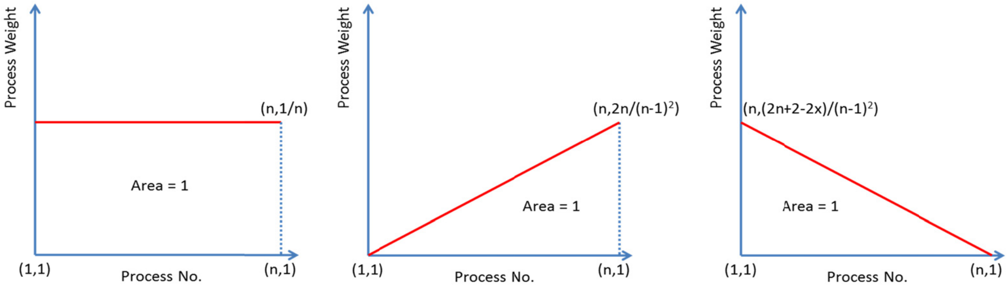


Figure 10, Weight functions plots, equal weight, increasing weight, and decreasing weight.

The simulation proceeds by repeating the runs in the form of batches where each batch consists of 1000 runs, then comparing the mean value and the variance for the total cost and total schedule between the consecutive batches, a decision is made as when to stop the simulation. In case the last two batches have an error of 0.01 between their mean and variance then we consider the simulation stable and we stop running further batches. In each run we calculate the total reliability of the project taking into account the weight of the processes. Figure 11, shows the simulation cycle of the model. It starts with loading the simulation parameters and the DSM matrices. A distribution generator generates the random pools of the PDF and CDF functions for the cost and schedule TRIPDF of each process. A random selection is carried out from the pools of random variables to select the cost and schedule of each process randomly. The simulator core is responsible for traversing the DSM matrix starting from the first process and going in order, process after process. Whenever a probable rework is found then a random number is generated to evaluate whether to do that rework or not. The rework is based on the rework probability matrix that is defined for each process in the DSM as shown in figure 9. If rework takes place then all the interim processes between the reworked process and the original process are evaluated for probable rework using the same concept of random number generation [1]. The model then starts to estimate the reliability based on a complete run of the model. Each run traverses the DSM matrix from the beginning till the end in increments of time step that is used to progress a duration counter for the processes as if they were executing. The cost is linearly dependent on the schedule progress and increases as portions of the schedule are counted. The system then produces the simulations graphs using a MATLAB script.

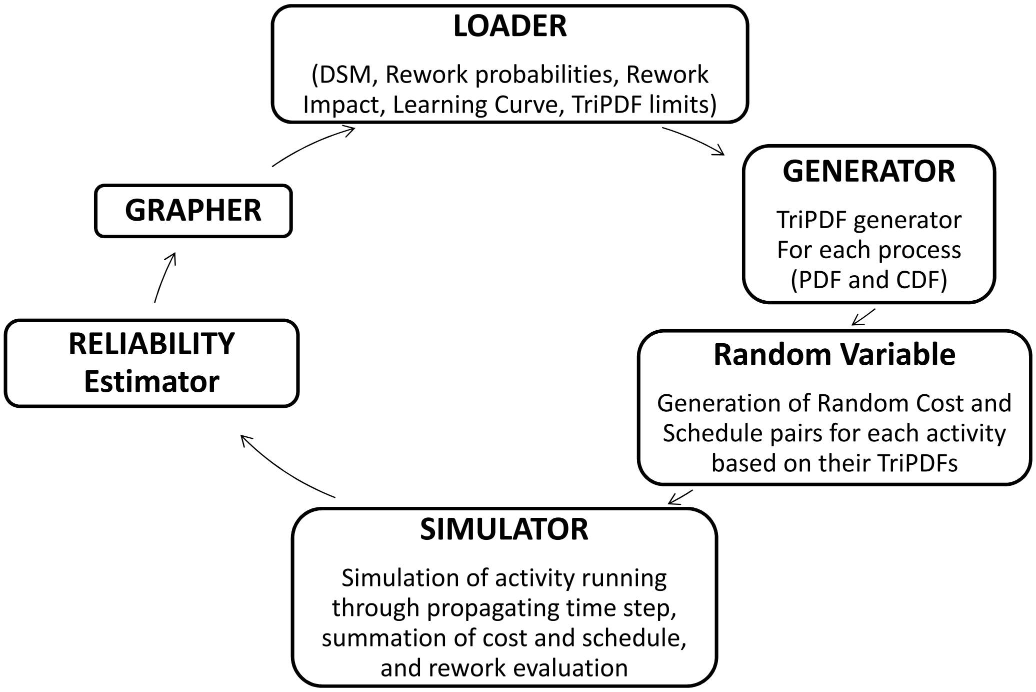

LOADER
(DSM, Rework probabilities, Rework Impact, Learning Curve, TriPDF limits)
GENERATOR
TriPDF generator For each process (PDF and CDF)
Random Variable
Generation of Random Cost and Schedule pairs for each activity based on their TriPDFs
SIMULATOR
Simulation of activity running through propagating time step, summation of cost and schedule, and rework evaluation
RELIABILITY Estimator
GRAPHER

Figure 11, the simulation cycle block diagram.

