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

OPTIMIZATION OF A NANO-SATELLITE COMMUNICATION SYSTEM USING A SOFTWARE DEFINED RADIO (SDR) PLATFORM IMPLEMENTATION STRATEGY

By
RAFAEL ARMANDO RODRÍGUEZ LEÓN
ID: 16595905

Thesis Supervisor
PROF. KENICHI ASAMI
Department of Integrated Systems Engineering

Graduate School of Engineering, Kyushu Institute of Technology
Kitakyushu, Japan
2020

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

OPTIMIZATION OF A NANO-SATELLITE COMMUNICATION SYSTEM USING A SOFTWARE DEFINED RADIO (SDR) PLATFORM IMPLEMENTATION STRATEGY

By
RAFAEL ARMANDO RODRÍGUEZ LEÓN
ID: 16595905

A dissertation submitted for in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering

Thesis Supervisor
PROF. KENICHI ASAMI
Department of Integrated Systems Engineering

ACKNOWLEDGMENT

I would like to thank to my supervisor Professor Kenichi ASAMI for allowing me to continue being a part of his laboratory for my PhD studies. I want to express my gratitude for his support, for his guidance, respect, kindness and patient along my formation and education process and during this research period.

I would like to express my profound gratitude to the Professor Kei-Ichi OKUYAMA for giving me the opportunity to join for the Ten-Koh project. His help and his support during my PhD studies were so important to reach fruitful results. I got many important knowledges which I could apply during this research and which I hope to apply in my future professional career.

Appreciations to all professors who taught me many tools in their classes useful in my professional life and for my research career, especially to the Professor Mengu CHO for his impeccable work and leadership converting the SEIC course one of the most relevant programs among the space engineering community. I am so proud to have been part of it.

I would like to thank to all my colleagues in the Kyushu Institute of Technology, in first place to all Asami's laboratory members (especially Mohamed Elhady, Amar, Tugi, Cosmas and Yasir). To all Ten-Koh developers team (especially Isai FAJARDO, Jesus GONZALEZ, Rigoberto MORALES, Sidi BENDOUKHA) and to all LaSEINE members. I learned many valuable things from them and I received support from them when I needed, both in my personal and professional life.

Finally, I deeply appreciate the love and support of my parents Elizabeth and Armando that even from the distance always have been cheering me up and helping me to overcome the difficulties presented in my life. A very special mention to Kaori, who always was by my side helping and supporting when I had difficulties during my life in Japan.

ABSTRACT

In nano-satellite missions, Software Defined Radios (SDR) have been widely used in the implementation of communication subsystems in order to increase the flexibilization both in the space segment and on the ground stations. Also, Commercial Off-The-Shelf components (COTS) are widely used to develop subsystems for nano-satellite missions in order to reduce development costs and because those are relatively easy to purchase especially for developing countries. However, COTS components are not space-certified and it becomes a problem when satellites are wanted to be used in high reliability missions. An example of that is Ten-Koh, a Low Earth Orbit (LEO) environment observation satellite developed in the Kyushu Institute of Technology in Japan, in which one of the top-level mission requirements was to re-use as much as possible the components utilized on a previous successful mission (Shinen-2) in order to mitigate the failure risks by using non-certified/non-space heritage components and to decrease the development time following the lean satellite design methodology.

In this research, an SDR implementation for the space segment is proposed in order to optimize the communication system designed for Ten-Koh satellite. The proposed implementation consists of the integration of two COTS modules (a single-board computer with a radio frequency module) using embedded Linux, Python and GNU radio developing tools. The purpose is to demonstrate that the proposed system can be used safely in future satellite missions overcoming the design constraints, limitations and issues experimented during the Ten-Koh design and operation phases showing the improvements in terms of performance, flexibility, cost and development time.

In addition to above, this research shows the on-orbit issues presented in the Ten-Koh mission due to the radiation effects and describes the facilities, equipment, methodology and results of a radiation test performed for the main processor used in the Ten-Koh mission and for the single-board computer used in

the proposed SDR system in order to find the possible causes of the failures presented on-orbit and to compare the results for verifying if the proposed system can be used safely in the radiation environment on LEO orbit.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1 - INTRODUCTION

1.1. Software Defined Radio (SDR)

A Software Defined Radio (SDR) is a communication system in which several of its components (typically filters, modulators, demodulators, tuners, etc.) can be implemented by software merely, instead than using a fixed hardware. It offers to the developers a very flexible wireless communication platform in comparison with the typical communication systems implemented entirely by hardware. In satellite communications, SDR systems have been used for minimizing the costs and give flexibilization in the ground station implementations. An example of it is the SatNOGS project, which is a complete satellite ground station network platform. This project offers the possibility to build a Do It Yourself (DIY) / fully capable ground station for receiving satellite data and for joining in a global ground station network [1]. Currently, it is possible to build a basic functional ground station using an ODROID U3 module or a Raspberry Pi 3 module in conjunction with some open libraries implemented in Python and GNU Radio platform. SDR systems has been also used to develop communication subsystems in the space segment. Also, it is possible to find several SDR platforms available in the market e.g. GomSpace SDR, URSP, LimeSDR and FunCube among others [2].

The key parameters that can be configurable via software are:

- Modulation schemes
- Transmitter and receiver frequency
- Transmitter power output gain
- Tunable Filters
- Tunable codification/packet processing
- Tunable sample rates

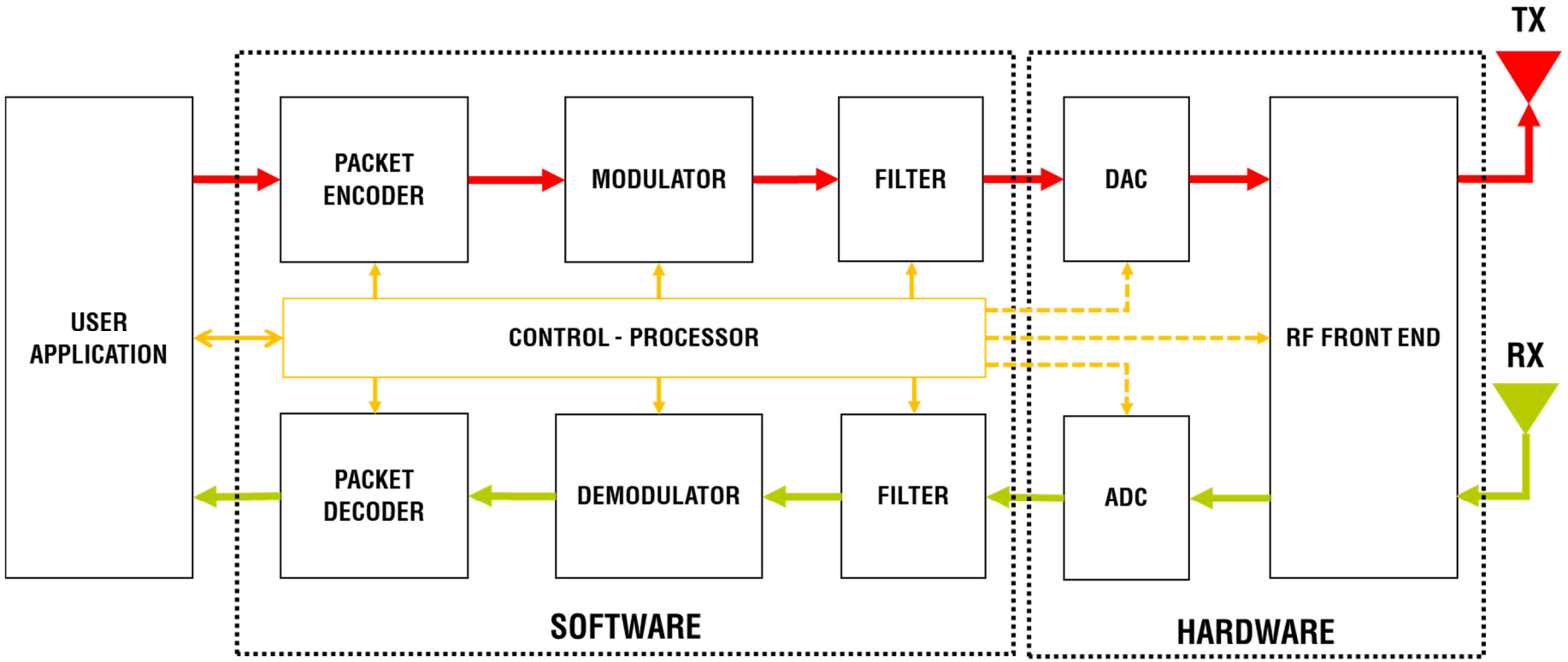


Figure 1 – General SDR architecture block diagram.

The SDR block diagram architecture is shown in the Figure 1. It is possible to observe that the SDR is divided in two main parts, the software part and the hardware part.

The software part usually consists in a processor, FPGA (Field-programmable gate array) or DSP (Digital Signal Processor) in charge of the following functions:

- Controls of the data flow between all SDR block components.
- Performing the data packet encoding/decoding.
- Performing the digital modulation/demodulation of the encoded/decoded data.
- Filtering the data transmitted and received from the hardware part.
- Controls the RF front-end module parameters (e.g. the frequency of the transmitter and the receiver, the power output of the transmitter and the sensitivity of the receiver).
- Processing the instructions that come from the user application and configure the respective SDR components according the received parameters.

The hardware part consists in the Digital to Analog/Analog to Digital converters (DAC/ADC) and the RF front-end module which is in charge to receive the FM signal (in MHz or GHz), demodulate it to the baseband frequency (in kHz) and vice versa.

1.2. SDR in the space

In [3] the author provides a wide overview, a detailed characteristics and advantages of using a software radio architecture in communications systems since its conception in 1995. As shown in [4], the grow of the terrestrial applications of SDRs has been exponential as shown in the Figure 2 which has resulted in an evolution on the hardware and software capabilities as well as the reduction of the size, mass and power consumption in comparison with the

traditional radio architectures. It became the SDRs a great option to be used in satellite applications. As a result of that, SDR systems started to be included as payload missions for bigger satellites as the Mars Reconnaissance Orbiter (MRO) and the NASA Space Communications and Navigation (SCAN) Testbed [5] in 2010 and 2013 respectively.

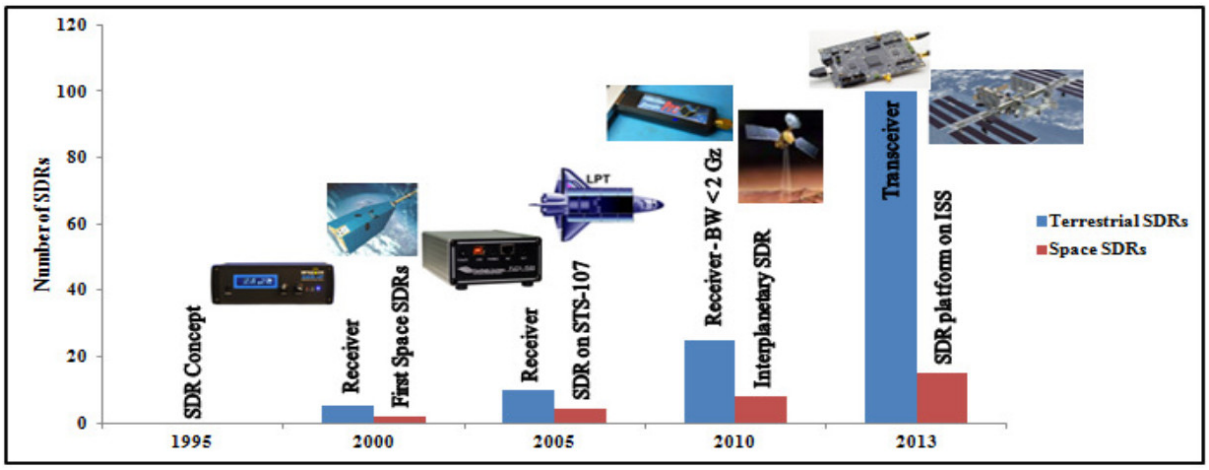


Figure 2. Evolution of terrestrial and space software defined radios [3].

However, the current trend of the small satellite missions (nanosatellite) has been increasing slightly as shown in [6] reaching until 236 nanosatellite launches in 2019 and it becomes a challenge to develop SDR platforms capable to overcome the constraints presented in this type of missions like the reduced size, mass and especially low power consumption, aspects that in terrestrial applications are not so relevant and that is why terrestrial SDR solutions do not provide currently. Also, most of these nano-satellite missions must to be developed in a very limited cost which force to the developers to use Commercial off-the-shelf (COTS) components on their implementations, aspect that commercial SDR systems available in the market lacks.

In the Table 1 based on [4], it is possible to observe clearly the key differences between the traditional radio and SDR architectures. As shown, the challenges to face in the development of an SDR architecture are to reduce the power consumption, to develop a reliable software and the use of adequate processing

units (e.g. FPGA, DSP or microcontroller).

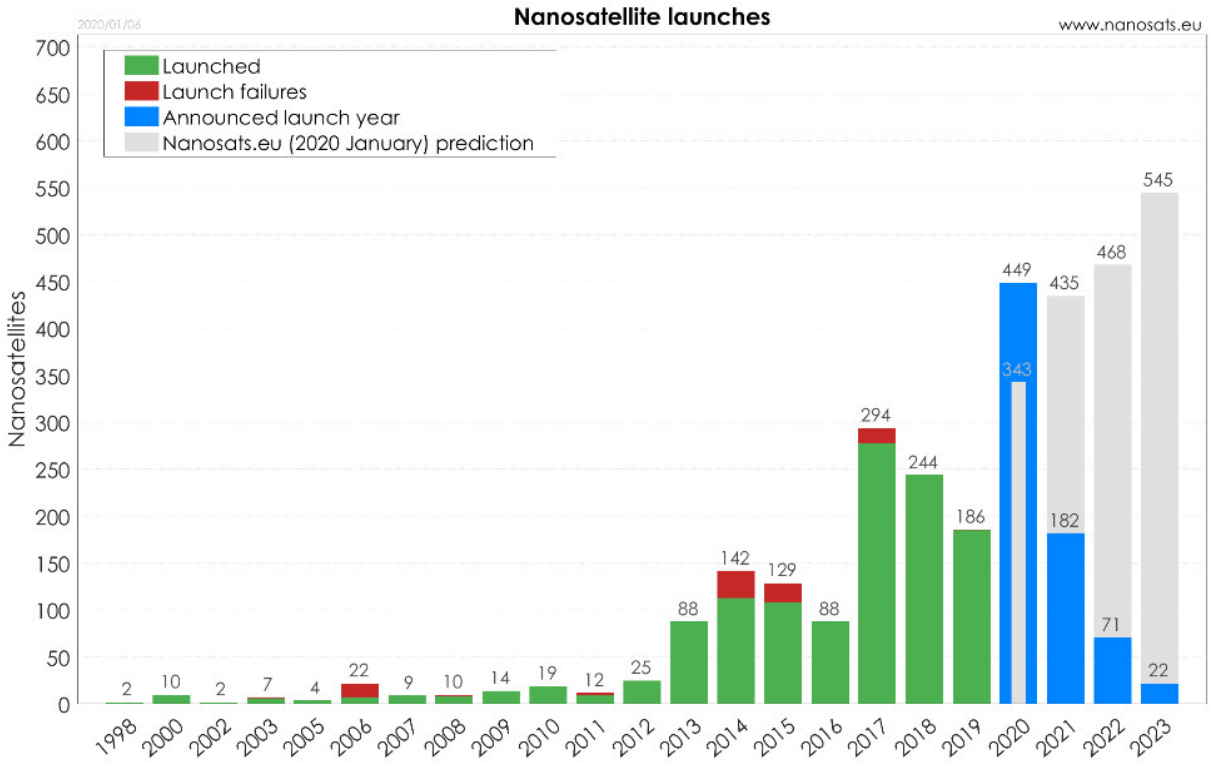


Figure 3. Nano-satellite launches by year [6].

Table 1. Traditional radios vs. SDR architectures [4].

	Traditional radios	**SDRs**
Pros	• Limited processing and the selection of processor / controller / ADC is less critical. • Cheap and readily available	• Flexible design: Multi-band / multi-mode. • Software based reconfigurable platform • Upgradable during mission lifetime
Cons	• Fixed design: Single-band / single-mode. • Complexity in hardware • More analogue components • Cross talk between the narrow bands due to aging	• Complexity in software • Vulnerable to software threats • Faster FPGAs and DSPs and larger bandwidth ADCs are required • Power Consumption

1.3. SDR platforms available in the market

Currently, there are several SDR platforms available in the market as

presented in [2]. In the Figure 4, it is possible to observe the relation between the cost and the mass of the available hardware systems. Platforms as the GomSpace SDR are very expensive because they offer space-proven products, also, platforms like the USRP or the EPIQ are not suitable for this research purposes due to the mass and cost constraints. However, platforms like the LimeSDR or FunCube are more suitable for low cost and mass constraint missions.

On the other hand, in the Figure 5, the channel bandwidth and frequency characteristics are shown. In this case, the FunCube and the RTL-SDR platforms have limited characteristics in comparison with the LimeSDR options.

Taking into account the mentioned above, the LimeSDR options are the more balanced options for the SDR implementation and for that reason the LimeSDR mini is the option chosen for the implementation of the SDR architecture proposed in this research.

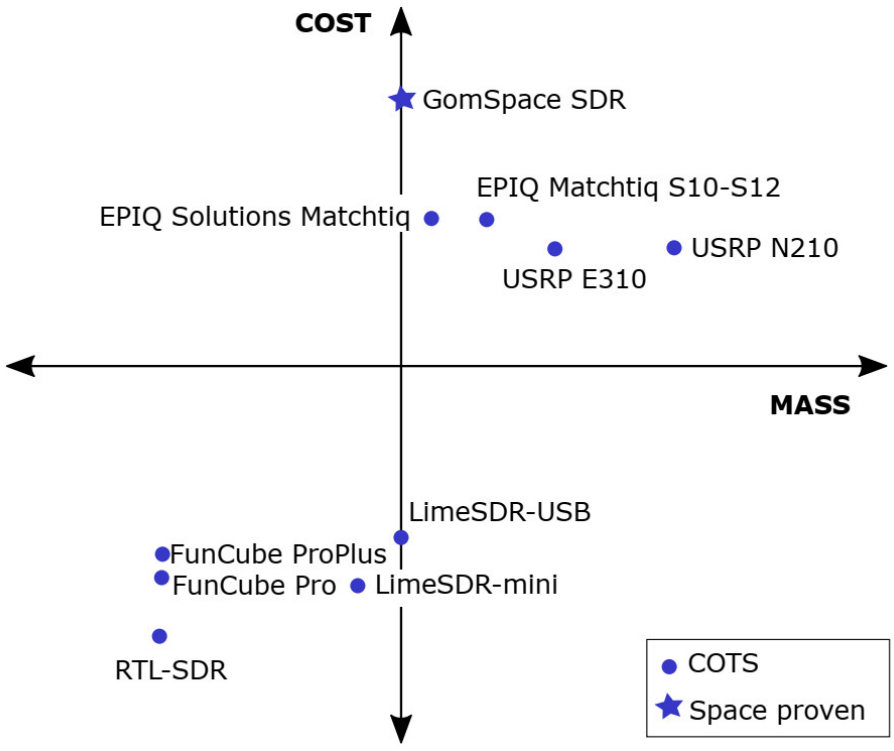


Figure 4 - SDR platform overview (cost vs mass) [2].

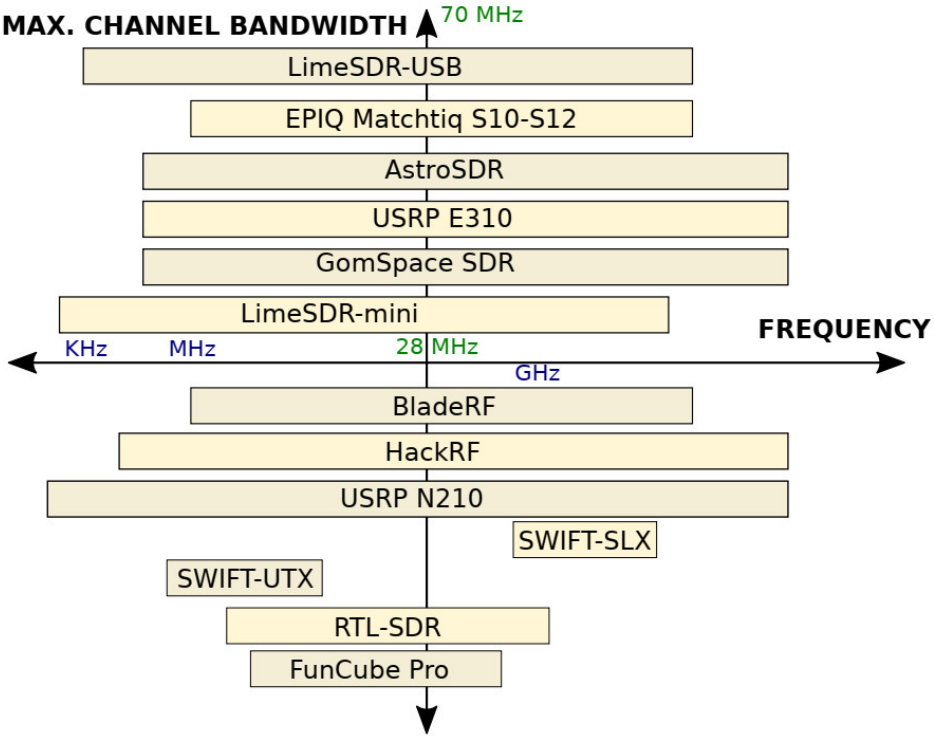


Figure 5 - Maximum channel bandwidth vs. Frequency bands in SDR platforms [2].

1.4. COTS based SDR in satellite applications

In this section a survey of the SDR systems currently available on the market or already developed for nanosatellite missions in the last years is presented. The idea is to analyze the key characteristics, performance parameters and software/ hardware platforms used for their development in order to get a clear idea about the current state of the art about SDR systems for nanosatellite applications.