Equations (1 to 7) show the formula used in the schedule, cost and reliability calculations. The schedule duration in Eq. (1) is a summation of all the schedule durations of the project processes. The duration for each process is calculated based on the initial random value that was generated from the TRIPDF distribution and modified by accumulating the extra durations due to reworks for that process. The cost in Eq. (2) is calculated in the same way as the schedule. The reliability in Eq. (3) is a multiplication of the subsystems reliabilities. Where the reliability of each subsystem is calculated as in Eq. (4) using the probability of failure of the subsystem itself. The probability of failure of any subsystem is affected with the rework of the processes used to develop it. It changes after rework due to the volume of reworked processes. This is represented in Eq. (5) and Eq. (6) where we used the logarithm to be effective when there is high volume of rework as it would lead to reasonable values of the probability of failure. If we consider using the exponential function instead of the logarithm when we have high volume of reworks then we would have very small values, almost zeros, for the probability of failure. The use of the logarithm has two cases, when the total number of rework is less than 1 and when it is greater than or equal to 1. This is to account for the negative values produced by the logarithm function for fractions, the less than one case, and also when there is no rework at all, total rework equal to 0.

The total rework number of each process is calculated as in Eq (7) by summing the product of the rework count for each process and the process weight effect. This gives a weighted average for each process. The weighted average value for each of the processes belonging to a specific subsystem is then summed together to form the total weighted average for that subsystem processes. We call this total value the rework index. We emphasize here on the concept of reliability improvement in this model. The reliability is improved by multiplying the probability of failure of each subsystem by its rework index logarithm.

$$C = \sum_{i=1}^{n} c_i \tag{1.}$$

$$S = \sum_{i=1}^{n} s_i \tag{2.}$$

$$R_{system} = \prod_{i=1}^{m} R_i \tag{3.}$$

$$R_i = 1 - P(failure\ after\ rework)_i \tag{4.}$$

Where (m) is the total number of subsystems

$$P(failure\ after\ rework)_i = P(failure\ before\ rework)_i / \log(x) \tag{5.}$$

Provided that [x > 1]

$$P(failure\ after\ rework)_m = P(failure\ before\ rework)_m \tag{6.}$$

Provided that [x ≤ 1]

Where (x) is the total number of rework for subsystem processes given by:

$$x = \sum_{h=1}^{m} \text{process rework count}_{\text{h}} \times \text{Weight Effect of process(h)}\ \text{x} = \sum_{\text{h}=1}^{\text{m}} \text{processreworkcount}_{\text{h}} \times \text{WeightEffectofprocess(h)} \tag{7.}$$

G.5 Results and Discussion

The simulation was run on two models. The full scale DSM of the nano-satellite project and the scaled down or we call it the grouped DSM. The purpose is to prove that the model results are valid and to assure the conclusions we reached from running the full scale model. Table 1 and Table 2 show the simulation parameters used for both models.

Table 1 Simulation Parameters of the full DSM model.

Parameter	Value
Number of processes	**177**
Number of subsystems	**8**
Number of runs per (Batch)	**1000**
Subsystem probability of failure	**0.1**
Mean error threshold	**0.01**
Variance error threshold	**0.01**

Table 2 Simulation parameters of the grouped DSM model.

Parameter	Value
Number of processes	**10**
Number of subsystems	**8**
Number of runs per (Batch)	**10000**
Subsystem probability of failure	**0.1**
Mean error threshold	**0.01**
Variance error threshold	**0.01**

The main difference is in the number of the processes being simulated and the number of runs per batch.

The simulation in the grouped DSM makes use of the TRIPDF distribution shown in Table 3. The best case is 10% below the most likely case. The worst case is 10% above the most likely case. The estimations of the cost and schedule for each process were deduced from the full scale DSM TRIPDF estimations.

Table 3 TRIPDF distribution of the grouped DSM processes.

Schedule (Days)			**Cost (K JPY)**		
BC	**ML**	**WC**	**BC**	**ML**	**WC**
99	100	101	13860	14000	14140
173.25	175	176.75	10890	11000	11110
181.17	183	184.83	26284.5	26550	26815.5
38.61	39	39.39	8068.5	8150	8231.5
60.39	61	61.61	46530	47000	47470
172.26	174	175.74	39501	39900	40299
105.93	107	108.07	25542	25800	26058
133.65	135	136.35	108652.5	109750	110847.5
110.88	112	113.12	20889	21100	21311
104.94	106	107.06	27423	27700	27977

The grouped DSM simulation, with equal weight function applied, generated the schedule, cost and reliability distributions shown in Figure 12 and Figure 13.

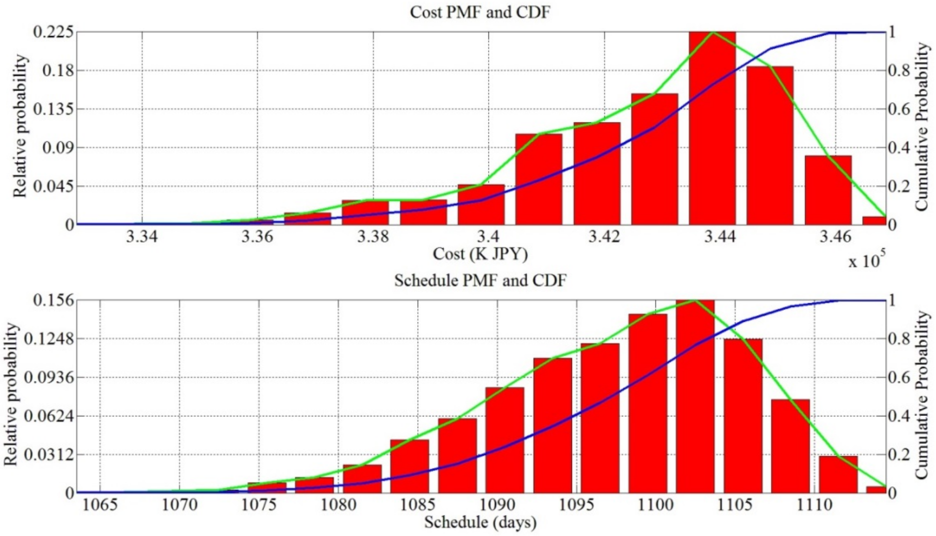


Figure 12, Schedule and Cost distribution of the grouped DSM (10 processes), with equal weights of the processes.

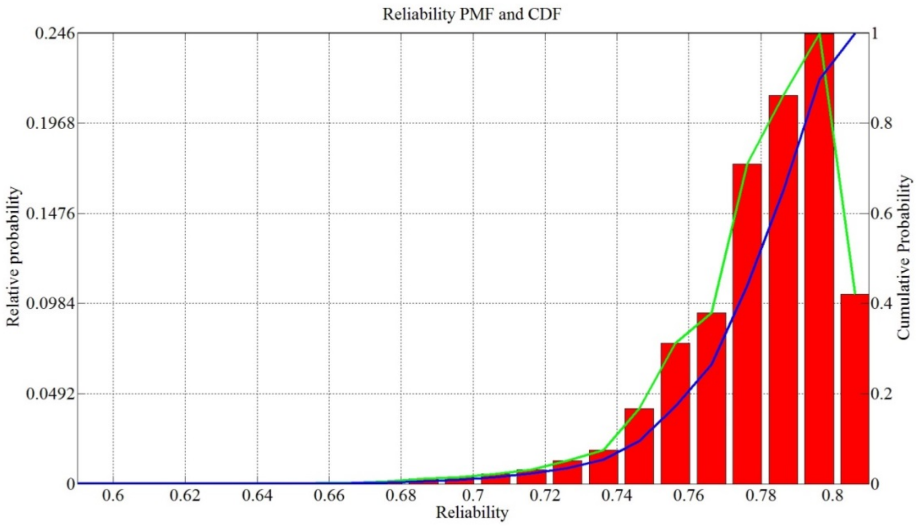


Figure 13, Reliability distribution of the grouped DSM (10 processes), with equal weights of the processes.

The statistical data of the distributions is shown in Table 4. Reliability reached up to 80.63% with iterations.

Table 4 Statistical data of the schedule, cost and reliability distributions of the grouped DSM simulation.

	Min	**Max**	**Mean**	**Std**	**Range**
Cost (K JPY)	3.329e+05	3.469e+05	3.399e+05	4472	1.4e+04
Schedule (Days)	1064	1115	1089	16.02	51
Reliability	0.5863	0.8063	0.6963	0.0678	0.22

The 3D plot of the cost schedule and reliability is shown in Figure 14 and the joint PDF between the cost and schedule is shown in Figure 15. It is clear that the reliability and cost increase with the schedule . Also from Figure 15, there is a peak at which the cost and schedule would have the highest joint probability value. The corresponding total schedule and total cost are the most probable to occur in the project. This is useful in planning as to expect the probability of finishing within certain target budget and time.