1.4.1. Previous studies/publications

Searching in the existing literature, the following studies and implementations about SDR in nano-satellite applications are presented due to their relevancy in this research and similarity. At the end in the Table 2, a summary of the key parameters and characteristics are presented in order make a comparison.

The first relevant study is an SDR implementation for CubeSat presented in [7]. The hardware platform consists in a single board that includes an Analog Devices ADSP-BF537 Blackfin DSP as processing unit. The RF module is divided

in two parts, the former which consist in a baseband Signal Front-End chip (Analog Devices AD9863) with digital to analog converters (DAC) and analog to digital converters (ADC) included, the latter is an RF daughter board which is not specified (just the author mentions that it could operate in VHF, UHF and S bands). Regarding the software, the author mentions that National Instruments LabView is used to create the software for the DSP module using the Blackfin embedded module. The study was published in 2008.

In [8] an SDR implementation for CubeSat based on an FPGA device is presented. The hardware platform consists of a single board designed from scratch which includes the processing unit, a Xilinx Spartan3A-1400 and the RF circuits. There are not details about what chip or device was used for the RF circuits, however, the author mentions that the hardware is largely similar to an USRP XCVR 2450 transceiver. Regarding the software, the author mentions that is developed using VHDL (Very High-Speed Integrated Circuits Hardware Description Language) to implement an embedded MicroBlaze processor used for controlling all the SDR modules and to modify the URSP libraries developed and provided by Ettus Research for their RF boards with the purpose to make them compatible with Xilinx devices. The system requires an external processor running Linux and GNU Radio tools in order to be controlled and operated. The study was published in 2012.

In [4], the results of two SDR implementations proposed as a communication system for a CubeSat constellation are shown. Regarding the hardware platform, the first testbed includes a SmartFusion2 board acting as a processing unit which include an ARM Cortex M3 with an FPGA fabric included in the same device and a FunCube Dongle as an RF module. The second testbed includes a ZedBoard which includes a Xilinx Zynq 7020 SoC (System on a Chip) acting as processing unit and a Lime Microsystem's Zipper + MyriadRF boards as RF modules. In the case of software platform, the first testbed was configured and controlled using a Linux operating system running the FunCube dongle utilities (FCD

Control). The second testbed was configured using Linux operative system running GNU Radio and the modules implemented in the FPGA fabric were generated using the Xilinx Processing System (XPS). As conclusion, the author mentions that the second testbed is the most suitable option for their application because the first testbed lacks of a direct interface to the received IQ signal, the USB interface creates a speed limit to 480 Mbps and it lacks of transmission functionality. The study was published in 2014.

The last relevant publication is [9] in which an SDR implementation of the satellite and ground segments are described. The hardware platform consists in an FPGA board which includes an Altera EP3C25E144I7N FPGA as processing unit and two independent boards for the receiver and receiver working in half-duplex mode. Those RF modules were developed completely from scratch (there are not details about which RF chips they used in the design). The software platform was designed in VHDL using the Quartus development tool provided by Altera and it does not run any operative system. The study was published in 2016.

Finally, in the Table 2, a summarize of the key performance parameters reported in the mentioned research studies are shown.

Table 2. Noncommercial SDR key performance parameters summary.

Study Reference	**Key Parameters**
[4]	**Modulation schemes:** BPSK, QPSK, 16 and 32 QPSK **Processor:** Xilinx Zynq 7020, **RF Front End:** LMS6002D **Frequency:** 914 MHz (1st Generation), 26.1 GHz (2nd Generation) - Fixed **Power Consumption:** RX=1.2W, TX= 2.5W, Total = 3.7W **Power Output:** 30dBm (1W) **Software:** Linux + GNU Radio
[9]	**Modulation schemes:** FSK **Processor:** Altera EP3C25E144I7N, **RF Front End:** Not specified **Frequency:** 433.92 MHz - Fixed **Power Consumption:** RX = 0.7W, TX = 2W, Total = 2.7W **Power Output:** Not specified. **Software:** VHDL + Quartus development tools
[7]	**Modulation schemes:** Not specified **Processor:** Analog Devices ADSP-BF537, **RF Front End:** Not specified

	Frequency: VHF - UHF **Power Consumption:** 165mW (only the processor unit) **Power Output:** Not specified **Software:** LabView
[8]	**Modulation schemes:** Not specified **Processor:** Xilinx Spartan3A-1400, **RF Front End:** USRP XCVR 2450 **Frequency:** 2.4 - 2.5GHz, 4.9 - 5.9GHz **Power Consumption:** Not specified **Power Output:** Not specified. **Software:** Modified USRP libraries and Linux + GNU Radio for controlling.

1.4.2. Commercial SDR available on market

At the time of this survey, the commercial SDR platforms available in the market are:

GomSpace NanoCOM SDR [10]**:** GomSpace is a manufacturer and supplier of nanosatellite parts and buses located in Denmark with experience since 2007. Their SDR is a PC-104 form factor, space-proven platform that includes as a processor unit one Xilinx Zynq 7030 programmable SoC which includes a dual ARM Cortex A) running at 800MHz plus an FPGA logic in a single chip. The device is on-orbit programmable and runs Linux operative system plus a proprietary software developed for themselves. The RF hardware is not detailed since it is a proprietary design.

Vulcan Wireless INC. SDR [11]**:** Vulcan Wireless INC. Is a provider of digital communication solutions for terrestrial and space applications, located in the USA. They offer two options of SDR, one for S-Band and the other for UHF frequencies. Both of them have a CubeSat form factor with space heritage and compatible with NASA Near Earth Network (NEN). Unfortunately, in their web page there is not a datasheet available (probably can be obtained through a quotation), then, there are not details about the hardware they utilized in their devices.

Tethers Unlimited, Inc. – SWIFT SDRs [12]**:** Thethers Unlimited INC. is a company that provides space services including satellite parts. They offer

SDR platforms in L, S, X, K and UHF frequency bands with configurable BPSK/QPSK/OQPSK/8PSK and 16PSK modulations. The form factor of the radios is 0.25U (a quarter of CubeSat standard). In their datasheets is not possible to find information about the hardware used in their devices.

Allen Space TOTEM [13]**:** Allen Space is a spin-off company from the University of Vigo's Xatcobeo satellite, the first Spanish satellite. TOTEM is a PC-104 form factor SDR based in the Xilinx Zynq 7000 SoC running embedded Linux with a GNU Radio support. The frequency is configurable from 70MHz to 6GHz with up to 56MHz of bandwidth. It offers safe in-orbit software updates, flight heritage and the front-end RF module is provided as an additional piggyback board.

In the Table 3, a summary of the relevant features of the mentioned commercial SDR platforms available in the market is presented.

Table 3. Commercial SDR key performance parameters summary.

SDR	Key Parameters
GomSpace - NanoCOM	**Modulation:** TDD (Time Division Duplex), FDD (Frequency Division Duplex) **Processor:** Xilinx Zynq 7030, **RF Front End:** Non-specified **Frequency:** 70MHz – 6GHz, **Power Consumption:** TX = 3W. **Software:** Linux and proprietary software
Tethers - SWIFT	**Modulation schemes:** Non-specified **Processor:** Altera EP3C25E144I7N, **RF Front End:** Non-specified **Frequency:** L, S, X, K and UHF bands **Power Consumption:** 15W total **Power Output:** Non-specified. **Software:** Non-specified
Allen Space - TOTEM	**Modulation schemes:** Depends of piggyback RF front end daughterboard. **Processor:** Xilinx Zynq 7000, **RF Front End:** Non-specified **Frequency:** 70MHz to 6GHz **Power Consumption:** 4W total **Power Output:** Not specified **Software:** Linux and GNU Radio support.

1.5. Processing Modules Description

1.5.1. Raspberry Pi

The Raspberry Pi is a family of devices created in the United Kingdom by the Raspberry Pi foundation in order to promote the teaching of basic computer sciences in schools, universities and developing countries. The hardware of those modules is open for everyone, but the firmware is closed source.

The first Raspberry Pi generation (Pi 1) was released in 2012 in two models, the model A and the model B, the difference between them is that the second one has higher performance specifications. Since that time, four families have been released with higher performance changes but keeping the same size and price.

The Raspberry Pi is one of the most popular single-board computers nowadays, it has been used in several educational, academic and scientific projects. It has a huge community that develops modules and open source code to handle several kinds of devices, sensors, screens and other peripherals. Also, the community is very enthusiastic to develop several kinds of applications with many functions to be used easily on the device.

In the hardware part, the community and some companies have developed several compatible modules that can be connected easily to the board, for example, the camera module is one of the most famous compatible hardware peripherals, it is easy to integrate to the main module and it is possible to find a variety of open source image processing software to be used with it.

In the Table 4, a comparison between the most popular Raspberry Pi modules is presented where it is possible to check the main features of each module in order to compare the pros and cons between the existing modules in the market.

For this research purposes, the Zero and the 3B+ models were chosen and the idea is to compare the benefits and disadvantages to use the high performance and a budget options available currently in the market.

Table 4 - Raspberry Pi modules main performance parameters.

	R-Pi 2B v1.2	**R-Pi 3B+**	**R-Pi ZERO**
CPU	64 bits Quad-core ARM Cortex-A7	64 bits Quad-core ARM Cortex-A53	32 bits Single-core ARM1176JZF-S
FREQ (MHz)	900	1400	1000
RAM (MB)	1024	1024	512
STORAGE (GB)	Up to 32 microSD	Up to 32 microSD	Up to 32 microSD
PERIPHERALS	17×GPIO, 2×I^2C, 2×UART, 1×SPI, 1xEthernet, 4xUSB-B and Camera	17×GPIO, 2×I^2C, 2×UART, 1×SPI, 1xEthernet, 4xUSB-B and Camera	17×GPIO, 2×I^2C, 1×SPI, 2×UART, PCM and PWM
SIZE (mm)	85.6 × 56.5	85.6 × 56.5	65 × 30
PRICE (USD)	$35	$35	$5

1.5.2. FPGA (Field-programmable gate array) module

An FPGA is a programmable device designed to be configured by the end-user after its manufacturing. It differs with the microprocessors in the fact that the hardware in the device can be modified as the application needs, making these devices a very flexible option used especially for parallel processing and digital signal processing. Xilinx, Inc. is a very well-known manufacturer of FPGA devices which includes a wide portfolio of several families divided by its application purpose and performance.

The Spartan is a well-known family of FPGAs for applications where the low cost, low power and high volume are the important targets. It is the most basic Xilinx FPGA and for that reason is most typically used. Those are built using the 45nm, nine metal layers, dual oxide process technology. The most common applications for that family is automotive, wireless communications and video surveillance.

Another set of families is the 7 series consisting of the Artix, Kintex and Virtex devices. Those families have better performance than the 6 family, however its cost is higher. The Artix family has 50% lower power consumption in comparison with the Spartan family and it can deliver the performance required to address cost-sensitive, high-volume markets previously served by ASICs (Application-Specific Integrated Circuits), and low-cost FPGAs. However, its price is still

higher than the Spartan families. Because of the low-power consumption improvements, the target applications for the Artix family are for portable equipment, military and avionics communications.

Table 5 - Xilinx FPGAs performance comparison.

	SPARTAN 6	**ARTIX-7**	**KINTEX-7**	**VIRTEX-7**
Logic cells	147K	215K	478K	1,955K
Block Ram	4.5Mb	13 Mb	34 Mb	68 Mb
DSP slices	180	740	1,920	3,600
Peak DSP performance	930 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
Transceivers	4	16	32	96
Peak Transceiver speed	3.2 Gb/s	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Peak Serial Bandwidth	51 Gb/s	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	X2 Gen1	x4 Gen2	x8 Gen2	x8 Gen3
Memory Interface	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O pins	500	500	500	1200
I/O voltage	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V
Supports partial reconfiguration	Not natively	Yes	Yes	Yes
Price	$400 USD	$880 USD	$980 USD	$7000 USD

The Kintex family is the mid-range FPGAs from Xilinx, It has 50% lower power consumption than the Virtex 6 family with similar performance. It can be used to cover applications that needs 12.6 Gbit/s or 6.5 Gbit/s serial communication, enough memory and logic performance needed in optical communications. It can provide a good balance between processing performance, power consumption and costs.

Finally, the Virtex which is the Xilinx high-end family, typically this family integrates FIFO (First Input First Output) logic, DSP, Ethernet blocks and high-speed transmitters additionally to the normal FPGA logic. Additionally, it can include embedded hardware functions like multipliers, memories, serial transceivers and processor cores in order to facilitate the application development.

The Virtex family usually is used for wired and wireless communication measuring instruments, medical equipment and defense systems.

In Table 5, a performance and features comparison table is shown. For this research purpose, the Spartan 6 and the Kintex 7 devices are the chosen options in order to compare the performance of two different generations of Xilinx FPGAs.

1.6. Single Event Effects (SEEs)

When charged particles such as electrons, protons or heavy ions pass through semiconductor material like silicon, they lose energy in two ways: electronic loss and nuclear interactions. As the particle travels along the sensitive volume in the device, the energy loss appears as a cloud of electron-hole pairs [14]. In the case of heavy ions, different failures in semiconductor devices occur as a consequence of the direct ionization created in the device (the charge deposited by a single particle in the sensitive volume of the device being irradiated).

Other sources of energetic particles that can produce SEEs are protons and neutrons. These particles can produce SEEs by nuclear reactions when the direct impact of the incident particle energy to a recoil atom transfer in the form of an elastic/inelastic collision or spatial mechanism. Depending on the orbit designated for a spacecraft, protons can be the main source of SEEs: "the probability of such reactions are low (approximately 10^{-5} for most devices of interest), however, fluxes of protons can be very high in the inner proton belt or during solar particle events and this mechanism can dominate the SEEs rates in many situations for modern devices that have a low Linear Energy Transfer (LET) threshold" [15]. Protons and neutrons generate secondary products that bring additional energy and charge deposition in the sensitive volume.

The injection of charge occurs in a very short time scale (in the order of picoseconds), and in small amounts (10^{-14} *to* 10^{-12} coulombs), however, these magnitudes are enough to disrupt temporarily or permanently the operation of

electronic devices [14].

Depending on the resulting effect on the device, temporal or permanent, the SEEs are divided into two main groups:

- **Non-destructive SEEs:** The resulting effect disables temporarily the device and its operation can be recovered after some time. Examples of those effects are Single event transient (SET), single event disturbs (SED), single event upset (SEU), multiple-cell upset (MCU), single-word multiple-bit upset (SMU), single event functional interrupt (SEFI) and single event hard error (SEHE).
- **Destructive SEEs:** The resulting effect on the device disables it definitively from operating. Examples of those effects are Single event latch-up (SEL), single event snapback (SESB), single event dielectric rupture (SEDR), single event gate rupture (SEGR) and single event breakdown (SEB).

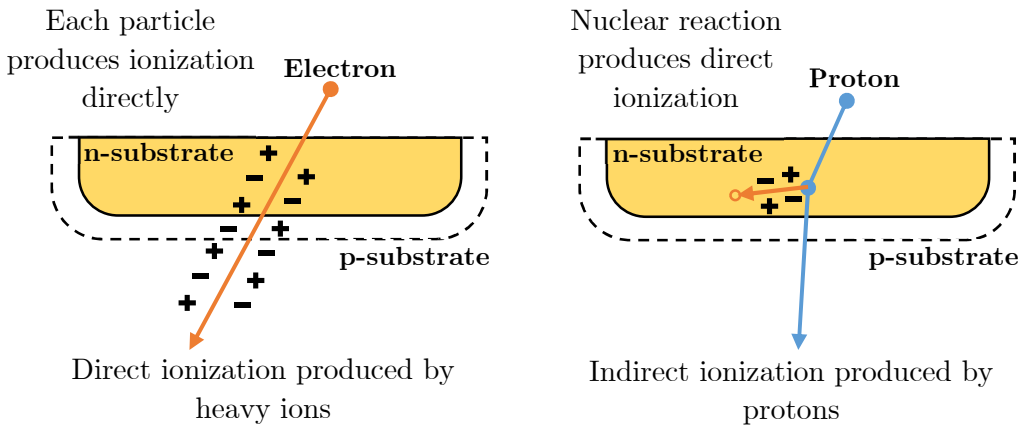


Figure 6 - SEE produced by heavy ions (electrons) and nucleons (protons and neutrons) in a semiconductor device.

For the most common digital devices such as processors, microcontrollers and solid-state memories used for space applications, extended tests for SEE (SEU and SEL) are required. Depending on the complexity, technology and

manufacturing process of the device, a particular test can be designated for performing an evaluation through ground testing. Figure 6 shows the SEE produced by heavy ions and protons.

1.6.1. Single Event Upset (SEU)

Single event upsets are non-destructive SEEs that affect mainly digital devices such as memories, registers, latch devices and solid-state recorders (e.g. SRAM, EEPROMs and FLASH devices). Those events are registered as a bit-flip leading to the change in the stored information or state in the device.

Similar to SEUs, Multiple-Cell Upsets (MCUs) or Multiple-Bit Upsets (MBUs) occurs in memories, registers and latch devices when a single particle impacts them affecting several adjacent bits due to the large particle ranges [15].

1.6.2. Single Event Latch-up (SEL)

Single event latch-up affect semiconductor devices by creating a path of low impedance between the power supply rails of a device due to the creation of an ionized path along the particle trace in the device. In CMOS devices, an SEL makes the transistors to enter conduction due to a forward biased state from the ionization created by heavy ions or protons. This increases the current consumption of the device to a level that can destroy it due to a current avalanche effect. A power reset is required to remove the SEL condition, however, if the system is not able to detect and provide a power reset after the SEL occurred, the device may end with a permanent damage.

1.6.3. Linear Energy Transfer (LET)

The linear energy transfer refers to the deposited energy per unit path length. For SEE analysis, the units of the LET are $[MeV \cdot cm^2/g]$ or $[MeV \cdot cm^2/mg]$. The LET is also related to the stopping power, which is the energy loss per unit path length by a particle in a medium. The LET is can be described by the

Equation 1 [15]:

$$LET(x) = \begin{cases} \frac{1}{\rho}\frac{dE}{dx}(x), & 0 \le x \le r \\ 0, & otherwise \end{cases}$$

Equation 1 - Linear Energy Transfer (LET).

Where E is the energy of the particle, x is the range of the particle, and ρ the density of the target material. For silicon, the value of ρ is $[2.32\ g/cm^3]$. From the values of the LET, the deposited energy over a distance d can be computed by Equation 2 [15] if the LET remains constant over the distance:

$$E_{dep} = LET \cdot d \cdot \rho$$

Equation 2 - Deposited energy over a distance.

The LET concept is important for the testing and evaluation of SEEs due to heavy ions. Depending on the ion source and energy, different LET values can be achieved in a target device, which for most of devices of interest is silicon (Si).

1.6.4. Device cross-section

The cross-section of a device refers to the probability of SEEs to occur. It is measured as the number of events recorded per unit of particle fluence. The fluence Φ (Equation 4 [15]) for a specific type of particle is the integral of the flux (unidirectional integral intensity flux represented j in Equation 3 [15]) over a given interval (e.g. one minute, one hour, one day):

$$j_{\geq E} = \int_{E}^{\infty} j\, dE$$

Equation 3 - Unidirectional integral intensity flux.

$$\Phi = \int_{\delta t} j \, dt$$

Equation 4 - Particle fluence.

The fluence has units of $[particles/cm^2]$, while the flux has units of $[particles/cm^2/s]$.

The SEE cross-section for ions is expressed as a function of LET and in energy for protons and neutrons as shown in Equation 5 and Equation 6 [15], respectively:

$$\sigma_{ion}(LET) = \frac{Number\ of\ events}{Ion\ fluence}$$

Equation 5 - SEE cross-section for ions.

$$\sigma_{proton}(E) = \frac{Number\ of\ events}{Proton\ or\ neutron\ fluence}$$

Equation 6 - SEE cross-section for protons.

In the case of ions, the evaluation of the cross-section is straightforward, it measures the sensitiveness of the device as a function of the LET. In the case of protons and neutrons, where the SEE is produced by nuclear interactions, the cross-section interpretation becomes more complex since it incorporates the probability of a nucleon-nuclear interaction and the probability that the nuclear recoil and other nuclear fragments results in charge deposition along the sensitive volume of the device that produces an event [15].

The cross-section is expressed in units of $[cm^2/device]$ for SEEs in general and in the specific case of SEUs, MCUs, and SMUs in $[cm^2/bite]$ or $[cm^2/byte]$. The typical cross-section curve obtained for a particular device for heavy ions is shown in the Figure 7 and for protons is shown in the Figure 8.

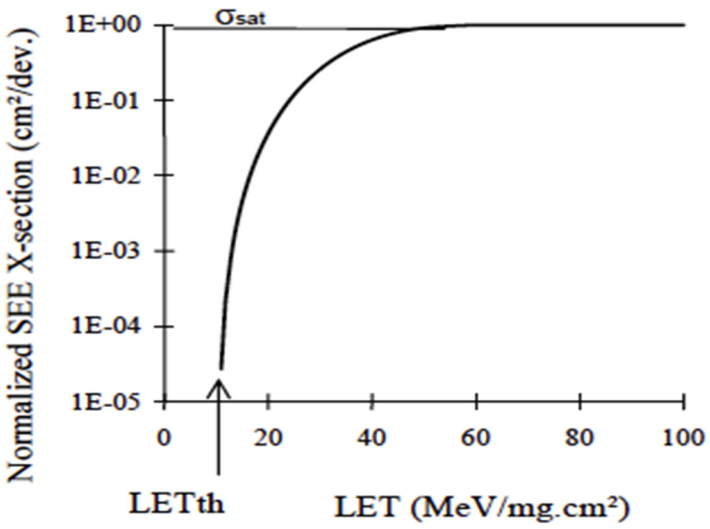


Figure 7 - SEE cross-section as a function of LET produced by heavy ion irradiation [15].