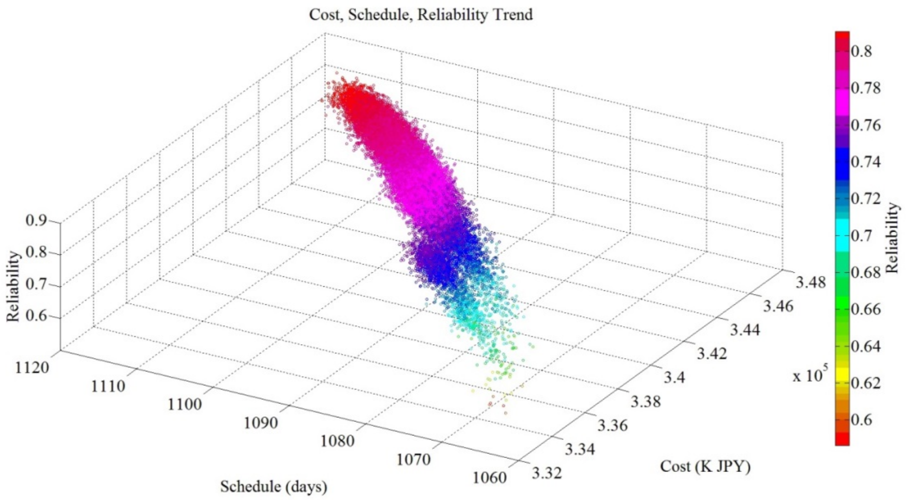


Figure 14, 3D plot of the cost, schedule and reliability for the grouped DSM.

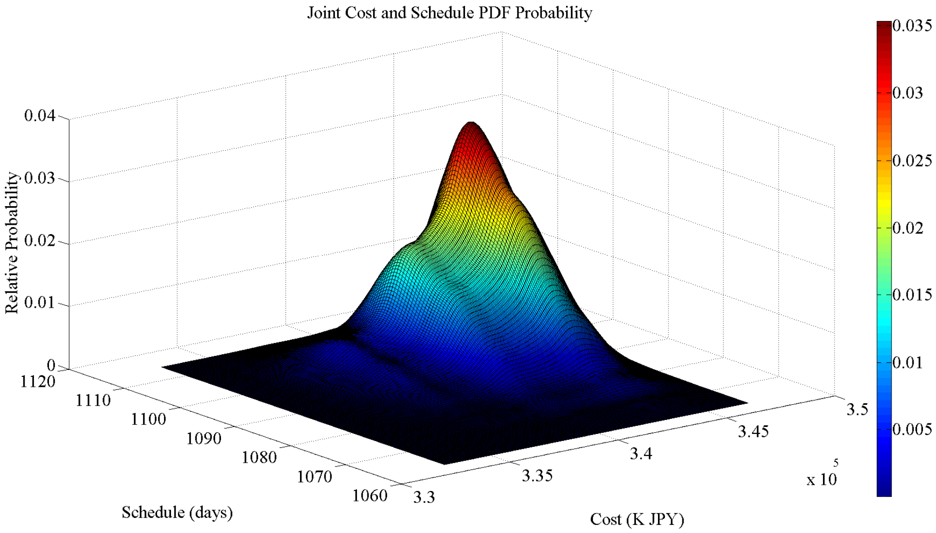


Figure 15, 3D plot of the joint cost and schedule PDF for the grouped DSM.

The grouped DSM model, 10 processes, shows a trend of increasing cost versus schedule in a linear function as illustrated in Figure 16. This is logical as the cost keeps rising as long as the project schedule keeps increasing, due to the incurred cost in operations and resources. However, Figure 17 shows a trend of saturation in between the cost and reliability. So as cost keeps rising, reliability tends to saturate. Which means that, at some point in the project, regardless of the amount of processes being iterated, which obviously leads to increment in the schedule and cost, the reliability will not improve significantly. More incurred cost due to more iteration does not mean endless improvement in the reliability. This fact is also clear from Figure 18, where the reliability tends to saturate even when more time is spent in iterating previously accomplished processes. The reason for having an upper reliability limit depends on the intrinsic goodness of the subsystems which is initially defined by the failure rates.

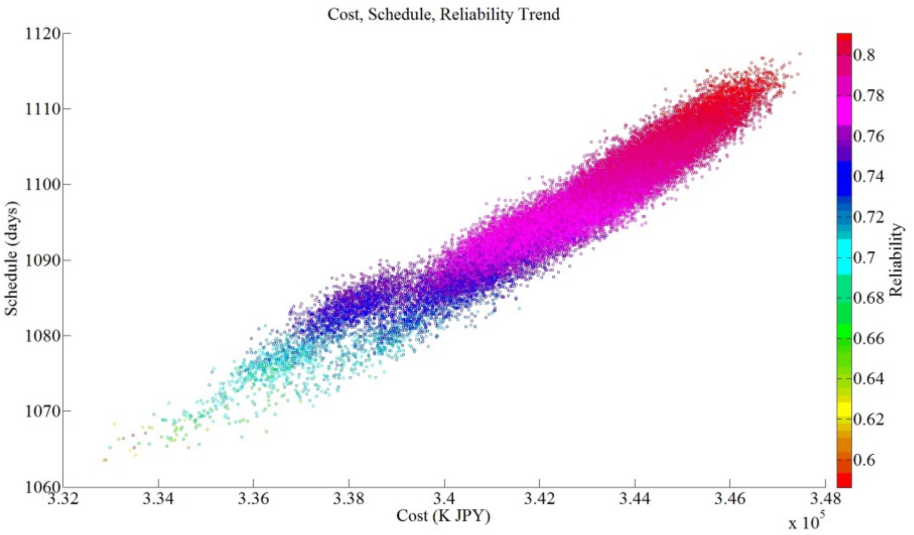


Figure 16, Cost and schedule relation in the grouped DSM.

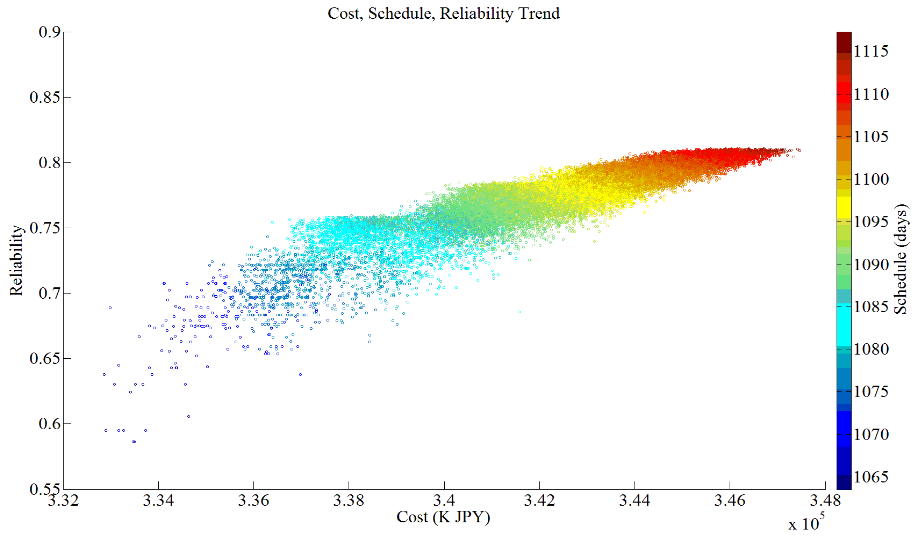


Figure 17, Cost and reliability relation in the grouped DSM.

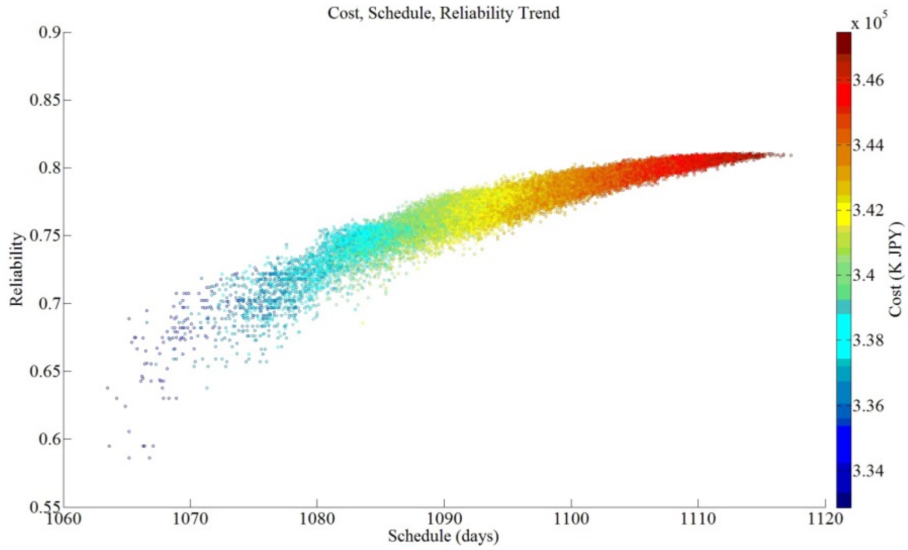


Figure 18, Schedule and reliability relation in the grouped DSM.