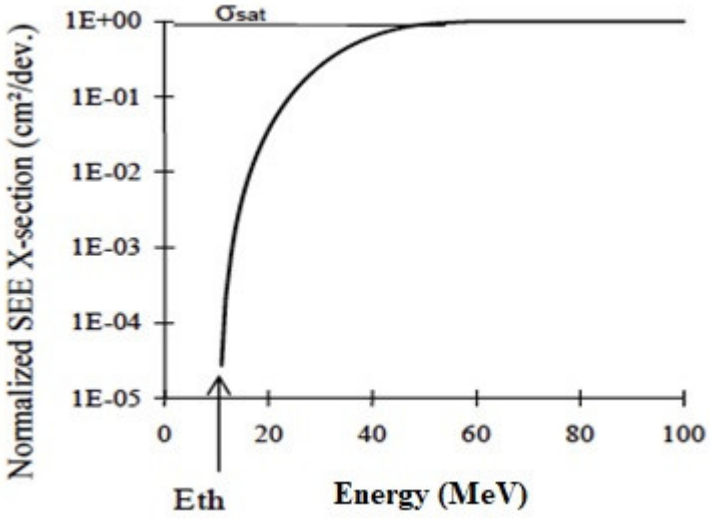


Figure 8 - SEE cross-section as a function of energy for proton irradiation [15].

The most important values to obtain from the cross-section curves are the σ_{sat} which is the cross-section of saturation where the device starts to experiment the same and stable probability of errors and the LET_{th} which is the Linear Energy Transfer threshold value where the device starts to experiment errors (E_{th} proton energy threshold in the case of protons).

When a device has a sensitive volume with a larger horizontal dimension compared to the vertical one, the deposited effective LET (LET_{eff}) is expressed

in Equation 7 [15], where θ is the angle of the flux beam hitting the normal surface of the DUT.

$$LET_{eff}(\theta) = \frac{LET(\theta = 0)}{cos\theta}$$

Equation 7 - Effective LET.

If the incident angle is varied, then the effective fluence will vary too, so, the cross-section has to be computed according to Equation 8 [15]:

$$\sigma_{eff}(\theta) = \frac{Number\ of\ events}{fluence \cdot cos\theta}$$

Equation 8 - Effective SEE cross-section.

1.7. Research objectives and outline

The main purpose of this research is to develop an SDR system architecture for improving the Ten-Koh communication system performance and flexibility overcoming the constraints and limitations found in the design, implementation and on-flight operation phases.

The objectives of this research are:

- To demonstrate the feasibility of using a Raspberry Pi module integrated with a COTS RF front-end module for designing a suitable SDR platform for the space segment application.
- To evaluate the performance of the proposed SDR implementation by comparing it with the implemented Ten-Koh communication system.
- To perform a radiation test to verify the possible causes of the issues presented in the microcontroller used in the Ten-Koh mission subsystems and to obtain the radiation tolerances of the Raspberry Pi module in the

space environment on LEO.

The dissertation is divided in 7 chapters as follows: In the chapter 1, the background, motivation, objectives and goals of the research are described.

The chapter 2 describes the Ten-Koh satellite mission overview, the payload data requirements, the system architecture, the constraints, limitations and on-orbit issues found during the satellite design and operation.

The chapter 3 describes in detail the communication system included on-board of Ten-Koh satellite. It includes the design methodology and the detailed description of the hardware and software architectures, analyzing the constraints, limitations and issues experimented with the design and operation phases.

The chapter 4 describes the proposed SDR implementation for optimizing the Ten-Koh communication system in which the design strategy, developing tools and the hardware/software architectures are described in detail showing the advantages and improvements obtained in the mission design phases in comparison with the used in Ten-Koh.

The chapter 5 presents the results of the simulations and test performed for the proposed SDR system and for the Ten-Koh implemented system in order to verify and compare the performance between the two systems.

The chapter 6 describes the radiation test performed for the PIC16F877 microcontroller used in the Ten-Koh subsystems in order to verify if the failures presented in the chapter 2 were produced due to radiation phenomena and for the single-board processor modules used for the implementation of the proposed SDR system in order to define with more clarity if the system can be used safely in future LEO satellite missions.

Finally, the chapter 7 summarizes the main results and conclusions obtained in this research and describes some recommendations for possible future works in related areas.

CHAPTER 2 - TEN-KOH MISSION OVERVIEW

Ten-Koh mission is a 23.5 kg satellite developed by Kyushu Institute of Technology, Japan, in conjunction with the Radiation Institute for Science and Engineering of Prairie View A&M University, Holland-Space LLC and the Space Research and Technology Institute of the Bulgarian Academy of Sciences. Ten-Koh was launched on October 2018 on-board the HII-A rocket as a piggyback payload of the JAXA's Greenhouse gas Observing Satellite (GOSAT-2). The main mission objective is the observation of the LEO environment measuring the radiation effects on the satellite. In order to achieve that, the main payloads are:

- A Charged Particle Detector (CPD), developed by the Radiation Institute for Science and Engineering Prairie View A&M University, TX, USA and the Space Research and Technology Institute - Bulgarian Academy of Sciences. The system includes 8 CMOS (Complementary Metal Oxide Semiconductor) detectors mounted in a cube form factor and one Liulin type detector [16] mounted on the top of the assembly. The CPD allows the measurement of the radiation environment inside the satellite, the detection of the MeV-range electrons and protons in LEO and the investigation of the space environment in the presence of a low solar activity.
- A Double Langmuir Probe (DLP) which is in charge of the characterization of the plasma environment around the spinning spacecraft.

Also, the mission includes the following secondary payloads:

- A material mission device which is in charge of measuring the degradation and thermal expansion of a Carbon Fibber Reinforced Thermoplastic (CFRTP) material covered with three different coatings exposed directly to the space environment.

- An ultra-capacitor system to test a high-density energy storage device for space applications.
- A thermal switch assembly, developed by The University of New South Wales, Canberra (UNSW Canberra). This payload includes one device for demonstrating a thermal control technique in nano-satellites.

Ten-Koh was designed with the lean satellite philosophy (low cost and reduced development time), for that reason, one of the top-level requirements was to use as much as possible the components and architectures used for the previous mission Shinen-2 [17], [18] due to it worked successfully in the presence of a high radiation space environment using COTS components. It was achieved at a high rate; however, it was necessary to change some components because some of them were obsolete, others were not able to purchase in the market and due to Ten-Koh mission complexity was higher than the Shinen-2 mission, it was necessary to change some components in order to meet the additional mission requirements. The main reason to do that was to reduce the risk to have failures generated by the hazardous space environment, using components already used successfully in previous missions, also to reduce the development time re-using some of the previous designs and architectures.

2.1. Ten-Koh system architecture

Ten-Koh system architecture is shown in the Figure 9. It consists in the bus section and the payload section. The bus section includes the Electronic Power System (EPS), the On-Board Computer (OBC), the Communication system (COMM) and the Attitude Determination (ADS). The payload section includes the Experiment Control unit (ECU), the Ultra-Capacitor Experiment Control Unit (UECU) and the Material Mission Experiment Unit (MMECU). All subsystems are interconnected using a single-master multi-slave serial communication bus I2C in which the OBC acts as a master who controls all the

writing and reading transactions, the rest of subsystems are acting as slave units. It means that the Ten-Koh architecture is a star shape architecture due to subsystems are not allowed to communicate each other's directly, all data transactions must pass through by the OBC who is in charge to manage and deliver the data property.

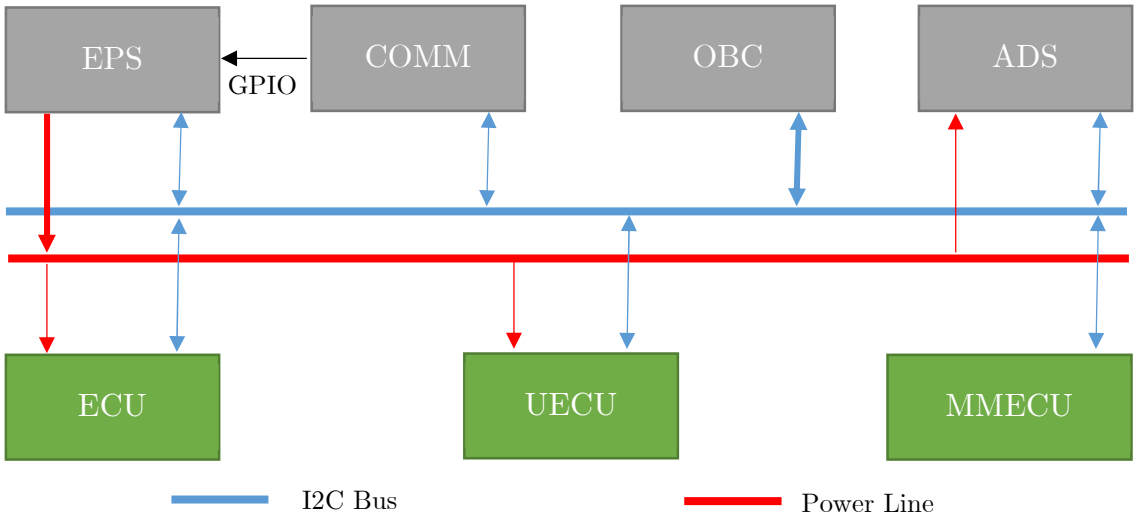


Figure 9 - Ten-Koh system architecture.

Due to the FLASH and RAM memory limitations, the system architecture includes a microcontroller in every single subsystem which basically is in charge to manage and process all local data (e.g. sensor measurements, calculation algorithms, etc.), pack it in a proper way and deliver it when OBC requests. The only direct communication between subsystems is between the COMM and EPS thru a General-Purpose-Input-Output (GPIO) connection in order to provide the possibility to reset the OBC if it fails or hangs sending a direct command from the ground station.

Depending the Ten-Koh operational mode, some subsystems are turned on or off by the EPS subsystem in order to optimize the power consumption and the processing resources of the OBC due to the processor limitations, as was mentioned before, the OBC is in charge to manage and deliver the data to every subsystem and additionally it has to change the operational mode of the satellite

depending of the received command from the ground station or depending the critical housekeeping parameters (e.g. battery voltage, battery temperature, etc.).

2.1.1. Main microcontroller

The microprocessor used in Ten-Koh mission was the Microchip PIC16F877 which was used in several successful missions (e.g. CubeSat-XI-IV and CubeSat-XI-V from Tokyo university [19], Shinen-2 from Kyushu Institute of Technology [17] - [18], and AlcatelSat [20]). The main parameters of this microprocessor are shown in Table 6. The data memory (RAM) of this microcontroller is divided into banks of 96 bytes which it is the maximum buffer allowed to use.

Table 6 - PIC16F877 parameters [21]

Name	Value
Program Memory (FLASH)	14 Kbytes
Data Memory (RAM)	368 bytes
Data memory (EEPROM)	256 bytes
Max Clock Frequency	20 MHz
Digital communication peripherals	1-UART, 1-MSSP(SPI/I2C)
Operating Voltage Range	2 to 5.5 V
Operating Temperature Range	-40 to 85 °C

2.1.2. Data flow architecture

The data into the satellite flows in two ways (uplink and downlink). In the case of uplink, the data is a command which is a data packet with all required information to perform different events in the satellite (e.g. start a payload mission, turn on/off any subsystem, reset satellite, request specific data parameters etc.) In the case of downlink, the data are formed as a telemetry/payload data which is a packet that contains all information about housekeeping and scientific data coming from the satellite. The uplink will be initiated always by the ground station and the data will flow as shown in the

Figure 10. The downlink will be initiated by the OBC due to a requested command from a ground station (complete telemetry request, for example complete sensor report or payload data) via FM link. Of course, a command should be received first, then, the data will follow the same flow as shown before in the uplink case. After that, data will flow as shown in the Figure 11.

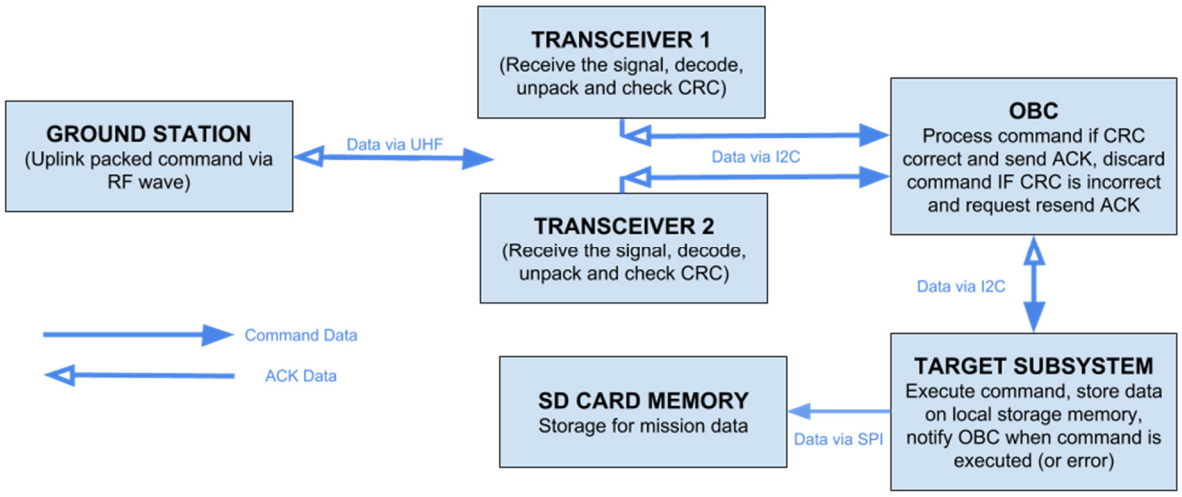


Figure 10 - Ten-Koh uplink data flow.

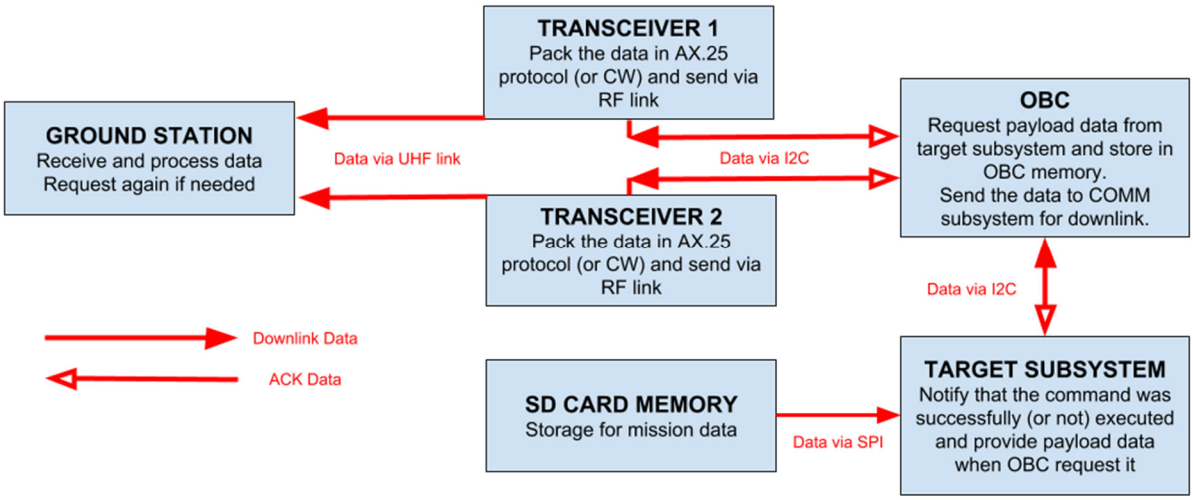


Figure 11 - Ten-Koh downlink data flow.

As is shown in the Figure 10 and Figure 11, every payload subsystem has a MicroSD card in which the measured data performed by any mission are stored in order to be sent when a ground station data download request command is

received by the OBC. All SD card parameters (last writing address, the read pointer address and counters) are stored into the local microprocessor EEPROM memory in order to be used lately even if the power supply is turned off by the EPS. The only two subsystems into the satellite, which do not include SD card are the OBC and the COMM subsystems.

Additionally, the OBC and the ECU subsystems store in the microcontroller EEPROM critical mission parameters. In the case of OBC, the satellite log which stores any event occurred into the satellite and the backup of the Real Time Clock (RTC) time. In the case of the ECU, the CPD threshold parameters are stored and can be updated via ground station command.

2.1.3. OBC EEPROM data management

One of the purposes of this research is to report the failures presented in four months of satellite operation due to the radiation environment. One of the presented failures was the OBC microcontroller EEPROM data corruption, but before to discuss the results in the next section is pertinent to revise the OBC EEPROM memory data usage. Into the OBC, the entire EEPROM memory (256 bytes in total) is used to store three types of data, the OBC reset counter, the minute byte of Real Time Clock (RTC) and the satellite log. The memory assignation for each part is shown in the Table 7.

Table 7 - OBC EEPROM memory usage.

Data Type	EEPROM Memory Address (Decimal)
Reset Counter	0 to 1
RTC	2 to 95
Satellite Log	96 to 255

The reset counter increases every time the OBC experiment a reset and consist in two bytes which allows to store up to 65,535 reset events. In the case of the RTC, the minute, hour and day bytes are stored every minute in the

EEPROM memory region in order to have a partial backup of the satellite time. Finally, in the satellite log region, the OBC stores every event that occurs into the satellite with the corresponding time taken from the RTC; it consists in four bytes in total, three for the RTC time (day, hour and minutes) and one for the command ID, wherewith, a total of 40 events can be stored. The reason for using a region into the EEPROM memory for the RTC time backup is to avoid to overwrite several times the same memory address constantly and produces a damage into the entire memory due to the excess of writing cycles given by the microcontroller datasheet [21].

2.1.4. OBC resets management

Finally, is important to show how the resets are managed into the satellite. The only subsystem allowed to reset other subsystems is the EPS. The resets can be of two types, soft resets in which the EPS sends a digital reset signal directly to the Master Clear Pin External Reset (MCLR) to the respective PIC microcontroller and hard resets in which the EPS turns off for a short time the respective power line in order to reset all the entire components into the subsystem. The EPS resets the OBC in two specific cases, the first is when the OBC does not send any I2C data request to the EPS during more than 15 minutes and the second is when the COMM subsystem receives a satellite reset command from the ground station; in this case, the COMM subsystem sends a direct digital signal to the EPS without any OBC intervention via GPIO and then, the EPS resets all the subsystems into the satellite. Based on the above, in the satellite log, it is possible to differentiate two types of OBC resets, a satellite reset which came from the ground station and unknown resets generated due to an OBC hang.

2.2. Ten-Koh On-orbit issues

2.2.1. OBC resets events

The operation of satellite was divided into two stages, the first one called Launch and early phase (LEOP) stage in which the operations were limited to test the correct functionality of every subsystem into the satellite and whose duration was one month, the second called mission stage in which the satellite started to perform payload missions in normal mode. Since LEOP operations, the OBC has started to show in the satellite log several resets due to OBC hangs or unexpected resets and that resets continued appearing during the mission stage. Those resets can be divided into two types, normal resets and unknown resets. As was explained in the previous section, the satellite log stores the events occurred into the satellite including the time in which those occurs taken from the RTC. In the case when the time of the reset event is correct and corresponds or is similar to the real time, then that event is a normal reset and the approximate position of the satellite in the orbit can be estimated. Otherwise, in the case when the time of the reset event is incorrect, then, that event is designated as unknown reset due to the value was corrupted in the EEPROM memory probably because a radiation effect and therefore, is not possible to know the exact point on the orbit where the event occurred. In view of above, the number of resets events occurred on the satellite since the launching day is shown in the Figure 12. The summary of the data obtained by the Figure 12 is shown in the Table 8.

Table 8 - OBC reset events summary.

Name	Value
Days of operation	123
Total reset events	132
Number of normal resets	102
Number of unknown resets	30
Average reset events by day	1.073

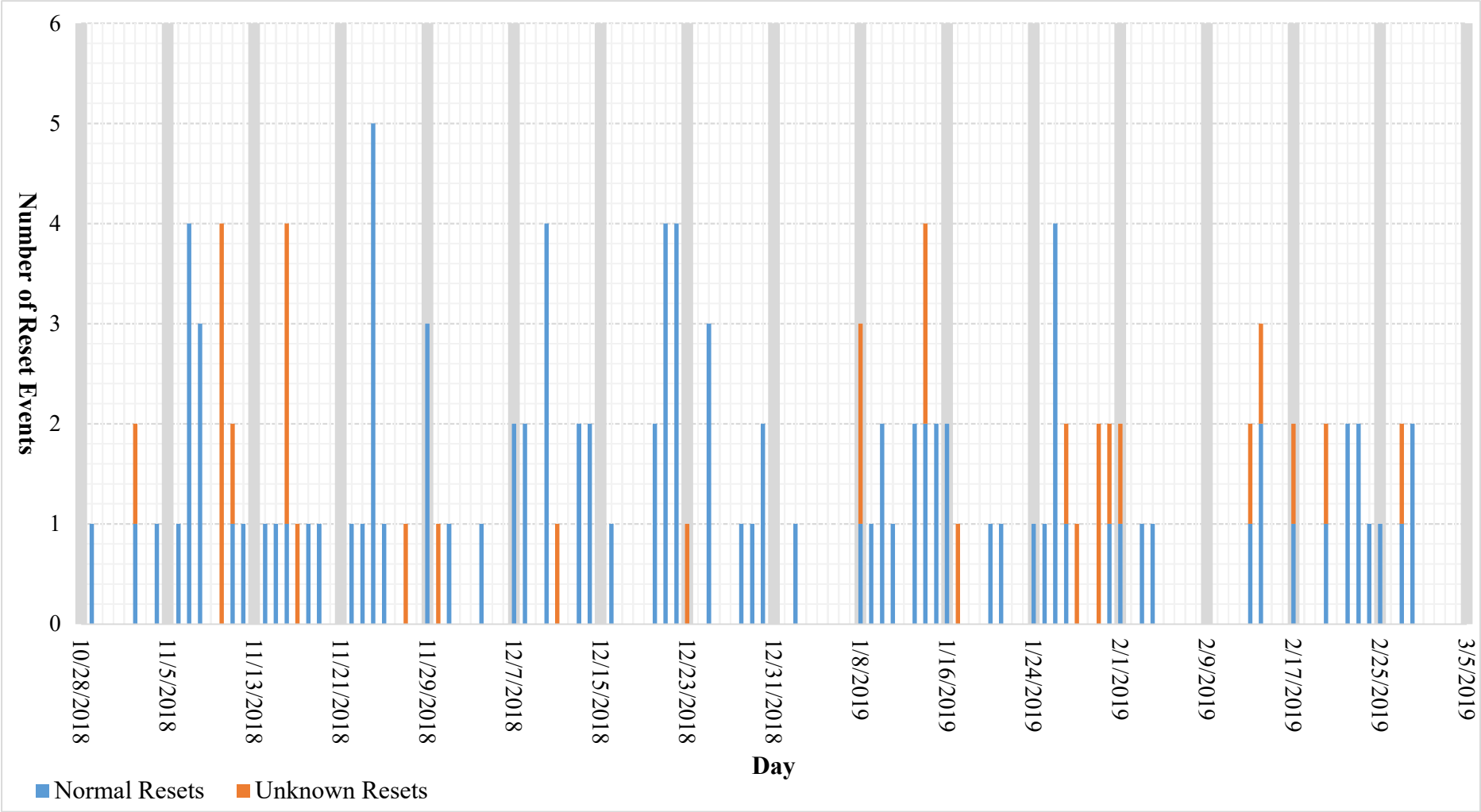


Figure 12. Number of OBC reset events during 123 days of Ten-Koh operations.