The simulation was run on the full scale DSM, 177 processes, with equal, increasing and decreasing weights of the processes. Figure 19 and Figure 20 show the distribution of the cost, schedule and reliability after running the simulations.

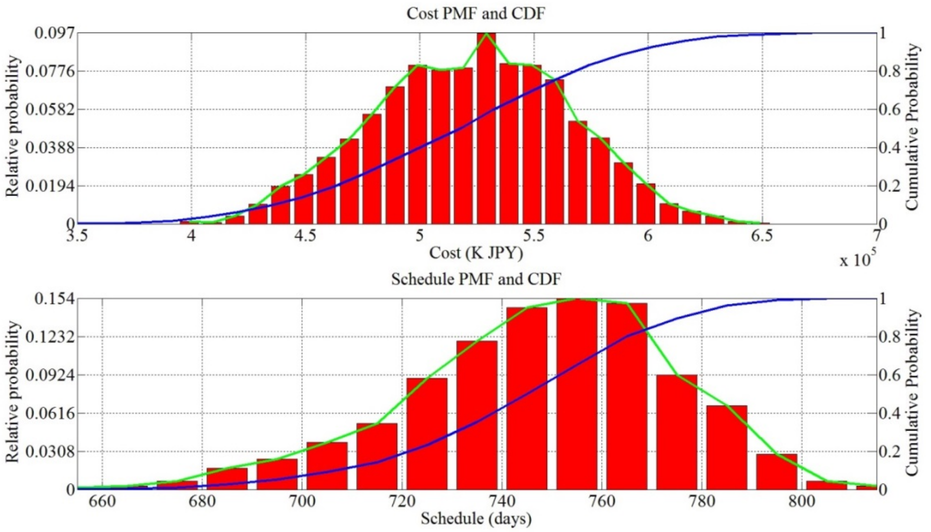


Figure 19, Schedule and Cost distribution of the full DSM (177 processes), with equal weights of the processes.

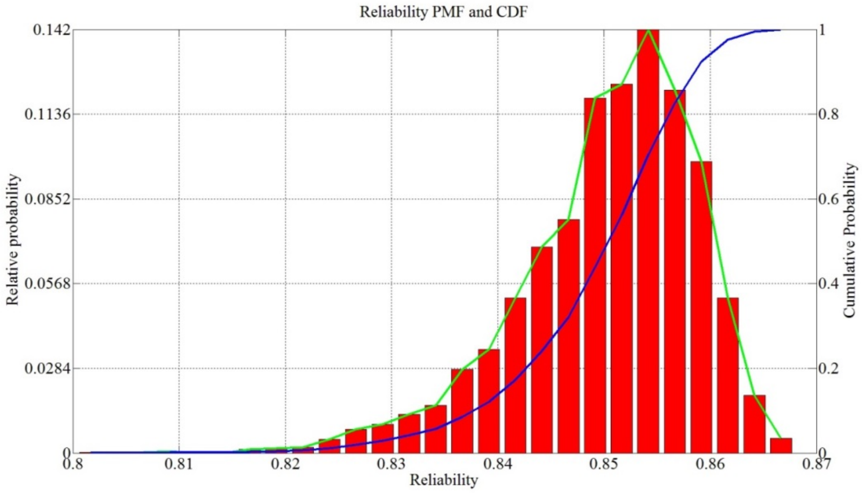


Figure 20, Reliability distribution of the full DSM (177 processes), with equal weights of the processes.

The statistical data of the distributions is depicted in Table 5. The reliability can reach a maximum of 86.67%. However, the schedule and cost would increase drastically. With the minimum reliability of 80.17% and the maximum of 86.67%, a 1% of reliability improvement corresponds to 38461.5 K JPY increase in cost and 24.62 days increase in schedule.

Table 5 Statistical data of the schedule, cost and reliability distributions of the full DSM simulation.

	Min	**Max**	**Mean**	**Std**	**Range**
Cost (K JPY)	3.99e+05	6.49e+05	5.24e+05	7.649e+04	2.5e+05
Schedule (Days)	655	815	735	50.5	160
Reliability	0.8017	0.8667	0.8342	0.01984	0.065

Figure 21, shows the 3D plot of the joint PDF function between the cost and the schedule. The multimodals reprent the effect of processes iterations. Several peaks exist due to variety of iteration probabilities and paths.