It is possible to estimate the location of the satellite when the normal reset events occurred using the time that appears in the satellite log and using a software for orbit calculation (e.g. Orbitron or STK). In the Figure 13, every red point represents the locations of the satellite where the normal resets occurred. In the case of the unknown resets, is difficult to estimate the position of the satellite where those resets occurred because the time stored in the satellite log is incorrect, however, comparing the satellite log with the ground station log, it is possible to estimate the correct value of the time when some resets occurred if the next event is a command sent from the ground station and the time between the two events is not big. Following that methodology, it was possible to recover the time of five unknown resets and therefore, it was possible to estimate their respective location in the orbit. In the Figure 14, the red points represent the position of the satellite when those unknown events occurred.

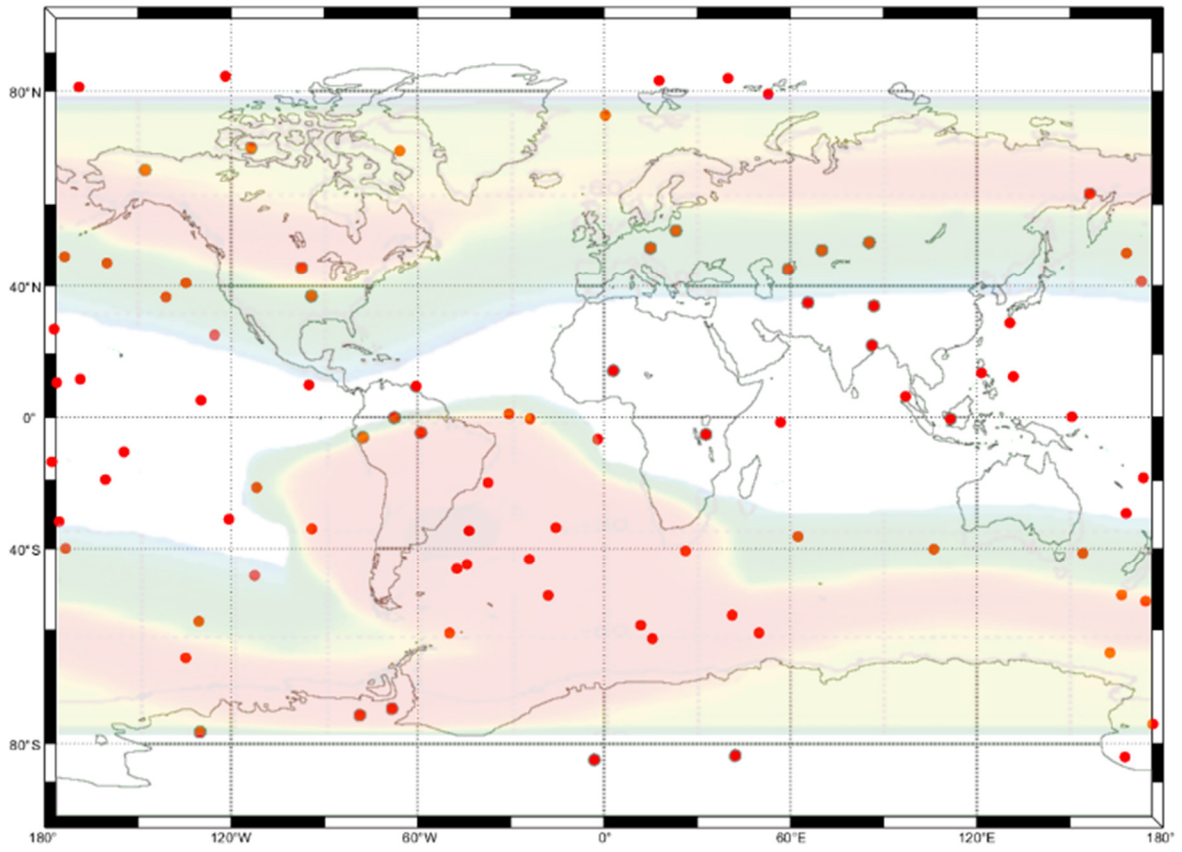


Figure 13 - OBC normal reset events occurrences. The red and green regions illustrate the count rate of electrons and protons greater than 0.5 MeV [22].

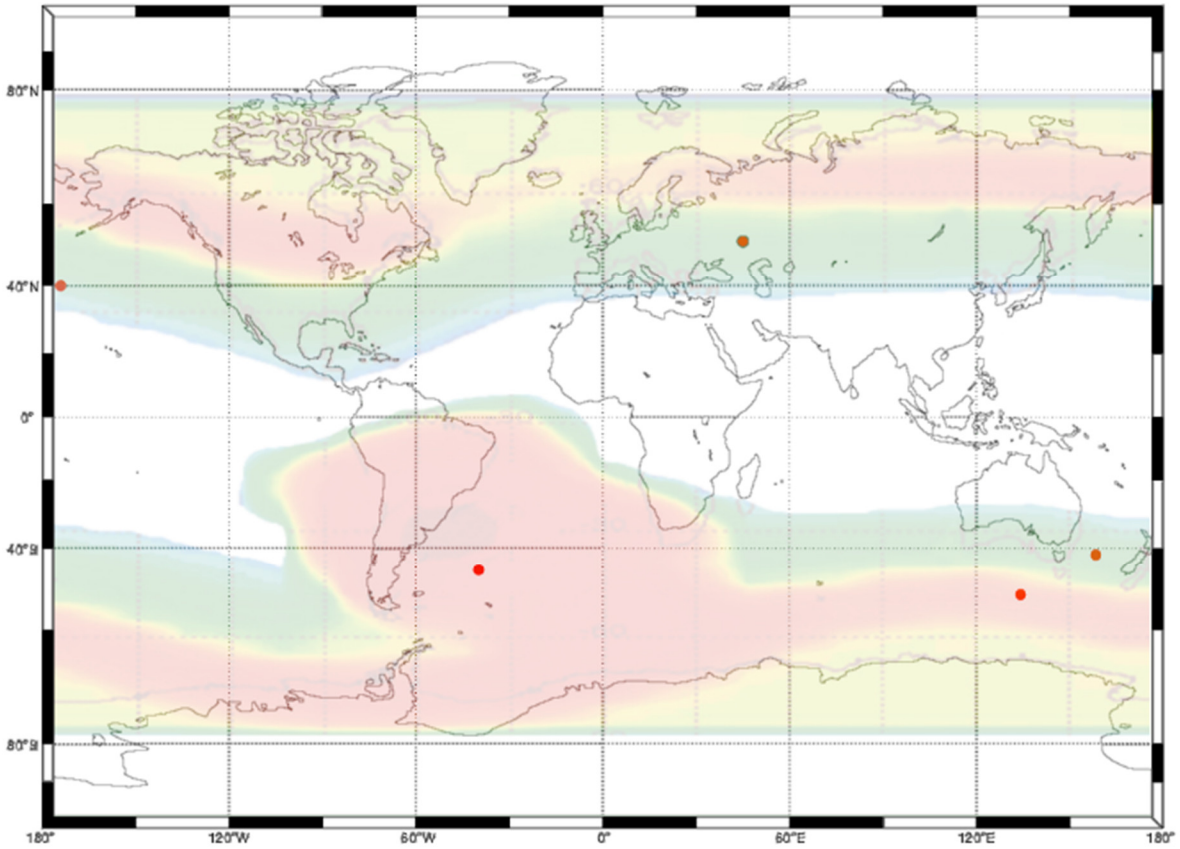


Figure 14 - OBC unknown reset events occurrences. The red and green regions illustrate the count rate of electrons and protons greater than 0.5 MeV [22].

In order to analyze if the reset events were presented in the zones of the orbit where the amount of radiation is high, the resets occurrence location points are shown over a count rate of proton and electrons greater than 0.5 MeV in LEO plot measured by the NASA/SAMPLEX satellite [22]. The summarize of the results for the normal and unknown reset events are shown in the Table 9.

In the case of normal resets, the 62.06% of those events occurred in the region with presence of proton and electron flux greater than 0.5 MeV while the 37.94% of those events occurred outside the radiation region. Regarding the unknown events, only five of those events could be recovered from the satellite log of which all of them occurred in the region with radiation fluxes greater than 0.5 MeV.

Table 9 - Reset events results summary.

Name	Value
Total number of normal resets plotted	87
Normal resets into the region > 0.5 MeV	54
Normal resets outside the region > 0.5% MeV	33
Total number of unknown resets plotted	5
Unknown resets into the region > 0.5 MeV	5
Unknown resets outside the region > 0.5% MeV	0

2.2.2. EEPROM failures into other subsystems

As mentioned previously, every subsystem except the OBC and the COMM use an SD card memory to store the measured data and all those subsystems store the SD card parameters (addresses and counters) into the EEPROM memory of each PIC microcontroller in order to be able to use it after a power off performed by the EPS. Same as the OBC, several subsystems also experimented data corruption into the EEPROM memories, probably for the same reasons as OBC.

Table 10 - Ten-Koh subsystems EEPROM failures summary.

Subsystem	Always turned on?	Failures?
OBC	Yes	Yes, EEPROM (satellite log, reset counter and RTC backup)
COMM	Yes	No, EEPROM not used
ADS	Yes	Yes, SD card does not respond
EPS	Yes	Yes, EEPROM (SD card address pointer)
ECU	No	No, SD card parameters and payload thresholds stored in EEPROM works well
UECU	No	No, SD card has not been used
MMECU	No	No, SD card has not been used

In the Table 10, a summary of all satellite subsystem failures is shown, also the table shows if the subsystem is always turned on in order to analyze if that can be another reason that facilitate the EEPROM failures into the PIC microprocessor.

It is possible to see based on the data shown in the Table 10 that only the subsystems that are permanently turned on (bus subsystems) are those that had been presented failures, especially with the EEPROM memory, the payload subsystems are turned on only in the mission mode that generally do not take more than one orbit, then those are not exposed to the radiation environment in operation for a long time. Regarding the EPS EEPROM failure, it also consists in a data corruption, similar to the OBC case. In this scenario, it is possible to know that the value was corrupted because the EPS stores the housekeeping values every minute and it stores the corresponding value in the successive SD card address, however, after OBC unknown resets also the EPS started to save the housekeeping data in a different memory address and in some cases even the previous values were overwritten due to the address pointer data corruption. The ECU uses the same methodology for saving the mission data into the SD card but because it is not turned on for a long time then the subsystem has not experimented failures as yet.

2.3. Reset events analysis

In the previous section, Ten-Koh on-orbit issues were presented and discussed. The results show that the EEPROM memory into the PIC16F877 microprocessor experiments failures when is working continuously and several writing and reading transactions are performed. However, it is not possible to conclude that the failures are generated only due to the radiation environment because it was not possible to know the position of the satellite in all resets events where the data into the EEPROM was corrupted. It was possible to estimate just 5 of 30

of those events and the location matches with the zones in which the proton and electron flux is greater than 0.5 MeV (matches with the south and north radiation belts and the South Atlantic anomaly), also the 62.06% of the normal resets occurred inside that region which it is possible to formulate the hypothesis that probably these failures are mostly presented due to radiation phenomena. However, previous missions like Shinnen-2 used the same processor and no malfunctions was reported, even some radiation test was performed and the results shown that the PIC16F877 is suitable for space applications as mentioned in [18]. Also, as mentioned previously, the Ten-Koh CPD includes a Liulin spectrometer detector which can measure the total absorbed dose rate and the flux surrounding energetic particles in the space. This instrument has been used in several missions (included the International Space Station and deep space missions). It includes two PIC microprocessors (PIC16C74) and they have not reported any malfunction. The Liulin instrument was used in the Indian Chandrayaan-1 satellite performing experiments crossing the SAA directly at about 3000 km altitude, the maximum dose per channel reaches $1800\ \mu Gy/hour$ ($\sim 100000\ \mu Gy/hour$ in total) [23]. Is important to mention that the PIC microcontroller used in the Liulin instrument belongs to the same family used in the Ten-Koh mission, but is not the same reference, also is not mentioned if the EEPROM memory into the microcontrollers has been used and additionally, the Liulin instrument is mounted into an aluminum cover plate plus some additional shielding which provide more protection against radiation issues [24].

In order to get more relevant information and to conclude the causes of the malfunctions discussed in this chapter, a radiation test was performed to the PIC microcontroller and the results will be presented in the chapter 6.

CHAPTER 3 - TEN-KOH COMMUNICATION SYSTEM ARCHITECTURE

3.1. Hardware architecture

The communication subsystem consists of two lines of communications, both were designated to work in the Ultra-High-Frequency (UHF) band. The downlink line includes a transmitter at 437.3 MHz and the uplink line includes a receiver at 435.2 MHz for downlink. The main purpose of the communication system is to receive uplink commands from the ground station and downlink data from satellite to ground station using the specified frequencies. The block diagram is illustrated in the Figure 15. The system consists in a Communication Control Unit (CCU) which is an 8-bit Microchip PIC16F877 microcontroller, it is in charge of managing the data received and to be sent to the On-Board Computer (OBC) using the Inter-Integrated Circuit (I2C) bus. For the transmitter part, the CCU is in charge to receive the data coming from the OBC, in order to do that, it has a software module which encodes the data using the AX.25 G3RUH protocol and send the encoded data to the Nishimusen TXE430MFMCW-302A module. Also, the CCU needs to configure the operation mode of the transmitter module using a digital interface. A level converter between the CCU and the transmitter module is needed in order to convert the voltage levels from 5V to 3.3V. For the receiver part, the CCU is in charge of processing the command data received from the ground station through the Nishimusen RXE430M-301A by generating a General-Purpose-Input-Output (GPIO) signal. It interrupts the OBC and send the data using the I2C bus. In addition, the CCU has an AX.25 Bell 202 decoder implemented by software and also, it has the duty to configure the receiver module frequency using a special digital interface similar to Serial Peripheral Interface (SPI). Two additional components are present between the

receiver module and the CCU, the first is a low pass filter at the input of the AFSK modem in order to reduce the noise in the audio output generated by the Nishimusen module and the second is the AFSK modem itself (MX-614) which is in charge to demodulate the AFSK analogue signal into a digital one. The final Printed Circuit Board (PCB) implementation is shown in the Figure 16.

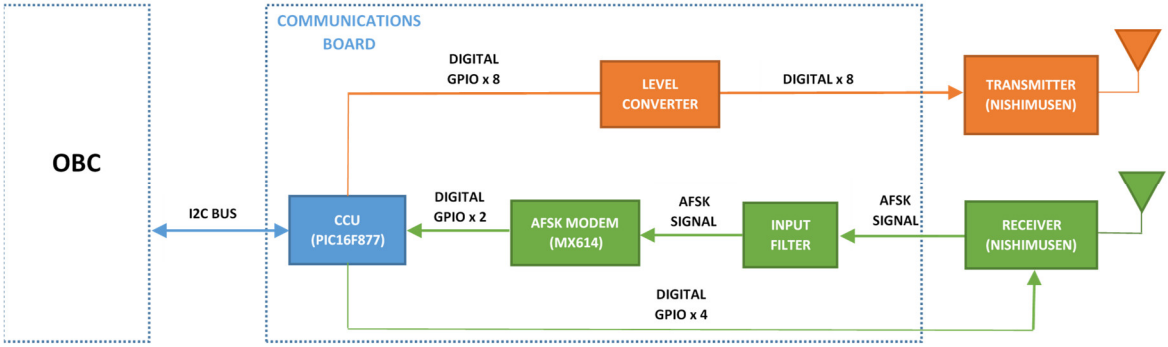


Figure 15 - Ten-Koh communication system block diagram.

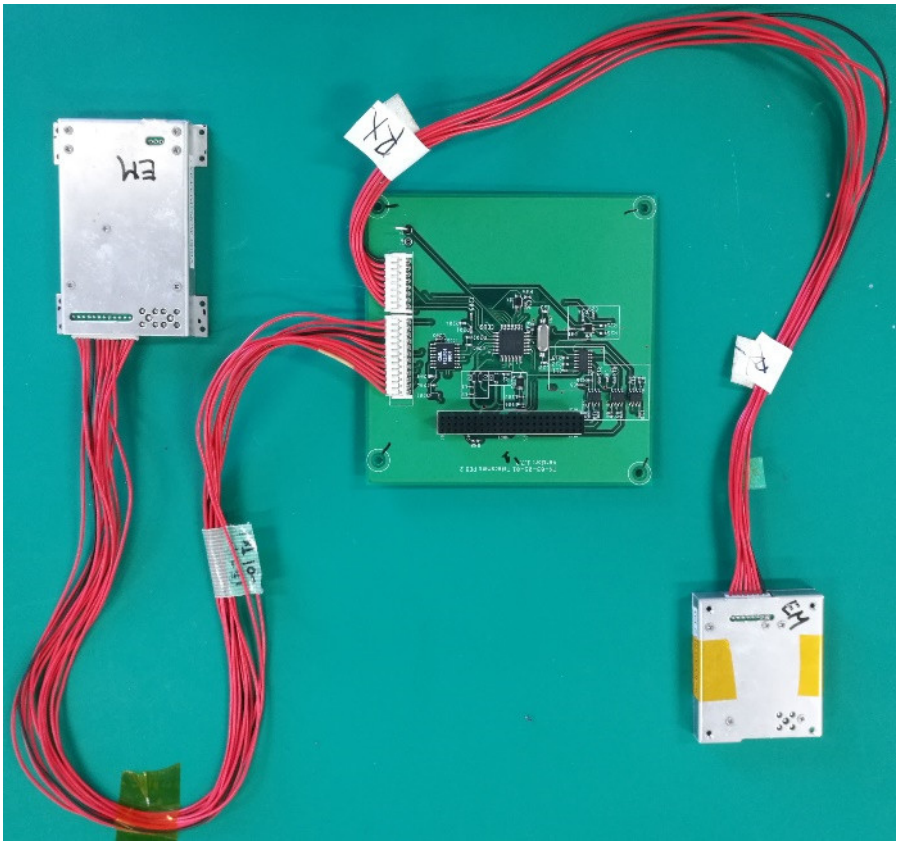


Figure 16 - Final Ten-Koh communication system PCB with RF modules.

Ten-Koh communication subsystem utilizes two UHF amateur frequency bands in full duplex mode. The data transmission is a packet based using the AX.25 protocol with baud rates of 1,200bps using a Frequency Shift Keying (FSK) modulation for uplink and 9,600bps using a Gaussian Minimum Shift Keying (GMSK) modulation for downlink. Also, as a system requirement, it includes a hot redundancy reception module and a cold redundancy transmitter module. The ground station is located in the Kyushu Institute of Technology (Tobata Campus) and includes an ICOM9100 transceiver, a Kantronics KPC9100+ Terminal Node Controller (TNC), a Yaesu G-5500 rotator, a Low Noise Amplifier and a dual Yagi antenna array.

3.2. Software architecture

The software architecture of the communication system is shown in the Figure 17. It consists of two hardware driver modules, the first one performs the I2C slave communication with the OBC for receiving the control commands e.g. CW mode, transmission mode and receiving mode. The second hardware driver is for controls the GPIO interfaces necessary to interrupt the OBC when a valid command is received from the ground station and to control and configure the Nishimusen transmitter and receiver modules. The receiver software module consists of two blocks, the Bell 202 AX.25 1,200bps modem and the configuration blocks, the first is in charge to decode the packets received by the Nishimusen receiver module and store those locally in the microcontroller buffer in order to be read by the OBC lately, the second is in charge to generate the digital interface needed to tune the Nishimusen receiver to the correct frequency values at boot up.

The transmitter software module also includes two blocks, the configuration block in charge to generate the GPIO signals needed to change the Nishimusen transmitter mode (CW or 9,600bps data) regarding the corresponding state machine state and the G3RUH AX.25 9,600bps modem block in charge to encode

the data received from the OBC to be sent to the ground station. Finally, the software is controlled entirely by a state machine which is constantly waiting for interruptions coming from the OBC thru the I2C bus.