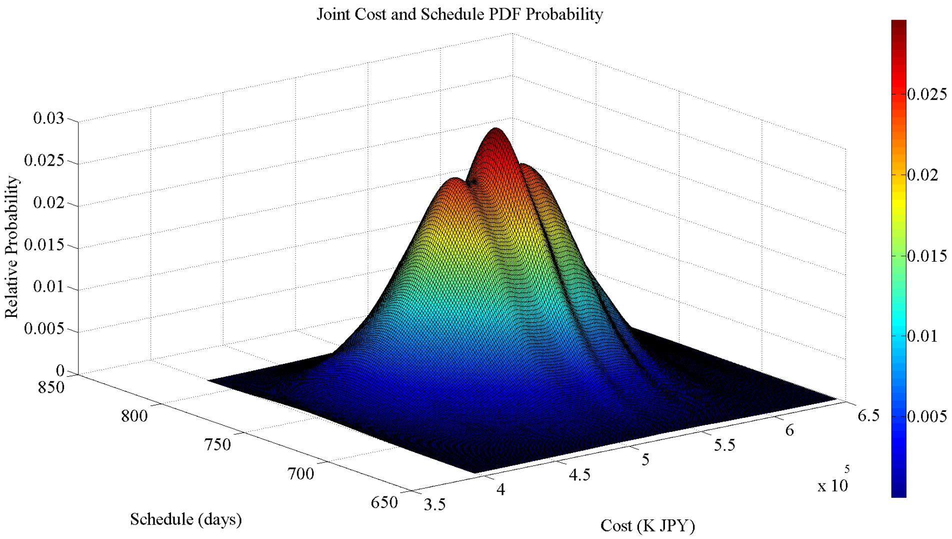


Figure 21, 3D plot of the joint cost and schedule PDF for the full DSM.

Figure 22, is a top view of Figure 21. It can be used in project planning. The highest joint probability of the cost and schedule corresponds to a range from 5.26 K JPY to 5.4 K JPY for the cost and from 746 days to 761 days in the schedule. A target should be defined, the black lines, and the project manager should calculate the probability of finishing the project at the required target based on the joint PDF values.

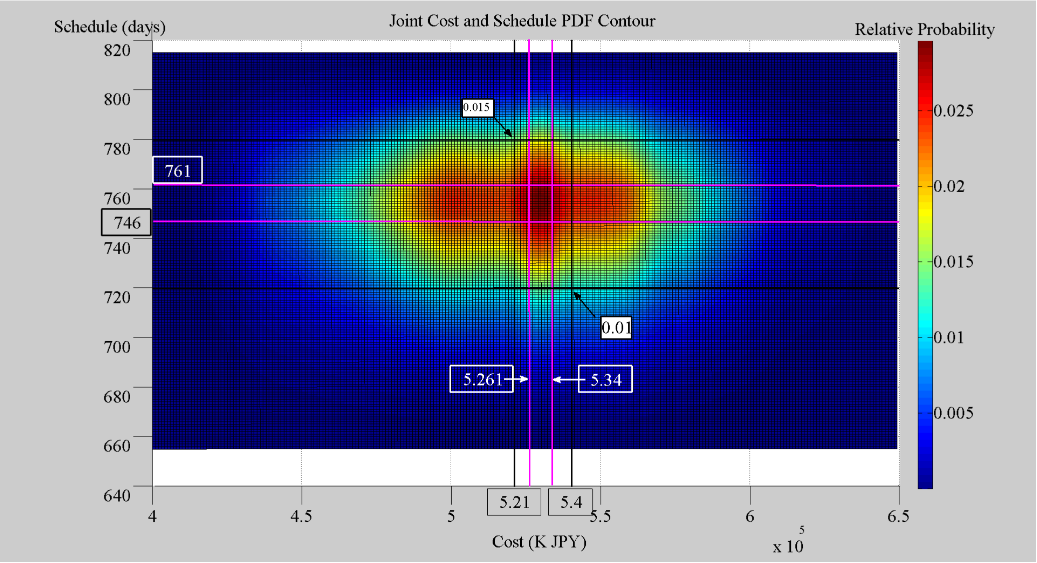


Figure 22, Top projection of the joint cost and schedule PDF for the grouped DSM.