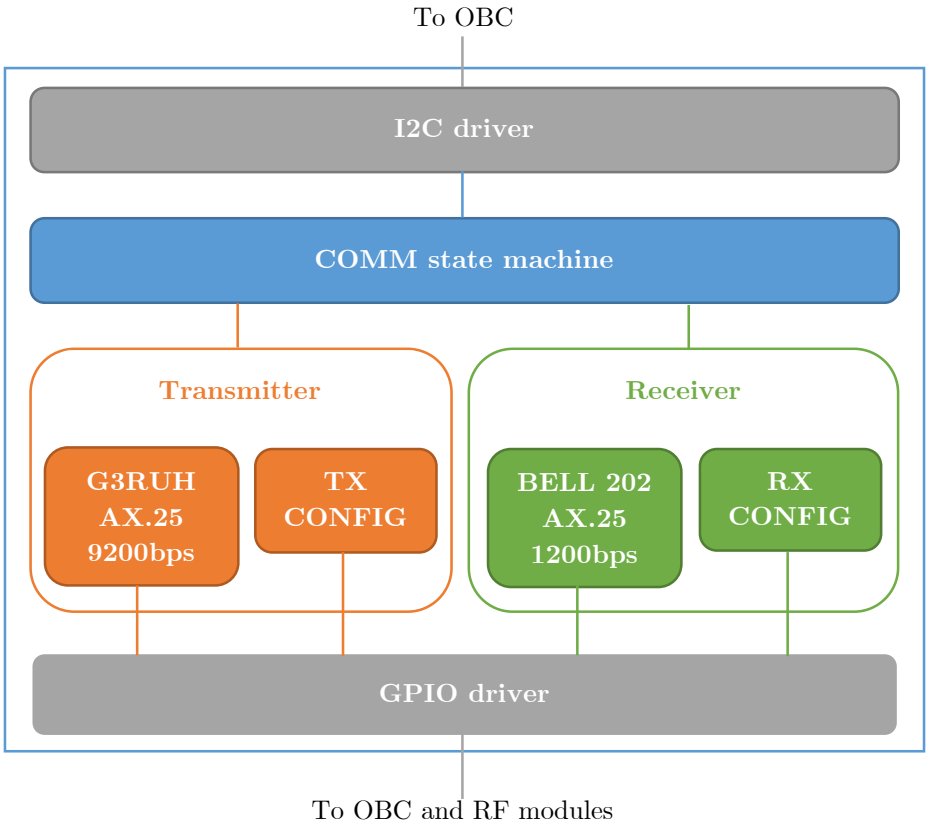


Figure 17 - Ten-Koh software architecture.

Regarding the development tools for the software development, the Microchip MPLAB IDE and the XC8 compiler were used and the programming language for writing the code was C.

3.3. System constraints and limitations

After four months of on-orbit operations and during the system development phase, some constraints were found that made the design and the satellite operations complex. The first and the most important was the microcontroller, the Microchip PIC16F877 which was used in several successful missions. The major limitation using this microcontroller was that it has a 368 Bytes of SRAM and those are divided into four parts of 96 bytes in total [21], then, the system

has two major buffers, one for reception (35 bytes) and the other for transmission (65 bytes). For that reason, the system was limited to sending packets of just 65 bytes when the AX.25 protocol allows to send packets until 255 bytes of data. Of course, it decreased the capabilities of the entire system and it took more time and processing resources when a large amount of data is needed to be sent, especially the data coming from the payloads. In the Table 11, the amount of data generated by the Ten-Koh payloads and the number of AX.25 packets needed to download a single measurement experiment using the 65 bytes packet size in comparison with the full AX.25 available packet size is shown. It is possible to observe that due to the microcontroller memory limitation in the system, the number of packets needed to download a single payload experiment is four times higher than using the full AX.25 packet size. According to the above, this constraint affects directly the number of passes required to download a single payload experiment.

Table 11. Data generation for the Ten-Koh payloads.

Payload	**Amount of data generated per single measurement by channel [Bytes/ch]**	**Maximum time duration of the experiment [minutes]**	**Total data generated for one experiment by 2 channels [Bytes]**	**Number of packets needed to download 1 experiment (Ten-Koh system – 65 bytes)**	**Number of packets needed to download 1 experiment (AX.25 – 255 bytes)**
DLP (Hi-Resolution)	3,000	15	900,000	13,847	3,530
DLP (Mid-Resolution)	1,500	15	672,000	10,339	2,636
DLP (Low-Resolution)	600	15	180,000	2,770	706
CPD	2,528	15	75,840	1,668	298
ADS	64	15	57,600	887	226
Material Mission	60	15	18,000	277	71

The Ten-Koh orbit is Sun Synchronous at 613 km altitude and 97.8 degrees of inclination, then, in one day, the satellite passes over the ground station in Japan

four times, however, typically only two of those passes have a good elevation for data downlink. In the real practice, in a good elevation pass, it was possible to download around 300 valid packets on average. Therefore, for example, to download a single DLP in high-resolution measurement with the mentioned limitation, the system requires 46 passes (23 days) which is a considerable operation time. On the other hand, using the full AX.25 packets, the operation time needed is 12 passes (6 days).

Other constraints regarding the microcontroller are the limitation to use only one level of priority in the interruptions, the low clock speed (20 MHz maximum) [21] and the size of the SRAM memory (for the final software implementation it occupied the 97% of the entire available memory).

The second major constraint is regarding the Nishimusen transmitter and receiver modules. These modules have been used successfully in several Japanese satellite missions e.g. Shinen-2 and Horyu-IV, but those devices are only limited to operate in VHF and UHF bands using AFSK (AX.25 Bell 202 at 1200 bps) and GMSK (AX.25 G3RUH at 9600 bps) modulations respectively which is not quite enough to meet the amount of data generated by the main payloads.

Other constraints regarding those modules are that is not possible to modify the modulation schemes by software and in the case of the transmitter, even is not possible to modify the transmission frequency by software. The only way to do it is reprogramming the corresponding values in a specific EEPROM memory allocation using a PICKit programmer [25].

CHAPTER 4 - PROPOSED SDR IMPLEMENTATION

4.1. Methodology and Raspberry Pi prerequisites

In order to be able to use the Raspberry Pi as the main processor for the SDR proposed architecture, some software tools are necessary. The first requirement is to prepare the operating system into the MicroSD card in order to be loaded at boot up. The operative system used for the proposed designs, simulation and testing is the Raspbian Stretch Linux distribution, it is based on Debian and is the official distribution provided by the Raspberry Pi foundation, it includes the required libraries that allows to use a graphical desktop environment (required for the GNU Radio Companion environment).

The second and the principal tool is the GNU Radio suite, which is a free and open source software toolkit that provides signal processing blocks in order to implement and simulate software defined radios. It includes a graphical environment called GNU Radio Companion in which it is possible to create, configure and connect the corresponding blocks required to build the design. Moreover, the graphical environment allows to execute and simulate the design in real time using the adequate blocks (QT GUI blocks), however, that blocks must be removed completely at the moment of the final implementation. Before installing the GNU Radio package, it is recommendable to check whether the system has the required dependencies and libraries to build various signal processing blocks [26], after that, the installation process can be completed by executing the command "*sudo apt-get install gnuradio gnuradio-dev*" on a terminal window.

By default, GNU Radio offers several signal processing blocks and modules for SDR development. However, the installation of additional open-source libraries is needed in order to implement the proposed architectures presented in this paper. That libraries are: The SatNOGS GNU Radio Out-Of-Tree module [27], the gr-

satellites library which includes several decoders for several radio amateur satellites [28] and the gr-bruninga library which includes some tools to encode and decode Audio Frequency-Shift Keying (AFSK) signals [29]. The libraries mentioned above were installed into the Raspberry Pi following the instructions that appear on their respective web sites.

Another advantage to using the Raspberry Pi in conjunction with Raspbian is that there are available native drivers for managing the I2C, SPI, UART and GPIO interfaces. It facilitates the development because if any of that interfaces are needed in the design, from the software standpoint, just importing and calling the corresponding drivers is enough to use them.

The methodology used to develop all the components into the SDR architecture is shown in the Figure 18. The creation, configuration and connection of the corresponding signal processing blocks are made in the GNU Radio Companion graphical interface in order to perform the corresponding testing, tuning and simulations to verify the design performance.

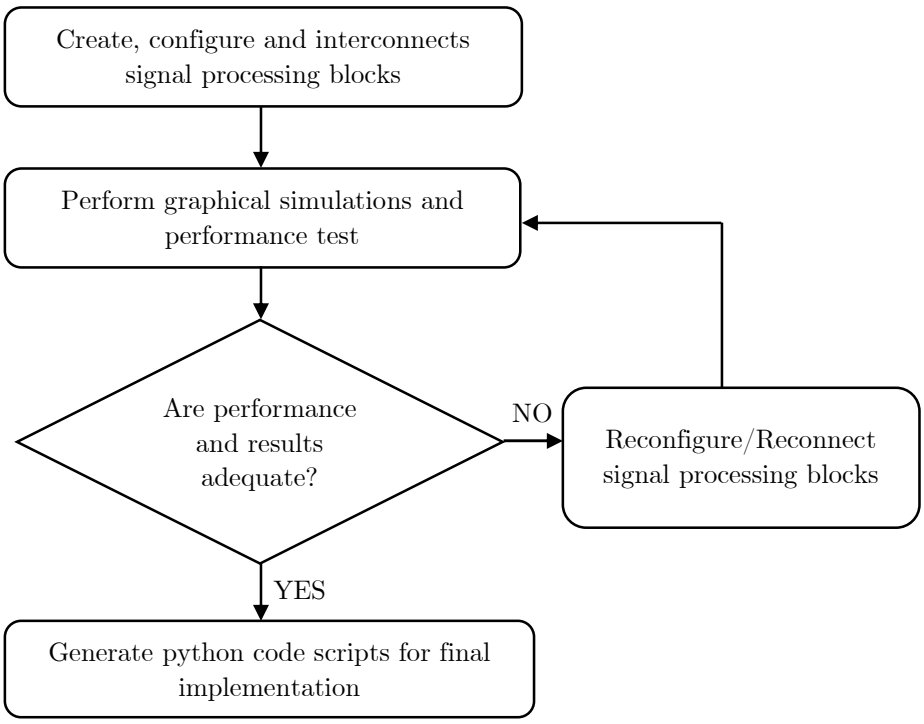


Figure 18 - SDR design methodology flow chart using GNU Radio.

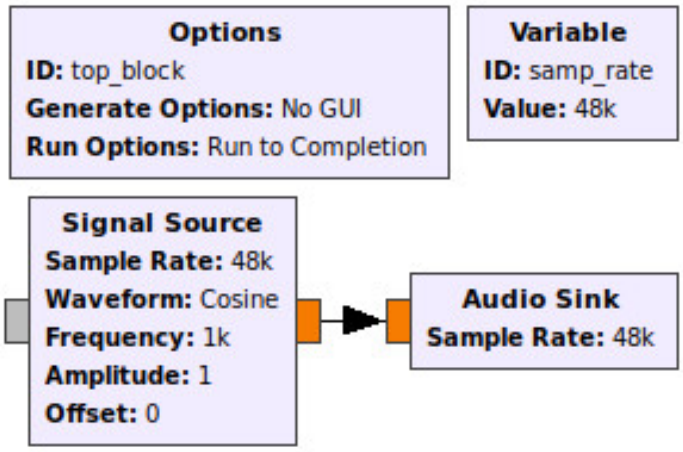


Figure 19 - GNU Radio Companion basic exemplar diagram.

Finally, the functional scheme is exported in a Python code script that can be used by the main program (state machine) to exchange input/output data and configuration parameters. To explain the above in a better way, the most basic exemplar diagram in the GNU Radio Companion is shown in the Figure 19, which is a Cosine Signal Generator. Here the Signal Source block is configured to output a 1 kHz cosine signal with amplitude equals to 1, without offset and the output signal is connected to the Audio Sink block which controls the audio output into the Raspberry Pi to generate the analog signal. In the Figure 20, a part of the generated python code is shown in which it is possible to observe how the variables are declared, how the blocks are defined and how the configuration values are included as function inputs into the blocks. For example, in the case of the Signal Generator block, it is declared calling the function "*analog.sig_source_f*" in which the input parameters are the sample rate, the waveform, the frequency in Hz, the amplitude and the offset. Those parameters can be reconfigured externally using another script or high-level application. Finally, the physical connection declaration between the two blocks is shown.

In the next section, the proposed SDR implementations will be shown. The GNU Radio companion diagrams that will appear corresponds to those that were used for testing the parameters and performance of the system.

```
class top_block(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Top Block")

        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 48000

        ##################################################
        # Blocks
        ##################################################
        self.audio_sink_0 = audio.sink(samp_rate, '', True)
        self.analog_sig_source_x_0 = analog.sig_source_f(samp_rate, analog.GR_COS_WAVE, 1000, 1, 0)

        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_sig_source_x_0, 0), (self.audio_sink_0, 0))
```

Figure 20 - Generated phyton code script from basic exemplar diagram.

4.2. Partial SDR implementation

The partial SDR proposed implementation consists in replacing the Ten-Koh communication PCB by only a Raspberry Pi module, a USB external audio card and keeping the same Nishimusen RF modules. In this case, it is not possible to implement an entire SDR architecture property because the constraints previously discussed and still present in the transmitter and receiver modules. However, the major constraints due to the microcontroller can be solved and the hardware implementation is significantly improved because basically the PIC microcontroller is upgraded by the Broadcom BCM2837B0, with Cortex-A53 (ARMv8) 64-bit SoC included into the Raspberry Pi.

The block diagram of the hardware architecture is shown and explained in the Figure 21. To be able to receive the AFSK audio signal generated by the receiver RF module, an available audio input is needed, but unfortunately the Raspberry Pi has only an available audio output. To solve it, a simple USB sound card is connected and configured as the main audio card into the Raspbian operative system. Another advantage that this implementation brings is the option to add easily and without additional hardware an AX.25 Bell 202 1,200bps encoder using the same audio interface and just connecting it to the audio input in the Nishimusen transmitter module as is shown in the Figure 21.

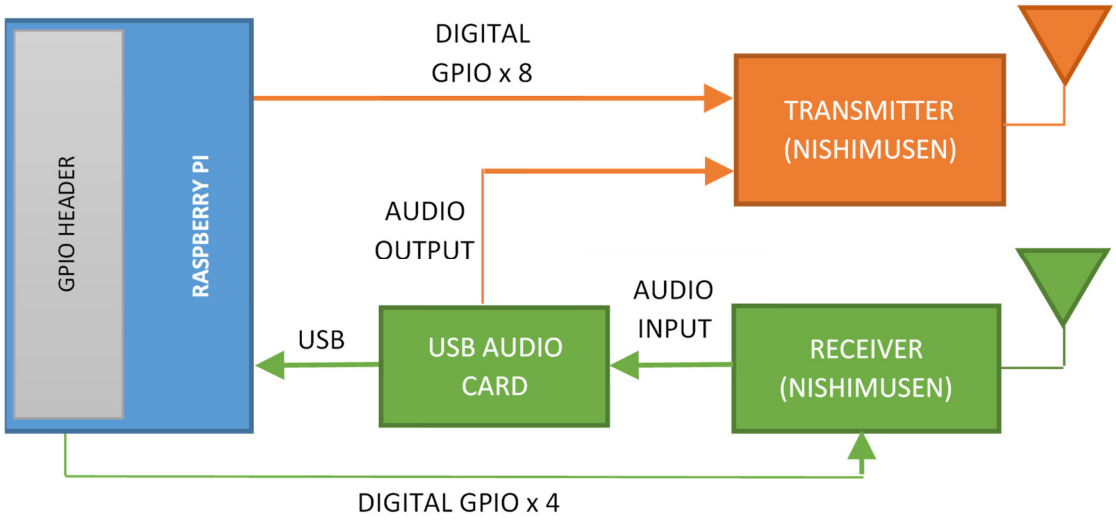


Figure 21 - Partial SDR implementation architecture.

In the case of the software, the architecture is similar as the Ten-Koh case, there is one part in charge to handle with the transmitter at 9,600bps and the other in charge to handle the receiver at 1,200bps. For the receiver, the GNU Radio platform is used to implement the AFSK decoder replacing the passive input filter and the FX614 AFSK modem into the Ten-Koh system. As shown in the Figure 22, the Audio Source block is used to receive the audio signal provided by the receiver RF module, after, the signal is connected to the FSK Demodulator block from the gr-bruninga library which decodes the AFSK signal into digital data (the same function that the FX614 does in the Ten-Koh system). Finally, the signal is connected to the input of the HDLC Deframer block from the gr-satellites library which is in charge to decode the AX.25 into raw data. To verify if the correct data is received, the HDLC to AX.25 block is used to print out in the terminal the decoded data. The GNU Radio Companion diagram of the receiver AFSK decoder is shown in the Figure 22.

In the case of the transmitter, the 9,600bps encoder and the control interfaces to configure the RF module cannot be implemented by using the GNU Radio platform because everything is controlled digitally by GPIO interfaces, a function

which the platform does not support. The same situation happens with the digital interface for configuring the receiver RF module. In this case, the proposed solution is modifying the Python code generated previously by GNU Radio, adding the GPIO driver support to handle the digital interfaces for controlling the Nishimusen modules and for implementing the AX.25 G3RUH encoder at similar way as implemented in the PIC microcontroller.

Additionally, as mentioned previously, due to the inclusion of the USB audio card that can provide an audio output to the transmitter RF module, it is possible to implement an AFSK Encoder using the GNU Radio platform. The diagram is shown in the Figure 23, on it is possible to observe three blocks that take the data coming from a text or hex file and convert it into strings, that strings are input to the AX.25 encoder and FSK modulator that consist in two blocks, the String to APRS block in charge to add the AX.25 headers (call sign source and destination) and the AX.25 AFSK Modulator block in charge to include the flags, preamble and postamble required to the AX.25 encoding, same as the value of the frequency of the two tones required for the FSK modulator. The two mentioned blocks are part of the gr-bruninga library. Finally, the encoded AFSK data is connected to the input to the Audio Sink block which will generate the analog signal in the USB audio card output in order to be sent to the Nishimusen transmitter input.

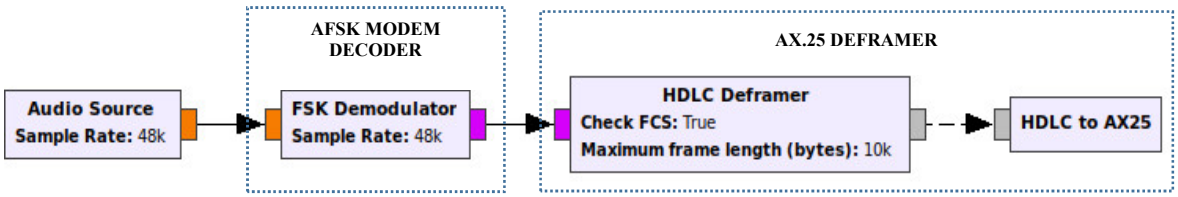


Figure 22 - AFSK decoder block diagram implemented on GNU Radio.

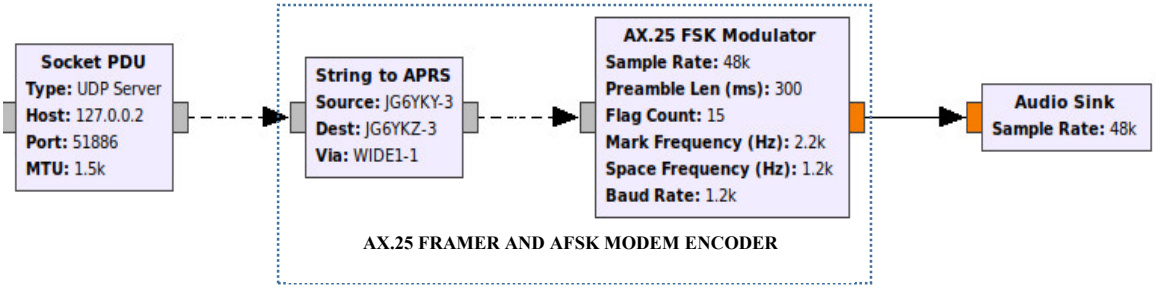


Figure 23 - AFSK encoder block diagram implemented on GNU Radio.

4.3. Complete SDR implementation

The complete SDR implementation consists in the previously shown improvement contributed by the replacement of the Ten-Koh CCU PIC by the Raspberry Pi module plus the replacement of the Nishimusen modules by a single LimeSDR-mini module. In this case, the complete SDR architecture is shown in the Figure 25 in which the two typical parts of an SDR architecture are present, the software part that is controlled entirely by the Raspbian operative system and the hardware part that consist in the Raspberry Pi module and the LimeSDR-mini RF module connected by one USB port. The hardware part is shown in the Figure 24.

The software part is divided into four modules, two of them consists of a transmitter and a receiver in AFSK Bell 202 1,200bps and the other two consist in a transmitter and a receiver in GMSK G3RUH 9,600, all of them were designed, simulated and tested using the GNU Radio Companion suite.

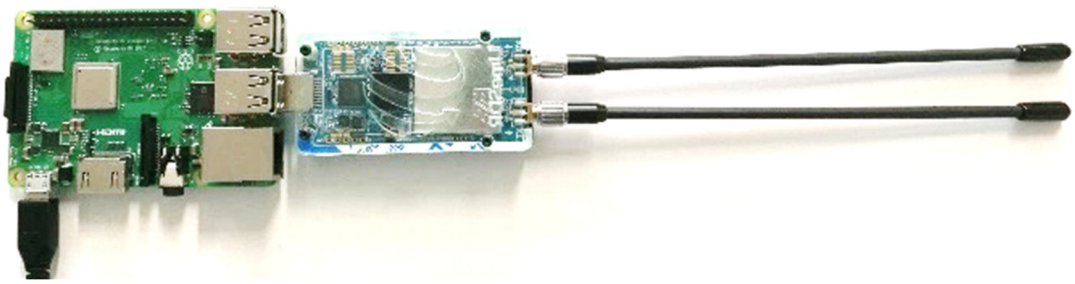


Figure 24 - Complete SDR hardware implementation.