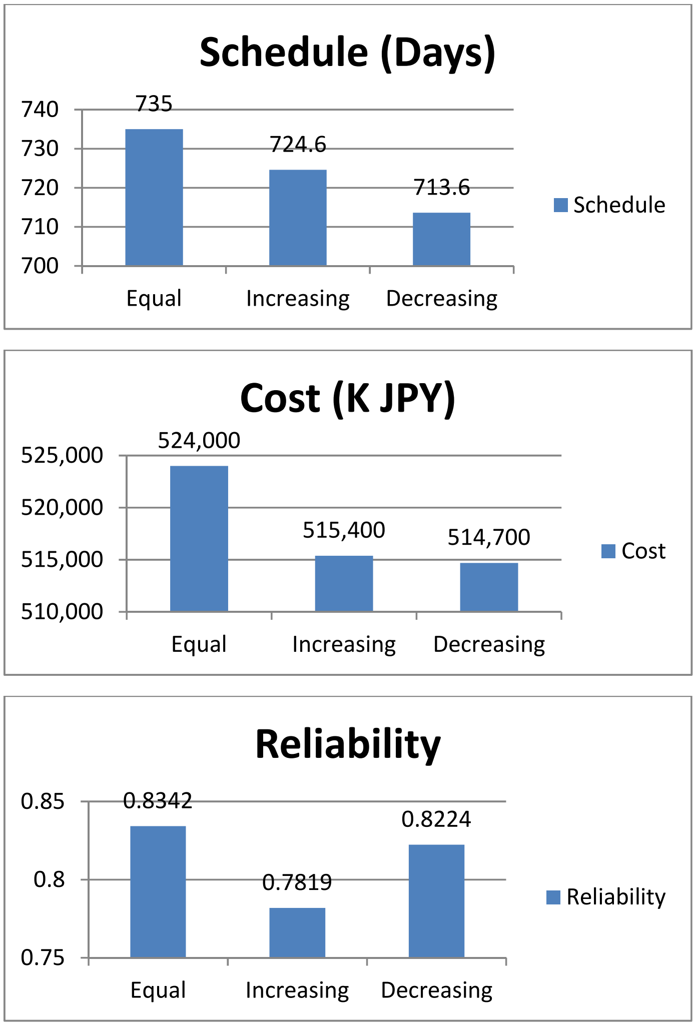


Figure 23, Comparison between the cost schedule and reliability in equal, increasing and decreasing weight processes for the full DSM.

The simulation was run on the three cases, equal, increasing and decreasing processes weights. Figure 23 shows the results in terms of the mean values. The lowest cost and schedule, while achieving high reliability, was at the decreasing process weight model. In that model the cost to improve reliability with a certain percentage is lower than the cost needed in the case of equal and increasing models. The same fact applies to the case of schedule. This means that the project reliability can be improved while not pushing the incurred cost and schedule too high.

G.6 Conclusion

This study presented the effects of rework on the Nanosatellite project key parameters. The effect of rework on schedule, cost and reliability, is random in its nature. Therefore a model based on Monte-Carlo simulation was developed to study the effects. The simulations showed that rework do affect the project reliability. The more rework we have the higher the reliability would be. Rework has an adverse effect as well. It leads to increasing the schedule of the project as more time is needed to re-do the previously done activities. The re-doing affects the project cost. As the cost is tied to the schedule in a direct forward proportionality, the increment in the schedule leads to increment in cost as well. The statistical results showed that the increment in schedule and cost due to rework corresponds to increment in the project reliability. The seriousness by which we handle the project processes was modeled using a weight function. The simulations showed that, although the probable rework was distributed in three main phases in the DSM, the beginning, the middle and the end, still higher reliabilities can be achieved with the decreasing weight function. The decreasing weight means we focus seriously on the early stages. This result is proven by the numbers in the statistical tables of the simulations. If we focus on the early specification stages then all of the project would progress with higher reliability. The project cost saturates with the reliability as we reach the project end. It means that the processes at the end of the project, which are of higher cost than the processes at the beginning, due to its nature such as building of the expensive flight model, cannot afford to raise the project reliability. We cannot correct all of the mistakes at the last moment. So it is of limited effect on the project. The bending in the cost-reliability curves is due to the overlapping in executing the tasks and rework effects on the schedule and the cost of the project. The simulation model presented can be very useful in project planning. It can be used to estimate the expected schedule, cost, and reliability of a Nano-satellite project with a set of iterative processes. The model can be used to assess the project risk management by calculating the profit-lost values. The significance of the study is that it merges the DSM approach with the project planning concepts and applies this to space based projects.

References

1. Tyson Rodgers Browning, Modeling and Analyzing Cost, Schedule, and Performance in Complex System Product Development, Doctor of Philosophy in Technology, Management, and Policy, MIT, 1998.

2. Ali A.Yassine, “An Introduction to Modeling and Analyzing Complex Product Development Processes Using the Design Structure Matrix (DSM) Method“, Quaderni di Management (Italian Management Review), No.9, 2004.
3. Steven D. Eppinger and Tyson R. Browning, “Design Structure Matrix Methods and Applications”, MIT Press, June 2012, USA.
4. Chun-Hisen Chen, Shih Fu Ling, and Wei Chen, “Project Scheduling for Collaborative Product Development Using DSM“, International Journal of Project Management, Vol.21, 2003.