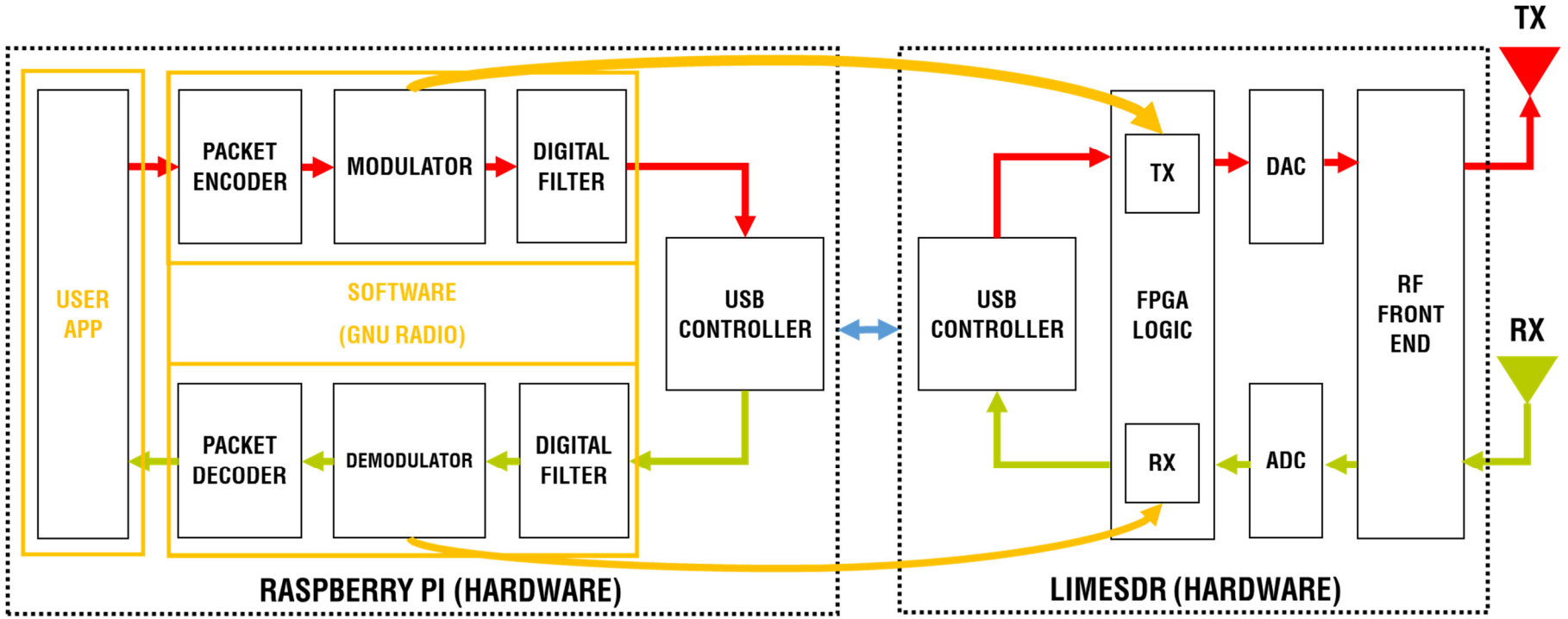


Figure 25. Complete SDR architecture block diagram.

The AFSK transmitter block diagram implemented in GNU Radio is shown in the Figure 26. It consists if a Socket PDU block which establishes a Protocol Data Unit (PDU) session in order to receive the data to be transmitted, an AFSK modulator that includes a block to generate the AX.25 headers (source and destination call signs) and the FSK modulator block itself in charge to generate the Bell 202 audio tone, same as the AFSK encoder in the partial SDR implementation. Also, it includes a Frequency Modulator (FM) block which is in charge to modulate the audio tone into the carrier frequency (437.3 MHz) and finally, the LimeSuite Sink transmitter block that configures all the parameters needed for the LimeSDR-mini module, e.g. frequency, internal filters, power output gain and frequency channel.

The AFSK receiver implemented in GNU radio is shown in the Figure 27. It consists in the LimeSuite Source receiver block that tune the RF frequency at 435.2 MHz, the internal filters, the channel band and the receiver Low Noise Amplifier (LNA) power level. It includes also a low pass filter block for removing the undesirable frequencies, an FM demodulator in order to recover the audio tone from the radio frequency carrier, an amplifier block to adequate the correct level value for the audio tone, an FSK demodulator which decodes the digital data from the audio tone and finally, a High-Level Data Link Control (HDLC) deframer that decode the AX.25 packets and outputs the row received data. Joining the AFSK transmitter and receiver, we have a complete SDR solution for the AX.25 BELL 202 at 1,200bps used in Ten-Koh and typically used in satellite communications using the radio amateur frequency band.

Regarding the GMSK G3RUH 9,600bps transmitter, the proposed solution implemented in the GNU Radio Companion is shown in the Figure 28. It includes a User Datagram Protocol (UDP) Message Source block which establishes a UDP session in order to receive the data to be transmitted, an AX.25 encoder (which is part of the SatNOGS library) for encode the data in AX.25 G3RUH which include the flags, headers and one additional scrambler required for 9,600bps baud

rate. Also, a GMSK modulator block is used to modulate the encoded data in a carrier wave. The resulting signal needs to be resampled depending the value of the samples per symbol used in the modulator block and the baud rate (10 samples per symbol and 9,600bps respectively). For that duty, the Rational Resampler block calculates the resampler rate adequate for the input of the LimeSuite transmitter block. The resampler rate is calculated as shown in the Equation 9. The $LimeSDR_{Sample\_rate}$ is the sample rate which is configured into the LimeSuite sink (TX) block, the baud rate is 9,600bps, the $samples_{symbol}$ are the number of samples to represent one GMSK constellation point (in this case is 10) and the interpolation and decimation values are the parameters needed by the resampler block in order to perform the sample rate escalation.

$$LimeSDR_{Sample\_rate} = \frac{(baudrate \times samples_{symbol}) \times Interpolation}{Decimation}$$

Equation 9 - LimeSDR resample rate calculation.

Finally, the resulting signal is connected to the input of the LimeSuite Sink (TX) block that configures the frequency value at 437 MHz, the sample rate at 500 kHz, the internal filters, the power output and the channel band. The power output has to be modified depending how far is the receiver in order to avoid the saturation.

For the GMSK G3RUH receiver, the GNU Radio Companion implementation is shown in the Figure 29. At first, the LimeSuite source (RX) block is used to configure the parameters of the receiver at the same way used in the AFSK implementation.

Then, the received signal has to be resampled properly to the 9,600bps baud rate using the Rational Resampler block, the interpolation and decimation values are chosen in adequate way using the Equation 10.

$$resample\_rate = \left(\frac{LimeSDR_{Sample\_rate} \cdot Interpolation}{Decimation} \right)$$

Equation 10 - Resample rate for GNU radio rational resampler block.

The above changes the sample rate from 500 kHz used by the LimeSuite block to 96 kHz needed in the Low Pass Filter block which is in charge to remove the undesired frequency component of the received signal. After the GMSK demodulator block is used to demodulate the signal and to divide the sample rate by the modulation samples per symbol to recover the original G3RUH baud rate (9,600 bps). Finally, to decode the data packets, the G3RUH AX.25 decoder is implemented using an NRZI decoder block, a descrambler block and one HDLC deframer block (part of the gr-satellites library). The decoded data packets are printed out and stored using the HDLC to AX.25 block which is part of the SatNOGS library which includes different utilities and tools in GNU Radio to develop ground station software for radio amateur satellites [27].

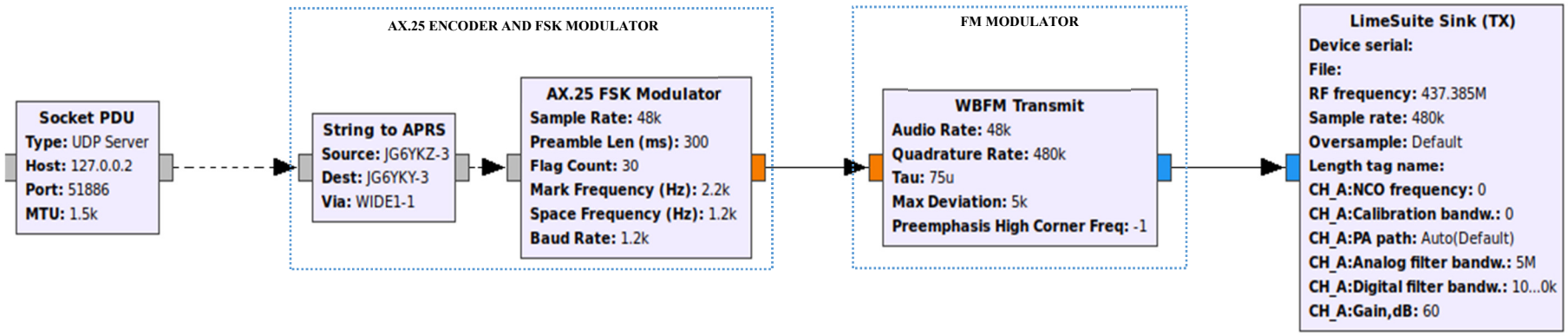


Figure 26 - AFSK BELL 202 transmitter implemented in GNU Radio.

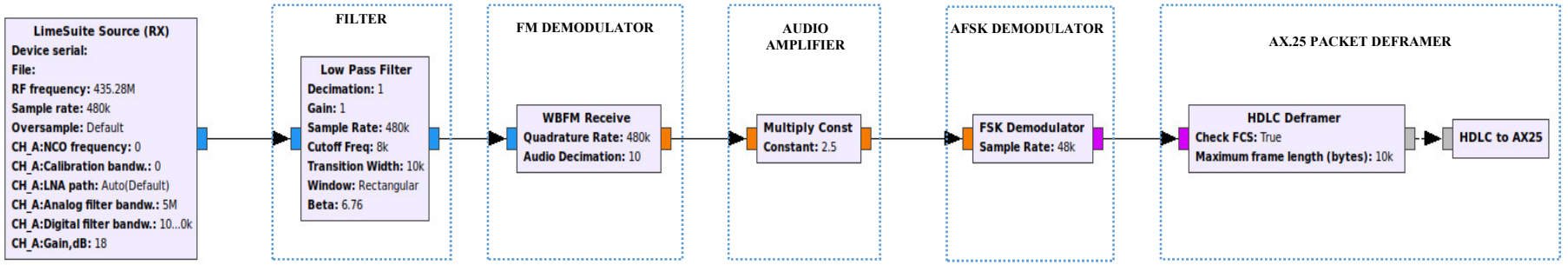


Figure 27 - AFSK BELL 202 receiver implemented in GNU Radio.

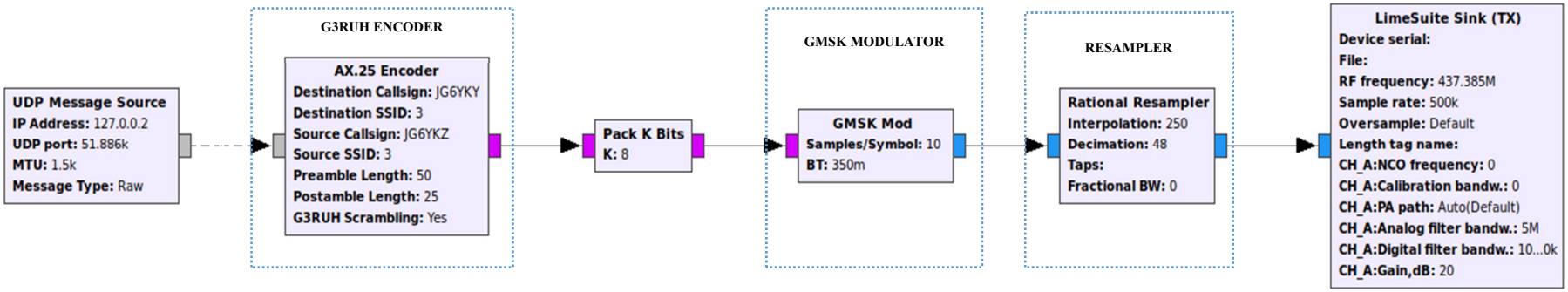


Figure 28 - GMSK G3RUH transmitter implemented in GNU Radio.

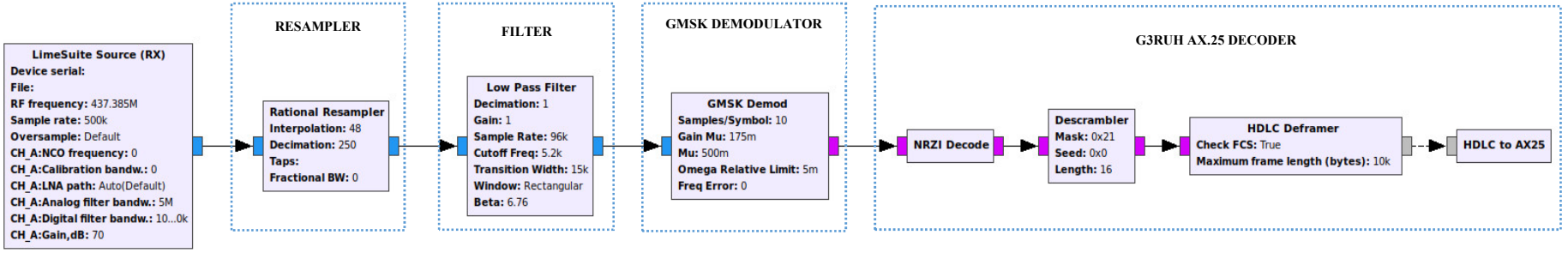


Figure 29 - GMSK G3RUH receiver implemented in GNU Radio.

4.4. SDR transmitter improvement

The proposed SDR transmitter has previously covered the Ten-Koh mission requirements and overcome the main limitations, however there are still two considerable constraints. The first one is the transmission data rate limitation due to the USB interface used in the LimeSDR mini module, it allows data transmission rate up to 19.2 MSPS (Mega Samples Per Second) which can be a limitation for satellite missions with more payload data generation. The second one is that there is not a possibility to reprogram the microcontroller on-flight, feature which would be very important for maintenance purposes and for correcting possible data corruptions due to radiation effects.

For the above reasons, a transmitter improvement is proposed and described hereunder in this numeral. It consists in the inclusion of an FPGA device in which the transmitter modulators are implemented. The FPGA device can be reprogrammed via the Raspberry Pi using a JTAG interface. The proposed hardware architecture is shown in the Figure 30.

On it is possible to observe the data interface connections which consist of a data line that provide the raw data from the Raspberry Pi to the FPGA and two selection lines that will be used to select the desired modulation scheme depending the application requirements. The modulated output is provided by eight-bit digital data which can be connected directly to any RF front end module for its transmission.

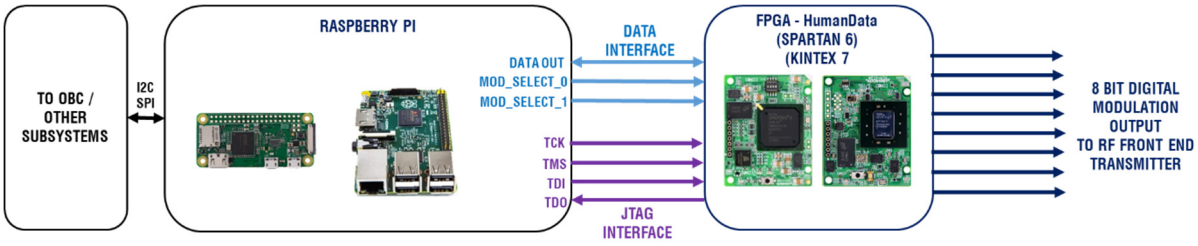


Figure 30 - SDR transmitter improvement hardware architecture

Regarding the software architecture, it is divided in two parts. The former is the implementation of the transmitter modulators into the FPGA and the latter is the implementation of the JTAG interface to reprogram the FPGA logic into the Raspberry Pi.

In the Figure 31, the transmitter improvement software architecture is presented. In the FPGA, four modulators (BPSK, FSK, QPSK and MSK) are implemented, those can all be programmed at the same time and also can be selected via GPIO selection pins data interface. Additionally, only one of the modulators can be programmed depending the mission requirements and to use the remaining free FPGA logic for other applications. Into the Raspberry Pi, the FPGA configuration files (bit files) are stored locally, those can be selected and reprogrammed into the FPGA logic via the JTAG controller depending the definitions and requirement defined in the user app. The advantage of the above is that it is possible to change the modulation schemes of the transmitter depending the mission requirements and also if there is a malfunction detected on the transmitter, the respective modulator scheme can be reprogrammed in order to recover the system due to a failure produced by a single event upset. Another advantage is that this architecture allows reprogramming the system in-flight due to the FPGA configuration files can be received from the ground station via uplink and stored locally into the Raspberry Pi for performing on-flight upgrades to the transmitter system.

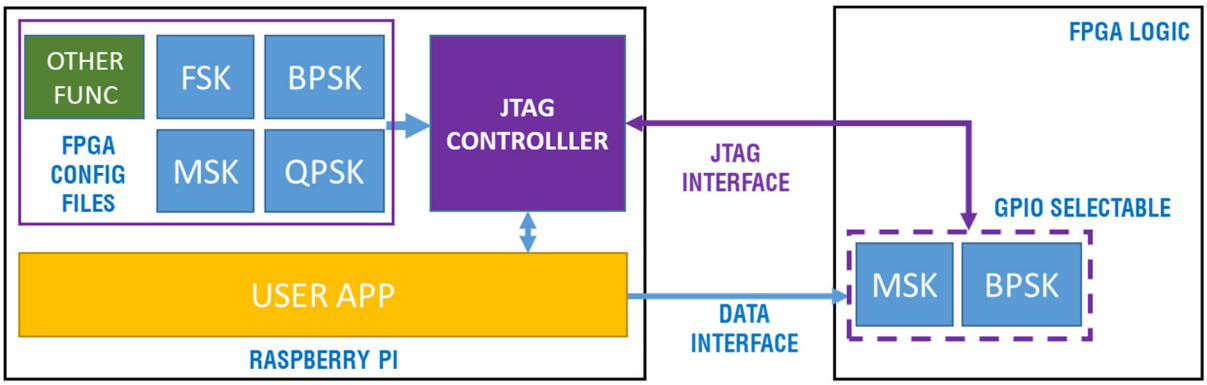


Figure 31 - SDR transmitter improvement software architecture

4.4.1. FPGA modulators design methodology

For the implementation of the respective transmitter modulators, the Xilinx Vivado IDE suite is used in conjunction with a MATLAB suite via the System Generator tool. This tool allows to design and simulate the modulators using the MATLAB Simulink environment to verify the correct functionality. After that, the design can be exported into a Xilinx Vivado project in which the design can be synthesized, implemented, programmed and simulated on the real FPGA environment.

In the Figure 32, a very basic example of the used methodology is shown. It is a sinusoidal signal generator which consists in the following blocks:

- A System Generator block used to configure all the parameters of the design like the FPGA model, the system clock settings, the hardware description language and the synthesis and implementation strategies.
- A DDS Compiler block used to configure all the parameters to generate a sinusoidal wave with a frequency of 1MHz.
- An FPGA output port used to assign a real FPGA pin in which the signal will be generated.

After configuring and connecting the respective blocks, a simulation can be executed to verify the correct functionality of the design.

The next step of the design methodology process is to generate the respective Xilinx Vivado Project. It can be possible pressing the "Generate" button into the System Generator block. This action will generate a complete project with all the required configurations that can be opened into the Vivado IDE suite. In the Figure 33 a screenshot of the generated project is shown, on it is possible to observe the exported VHDL code simulated, synthetized and implemented into the FPGA.

Finally, the project can be programmed into the FPGA and it is possible to debug and make real time simulations using the real hardware to finishing to verify the correct functionality of the design.

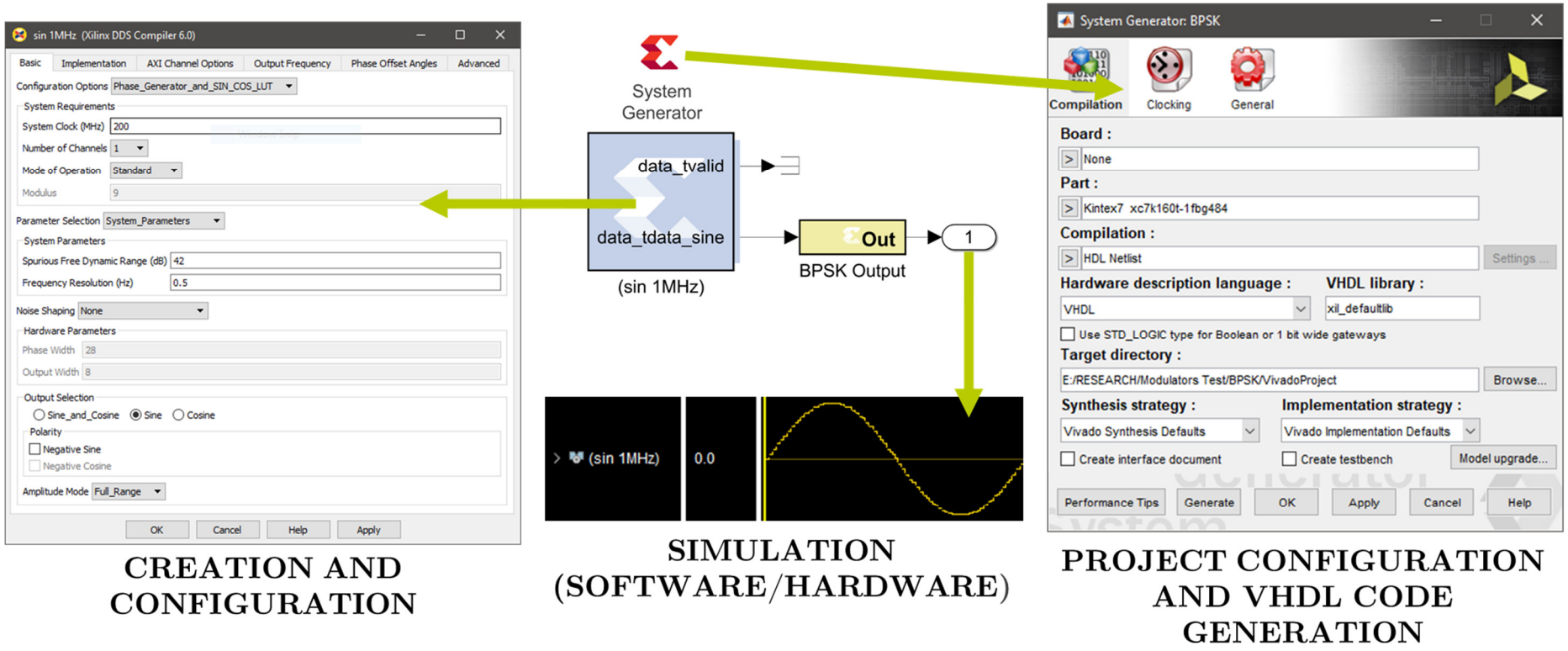


Figure 32 - FPGA design methodology example using Xilinx System Generator and MATLAB Simulink

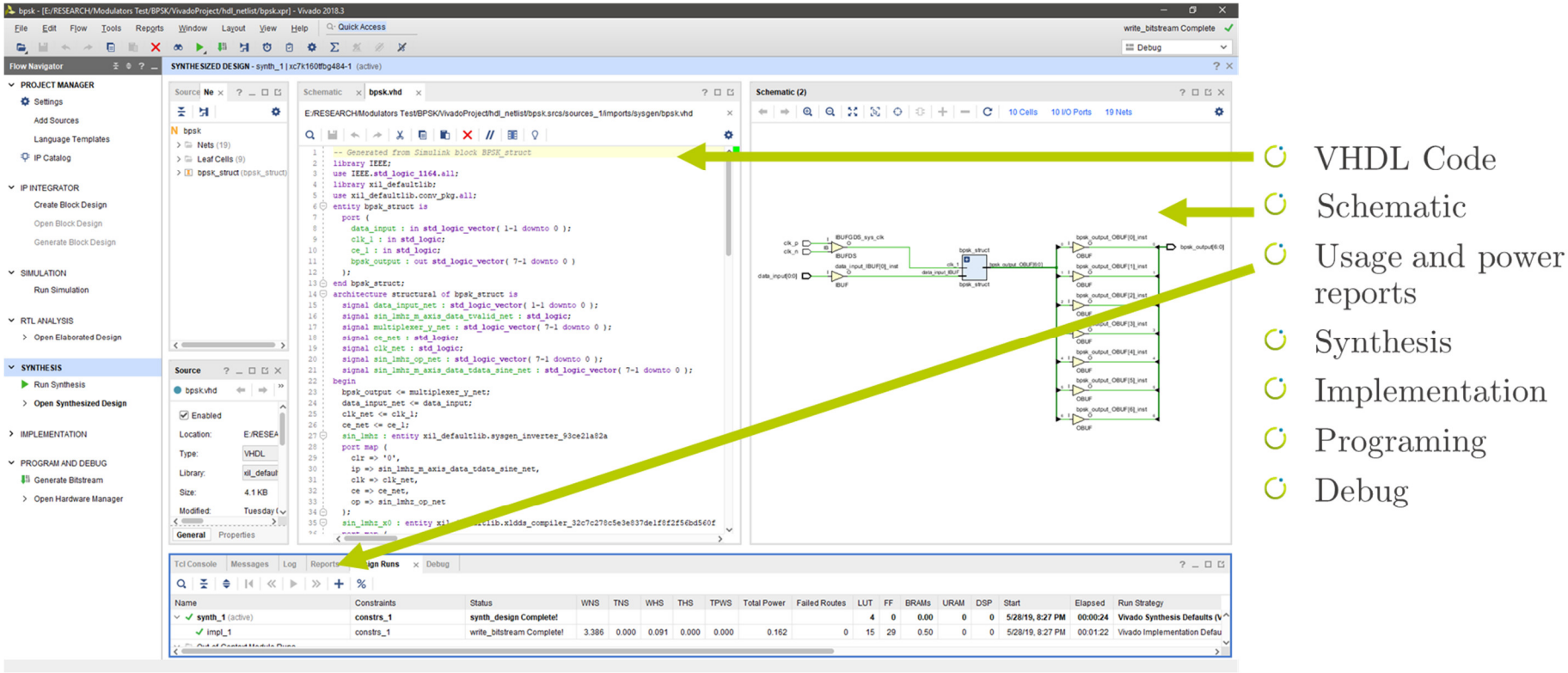


Figure 33 – Generated Xilinx Vivado IDE suite from the System Generator design methodology example

4.4.2. FPGA modulators implementation

In this section, the design of four modulators is presented. The respective System Generator block diagram generated in MATLAB Simulink and the simulation results is presented for a BPSK, FSK, QPSK and MSK modulators.

- **Binary Phase-Shift Keying (BPSK)**

The BPSK is a phase-based modulation where the phase of the carrier wave is shifted by π (180 degrees) to represent a "1" or "0" binary values [30]. The signal waveform output of the modulator is given as:

$$S_1(t) = \begin{cases} \sqrt{\frac{2E_b}{T_b}}\sin(2\pi f_c t) & , \ input_{bit} = 0 \\ -\sqrt{\frac{2E_b}{T_b}}\sin(2\pi f_c t) & , \ input_{bit} = 1 \end{cases}$$

Equation 11 - BPSK modulation waveform output

Where T_b is the bit period and $E_b = \int_0^{T_b} s_1^2(t)dt \quad 0 \leq t < T_b$ is the energy per bit.

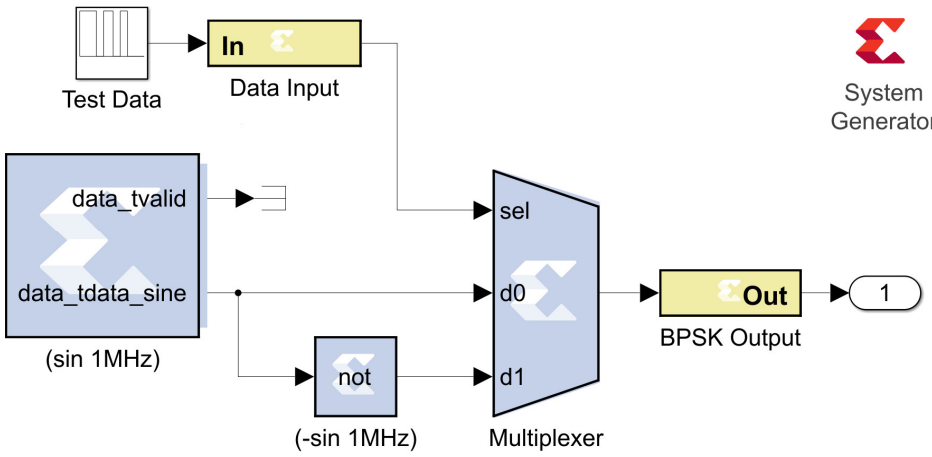
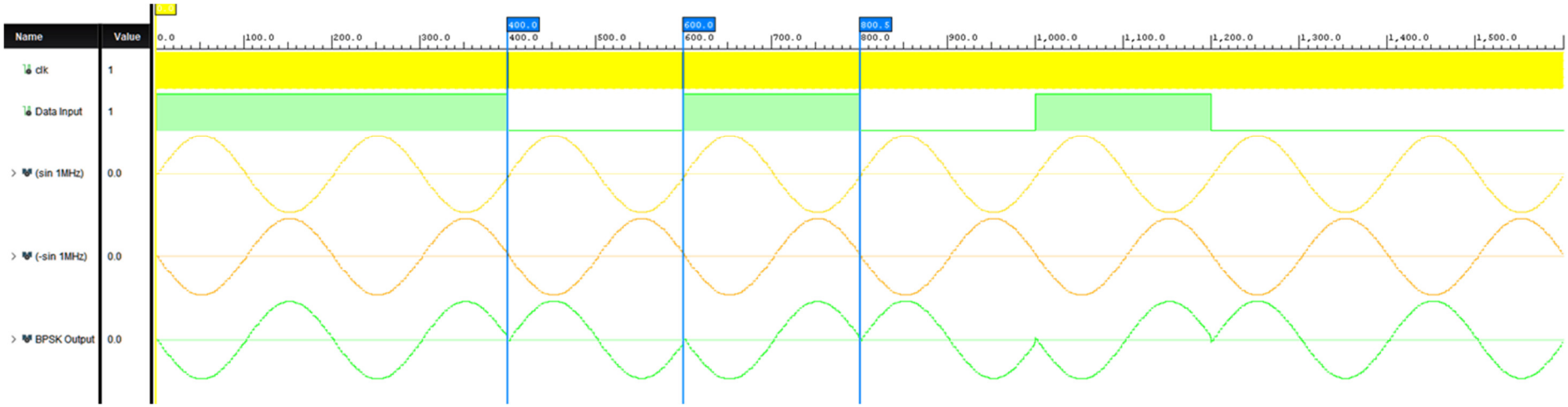


Figure 34 - BPSK modulator implementation



Resource	Utilization	Available	Utilization %
LUT	15	101400	0.01
FF	29	202800	0.01
BRAM	0.50	325	0.15
IO	10	285	3.51

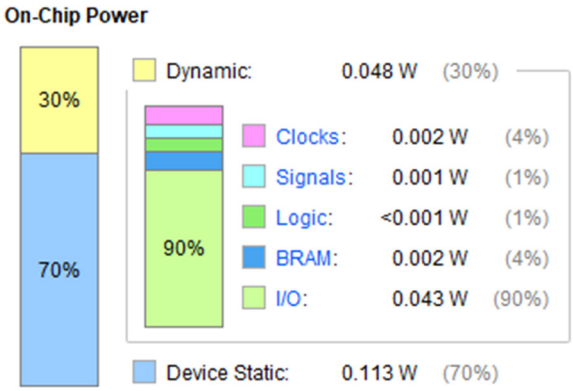


Figure 35 - BPSK modulator simulation results

The implementation of the BPSK modulator in the System Generator is shown in the Figure 34. On it is possible to observe a DDS compiler generator that is configured to generate a sinusoidal signal with a frequency of 1MHz, that signal is connected to the first input of a multiplexer (d0) and represents the carrier wave when the data input bit is "0". The same sinusoidal signal is inverted using the logical "not" block and it is connected to the second input of the multiplexer (d1). The data input is connected to the selector input of the multiplexer, then, depending of its binary value, the signal is commuted at the output.

In the Figure 35, the simulation results of the modulator are shown. The data input bits, the sinusoidal wave and the modulation output signals are plotted. Here, it is possible to see how the phase is shifted π (180 degrees) when the input bits change which is the expected behavior of the BPSK modulator. Also, the FPGA logic utilization and the estimated on-chip power consumption is shown.

- **Binary Frequency Shift Keying (FSK)**

The FSK is a frequency-based modulation where the frequency of the carrier signal varies with the value of the binary data input in which "0" corresponds to one frequency value and "1" corresponds to another frequency value. The modulated signal output is given by:

$$\begin{cases} S_1(t) = \sqrt{\dfrac{2E_b}{T_b}}\cos(2\pi f_1 t), \ \ input_{bit} = 0 \\ S_2(t) = \sqrt{\dfrac{2E_b}{T_b}}\cos(2\pi f_2 t), \ \ input_{bit} = 1 \end{cases}$$

Equation 12 - FSK modulation waveform output

Where T_b is the bit period ($0 \leq t < T_b$) and $E_b = \int_0^{T_b} s_1(t)s_2(t)dt = 0$ is the energy per bit.

The implementation of the FSK modulator in the System Generator is presented in the Figure 36. The implementation is based on a multiplexer as same as the BPSK implementation with the difference that in this case, there are two DDS Compiler generator blocks in charge of generating the sine carrier waves at 1MHz and 2MHz respectively which are connected in the inputs (d0 and d1). The modulator output will change between the different frequency carrier waveforms depending of the binary value of the data input connected with the selector port of the multiplexer.

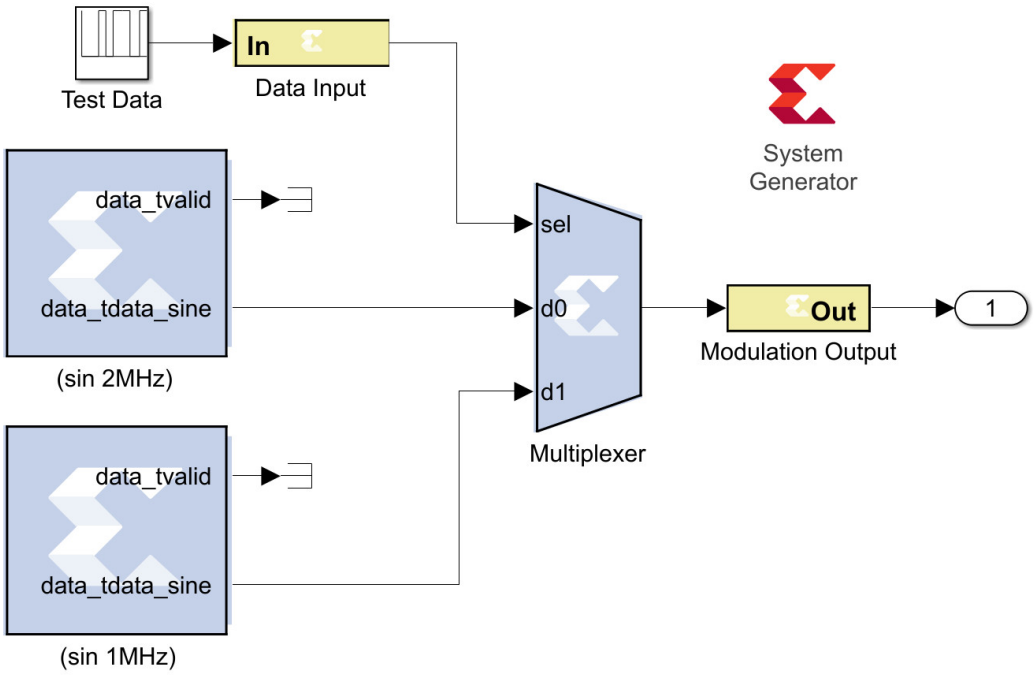
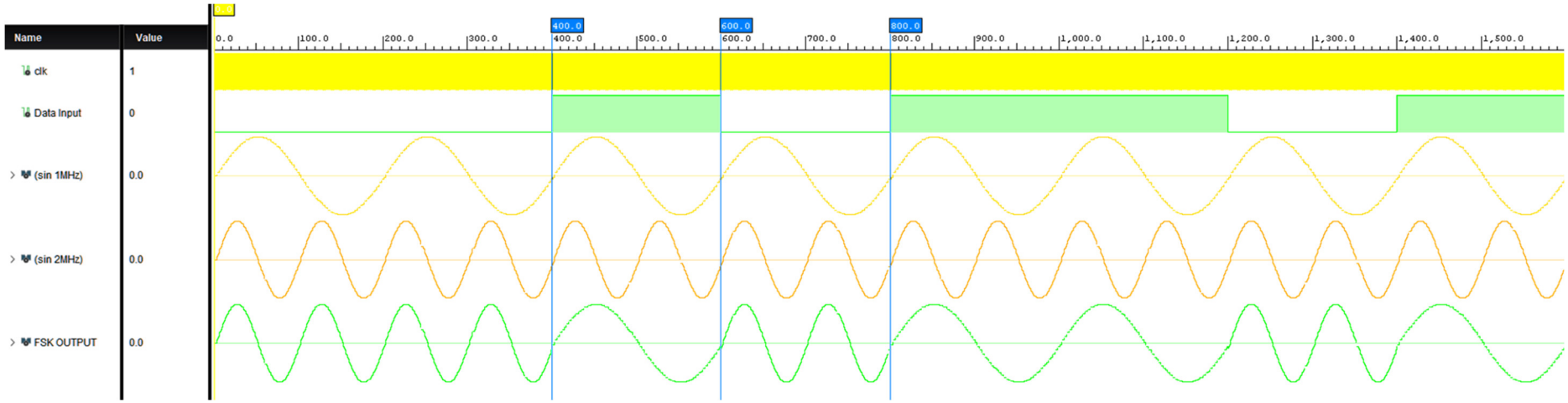


Figure 36 - FSK modulator implementation

In the Figure 37, the simulation results of the FSK modulator are shown. The data input bits, the two sinusoidal waveforms and the modulation output signals are plotted. Here, it is possible to observe how the carrier wave signals are commuted when the binary value of the input changes, which is the expected behavior of the FSK modulator. Also, the FPGA logic utilization and the estimated on-chip power consumption is presented.

.



Resource	Utilization	Available	Utilization %
LUT	27	101400	0.03
FF	58	202800	0.03
BRAM	1	325	0.31
IO	10	285	3.51

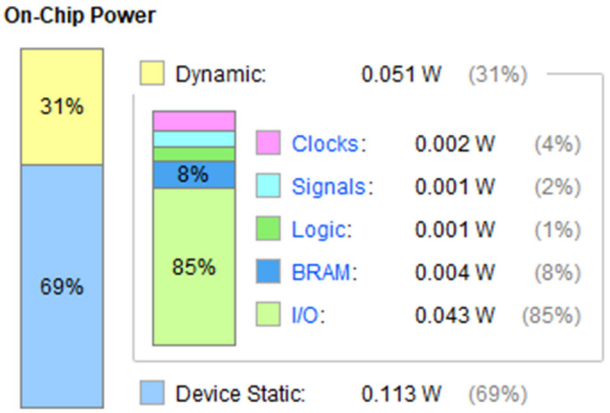


Figure 37 - FSK modulator simulation results

- **Quadrature Phase-Shift Keying (QPSK)**

As explained previously, a BPSK modulator is defined by two carrier signals with a phase shifting of π (180 degrees) which represents the binary data input ("0" or "1"). In the case of the QPSK modulator, it is defined by four carrier signals with a phase shifting of π/2 (90 degrees) in which each signal represents a pair of binary data input ("00", "01", "10" and "11"). It means that two data bits are transmitted per carrier signal duplicating the data transmitted by the BPSK modulator. The QPSK modulator output is defined by:

$$S(t) = \sqrt{\frac{2E_b}{T_b}}[\cos\phi(t)\cos(2\pi f_c t) - \sin\phi(t)\cos(2\pi f_c t)],$$

$$\phi(t) = f\begin{cases} 0 \rightarrow 00 \\ \pi/2 \rightarrow 01 \\ -\pi/2 \rightarrow 10 \\ \pi \rightarrow 00 \end{cases}$$

Equation 13 - QPSK modulator waveform output

Where T_b is the bit period ($0 \leq t < T_b$) which is the double than in the BPSK case.

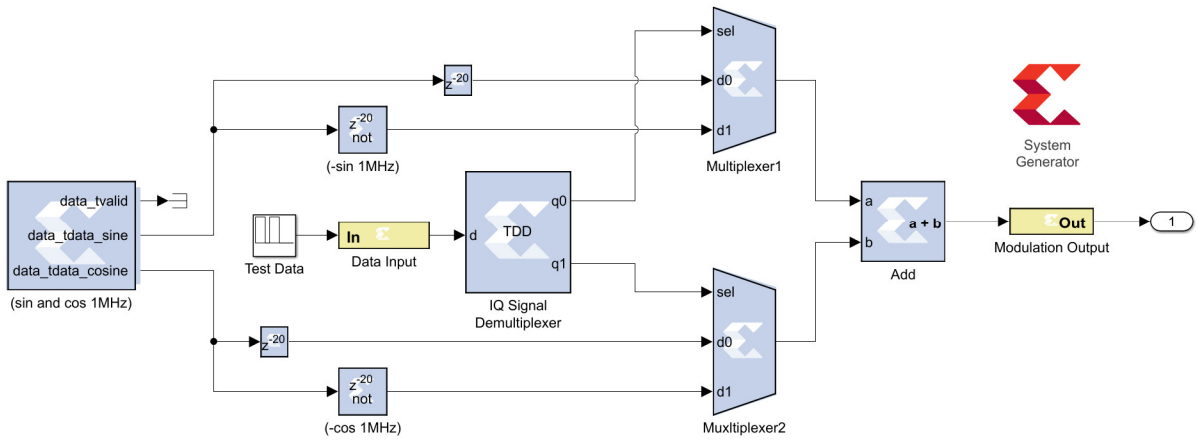


Figure 38 - QPSK modulator implementation

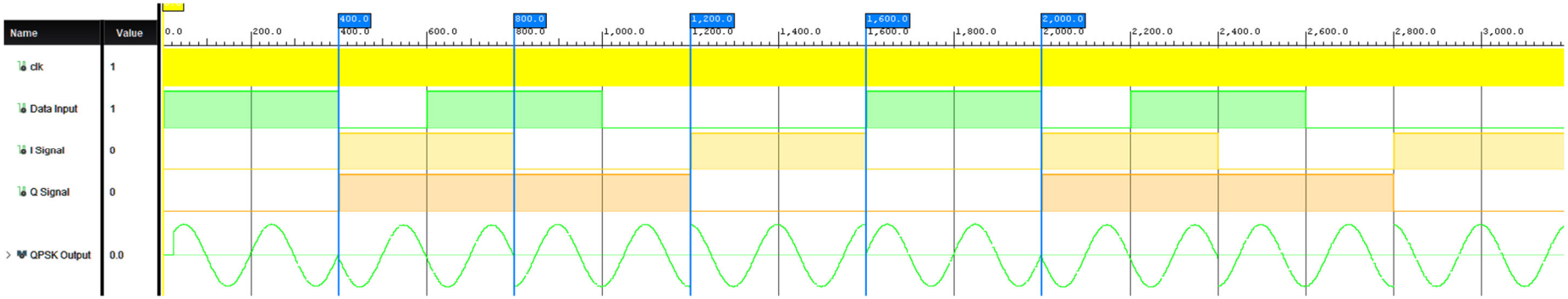
The implementation of the QPSK modulator in the System Generator is presented in the Figure 38. It consists basically in two BPSK modulators implementation, same as previously explained. The DDS Compiler generator block is in charge of generating the sine and cosine waves. Each signal is inverted using an inverter "not" block and the resulting two signals are connected to the inputs (d0 and d1) of a different multiplexer (1 and 2). For dividing the data input bit stream into the even and odd components, a Time Division Demultiplexer (TDD) block is used, it acts as a serial to parallel (2 bit) converter which provides the selector outputs to control the output of the multiplexers that represent the I and Q components of the modulator signals. Finally, for obtaining the QPSK modulated signal, the I and Q components are added using an addition block.

In the Figure 39, the simulation results of the QPSK modulator are shown. In the first place, the data input bit stream and the I and Q components resulting in the output of the TDD block are plotted for verifying the correct 2-bit parallel conversion. Here, it is possible to observe clearly that the period of the I and Q signals are two times the period of a single bit from the data stream. Finally, it is possible to see how the carrier wave phase varies correctly depending on the I and Q combinations as explained on the Equation 13.

In addition, on the Figure 39, the FPGA logic utilization and the estimated on-chip power consumption is presented.

- **Minimum-Shift Keying (MSK)**

The QPSK modulation presented previously is one of the most used in satellite communications due to the better spectral efficiency that offers the possibility to send two bits trough one carrier signal. However, in a satellite communications system, RF amplifiers are typically used to increase the power output of the transmitted signals.



Resource	Utilization	Available	Utilization %
LUT	44	101400	0.04
FF	61	202800	0.03
BRAM	0.50	325	0.15
IO	11	285	3.86

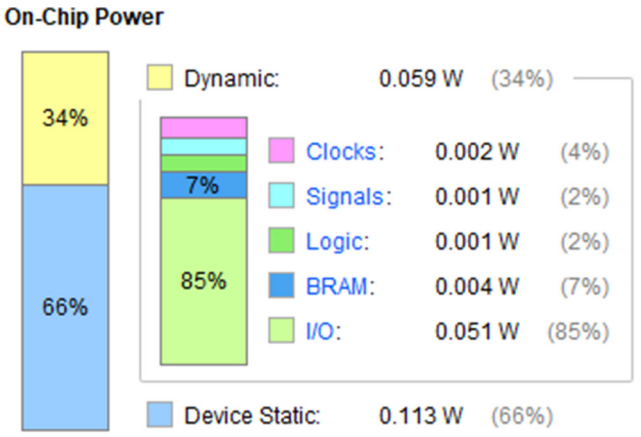


Figure 39 - QPSK modulator simulation results

When those amplifiers operate near to the saturation, those generally exhibits a nonlinear output causing distortion to non-continuous modulated signals [30]. That is the case of the BPSK and QPSK modulators due to the carrier wave phase changes. To overcome the above, continuous or near-continuous phase modulated signals are desired like the OQPSK (Offset QPSK) or MSK modulators.

The OQPSK (Offset QPSK) modulator is a variation of the QPSK modulator where the I and Q bitstreams are offset in time by one time period T_b. It results in a phase change of only 0 and $\pm\pi/2$ which means that the modulation signal is nearly continuous. [30]. Additionally, if the I and Q signals of the OQPSK modulator are shaped with a sinusoidal pulse, then the phase changes became completely continuous and it is the principle of the MSK modulator.

Taking into account the above, the MSK modulated signal can be represented as same as the QPSK modulator (Equation 13) whit the addition of the sinusoidal shaped pulses to the I and Q components which result in the following expression:

$$S(t) = \sqrt{\frac{2E_b}{T_b}}\left[\cos\phi(t)\cos\left(\frac{\pi t}{2T_b}\right)\cos(2\pi f_c t) - \sin\phi(t)\sin\left(\frac{\pi t}{2T_b}\right)\cos(2\pi f_c t)\right]$$

Equation 14 - MSK modulation waveform output

Where $\cos(\pi t/2T_b)$ is the sinusoidal shape of the I component and $\sin(\pi t/2T_b)$ is the sinusoidal shape of the Q component.

The implementation of the MSK modulator in the System Generator is presented in the Figure 40. It consists in a TDD block that takes the binary data input to generate the binary I and Q signals with bit period $2T_b$; after, a combination of a DDS Compiler and an inverter "not" blocks which are used to generate two 0.25MHz sinusoidal signals with a phase difference of π (180 degrees). Those sinusoidal signals are connected to the inputs of two multiplexers which are controlled by the binary I and Q signals generated by the TDD block. Doing the above, the sinusoidal shaped I and Q signals are

generated as the MSK modulator requires. Next, the resulting signals are multiplied with a 1MHz sine and cosine signals generated by another DDS Compiler block respectively. Finally, the two resulting signals are added using an addition block generating the MSK modulated signal.

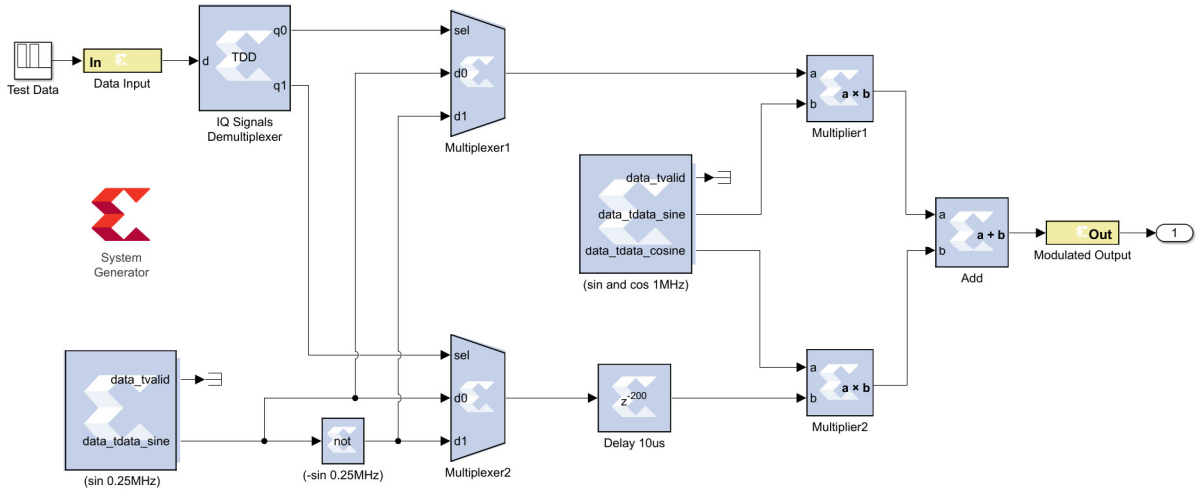
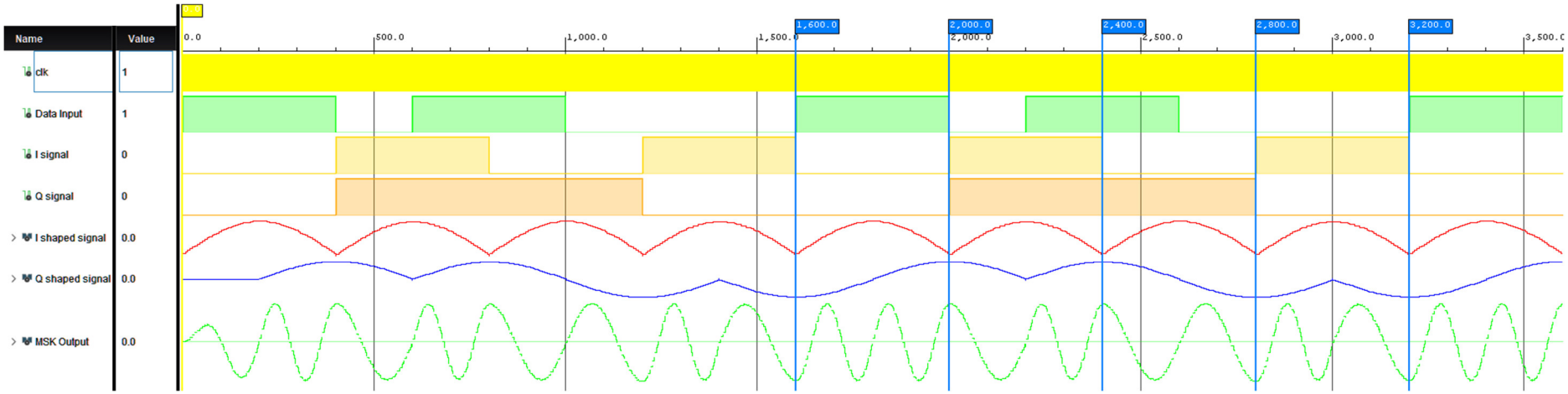


Figure 40 - MSK modulator implementation

In the Figure 41, the simulation results of the MSK modulator are presented. In the first place, the data input bit stream and the I and Q components resulting in the output of the TDD block are plotted for verifying the correct 2-bit parallel conversion, same as in the QPSK modulator simulation. Next, the sinusoidal shape I and Q signals are plotted; here, it is possible to observe the T_b offset of the Q component to comply with the OQPSK modulator property as explained before. Finally, we can notice how the resulting MSK modulated signal varies the phase changes are completely continuous as expected. In addition, on the Figure 41, the FPGA logic utilization and the estimated on-chip power consumption is presented.

The presented implementations until here can be implemented individually into the FPGA logic. Then, the configuration files generated can be used by the Raspberry Pi application to reprogram the FPGA logic depending the mission needs.



Resource	Utilization	Available	Utilization %
LUT	100	101400	0.10
LUTRAM	46	35000	0.13
FF	136	202800	0.07
BRAM	1	325	0.31
DSP	2	600	0.33
IO	10	285	3.51

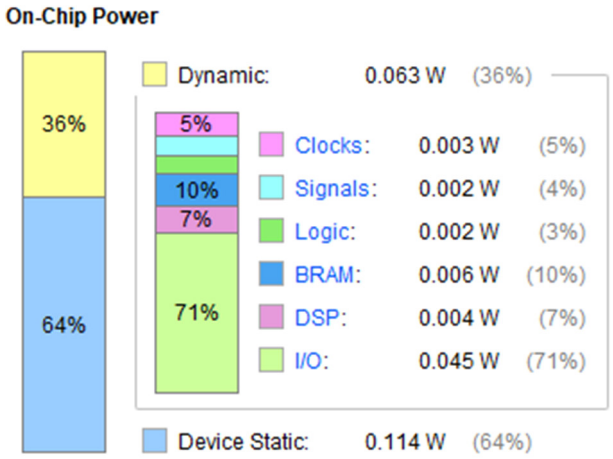


Figure 41 - MSK modulator simulation results

Next, an integrated implementation where the four presented modulators are included in the case that the Raspberry Pi application can change the modulation scheme via GPIO selection pins.

- **Integrated modulators implementation**

In the Figure 42, a complete integration of the previous modulators implementations is presented. As we can see, the DDS Compiler block that generates the 1MHz sine and cosine signals is shared for all modulators as same as the TDD block is shared for the generation of the I and Q signals required for the QPSK and MSK modulators. Each modulator block consists in the same implementations presented and explained previously, however, to be more specific, the internal block diagrams that conforms each modulator are detailed in the appendix 1. Finally, to be able to select the desired modulation output, a multiplexer is added to the implementation. All the outputs of the modulators are connected to the multiplexer inputs and a modulator selection is performed via a selection port that consist of two lines which are controlled externally; depending the input combinations of that port, each modulator can be selected.

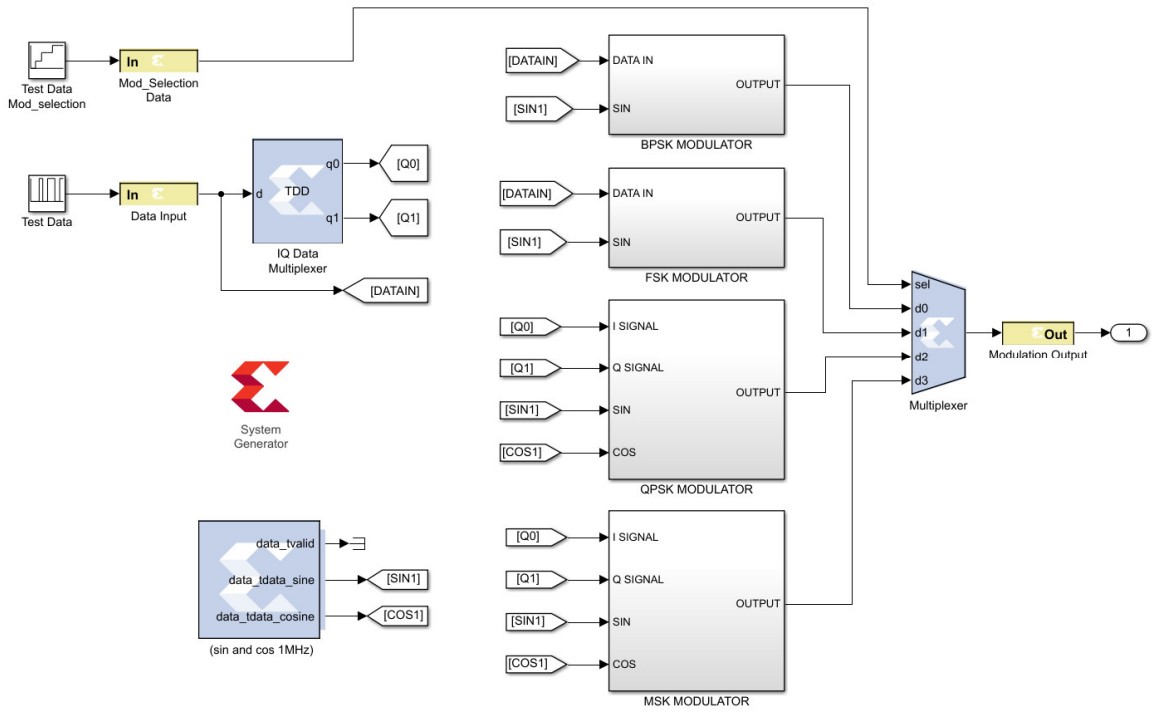
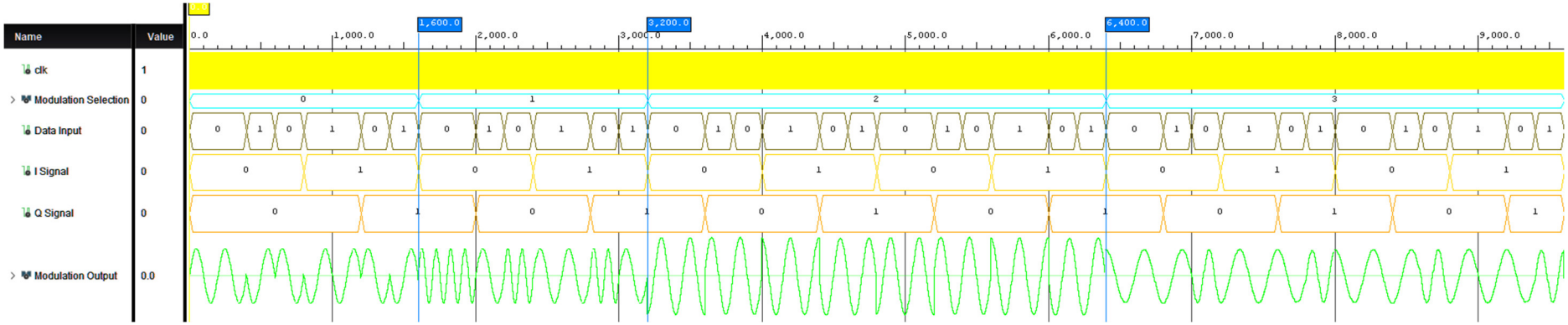


Figure 42 - Integrated modulators implementation



Resource	Utilization	Available	Utilization %
LUT	141	101400	0.14
LUTRAM	46	35000	0.13
FF	173	202800	0.09
BRAM	1.50	325	0.46
DSP	2	600	0.33
IO	13	285	4.56

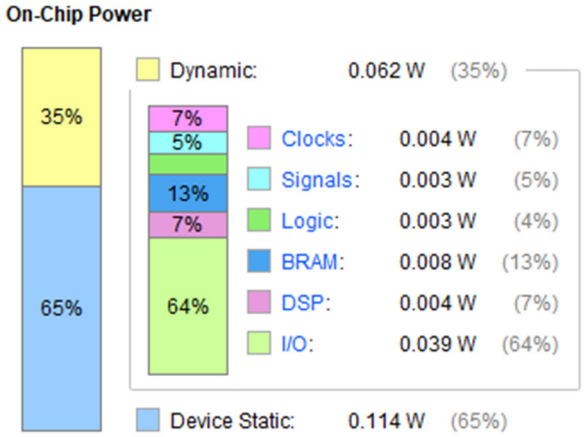


Figure 43 - Integrated modulators simulation results

The result of the above implementation is presented in the Figure 43. In the plot the following signals are shown:

- The decimal value of the selection port, if the value is “0” (00) the BPSK modulator is selected, if the value is “1” (01) the FSK modulator is selected, if the value is “2” (10) the QPSK modulator is selected and finally, if the value is “3” (11) the MSK modulator is selected.
- The data input serial stream
- The binary I and Q components generated by the TDD block
- The modulated output in which it is possible to observe the BPSK output from 0 to 1,600ns when the value of the selector input is “0”, the FSK output from 1,600 to 3,200ns when the value of the selector is “1”, the QPSK output from 3,200 to 6,400ns when the value of the selector is “2” and finally, the MSK output from 6,400 to 9,600ns when the value of the selector is “3”

As it is possible to observe, the simulation of the complete integration of the four modulators works according to the explained previously. Additionally, on the Figure 43, the total FPGA logic utilization and the estimated on-chip power consumption is presented.

4.4.3. FPGA reprogramming system

As shown in the Figure 30 and the Figure 31, the complete SDR transmitter optimization consists into the FPGA part in which the modulators can be programmed and executed and the Raspberry Pi part in which the reprogramming module is implemented via a JTAG interface and can be controlled by the application app. The implementation of the modulators into the FPGA part was already described, now, the FPGA reprogramming system implemented into the Raspberry Pi is explained.

The base of the system is a JTAG controller which is in charge to control and execute the reprogramming process using the JTAG protocol. JTAG (Join Test

Action Group) is the name of the group that developed the IEEE 1149.1 standard used for verifying, testing and debugging printed circuit boards via Test Access Ports (TAP), the complete description of the standard can be consulted on [31]. A good summary of the JTAG standard is presented in [32] in which the most relevant features and characteristics are explained as well as the flow charts about the TAP controller state machine.

The JTAG standard is used for Xilinx FPGAs also for programming, debug and readback their devices, usually using their Platform Cable USB II device [33]. However, on [34] there are a clear explanation about the Xilinx In-System Programming for their devices using bit configuration files, Serial Vector Format (SVF) and Xilinx SVF (XSVF) files.

Due to the Raspberry Pi is a device capable to run under Linux distribution, there are some tools that allows to use it as a JTAG standard controller using the GPIO drivers as TAP controller, for example, UrJTAG [35] and OpenOCD [36]. Both have the capability of executing SVF files conforming to the JTAG standard in order to program or debug Xilinx FPGAs, however, not all SVF commands are compatible and it generates some errors executing the SVF files generated by Xilinx IDEs (Vivado and ISE). Another limitation using SVF files is the size which, depending the logic utilization, can be in the order of MB which is a problem if we want to send the file via uplink to the satellite for maintenance purposes.

After testing the both options, the tool used for this implementation was OpenOCD because additionally to the SVF player feature, it has the possibility of programming the configuration bit file directly to the FPGA which decrease the time needed for the FPGA programming for flashing the device and also allows to send the configuration file easily via uplink to the satellite due to the typical size of the bit files is in the order of hundreds of kB.

The block diagram of the FPGA reprogramming system is shown in the Figure 44. The system was tested for two Xilinx FPGA families, the Spartan 6 (included

in the HumanData XCM-110-LX75 module [37]) and the Kintex 7 (included in the HumanData XCM-112-160T module [38]). According with the Xilinx application note "Using SPI Flash with 7 Series FPGAs" [39], an SPI flash memory is recommended to store the configuration bit file of the FPGA in order to reprogramming the logic every time the power is shut down. Also, a typical connection between the FPGA and the flash memory is shown and it explains how that memory can be programmed via JTAG standard. Both tested modules already include the flash SPI memory with their respective connections as shown in detail in the mentioned application note.

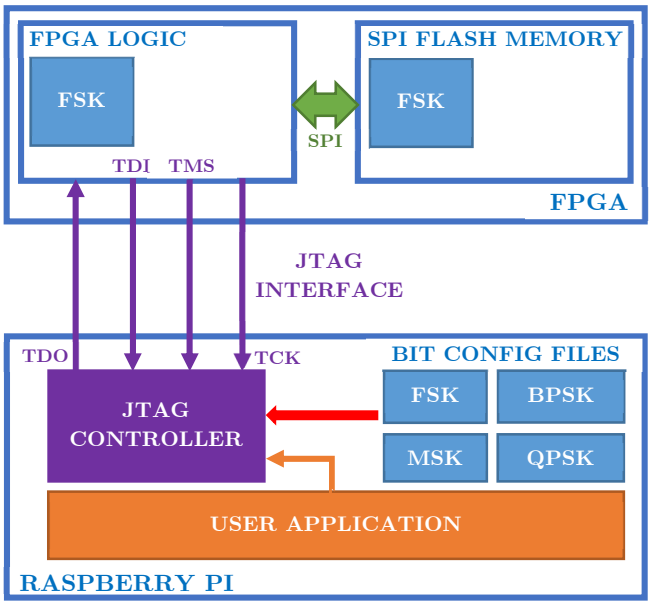


Figure 44 - FPGA programmer block diagram

In the Raspberry Pi side, the bit configuration files of the implemented modulators for the FPGA are stored into the SD card, then, the JTAG controller is in charge to program that files only into the FPGA logic or into the FPGA logic plus the SPI flash memory depending the user application needs. As mentioned previously, the selected tool for the implementation is the OpenOCD, then, the first step is to compile and install it into the Raspberry PI. To do that, the followed procedure was based the official repository [40] and the tutorial created by Ardafruit for ARM microcontrollers [41].

In [42], an extensive user manual about all the OpenOCD functions and commands is presented. The tool needs two configuration (.cfg) files, the first is the interface file which includes the details and definitions about the hardware used as a JTAG interface (e.g. a JTAG adapter or specifically in this implementation, the Raspberry Pi GPIOs); the second is the target file which includes the definition of the JTAG TAPs of the device that OpenOCD should control/debug/program (e.g. ARM CPU or specifically in this implementation the Xilinx FPGAs).

Regarding the interface configuration file, by default, the OpenOCD repository includes three files to use a Raspberry Pi as a JTAG interface in the "interface" folder, one of them uses the sysfs driver to access to the GPIOs which is compatible with all Pi versions and the other two files use the bcm2835 driver, one of them is compatible only with the Pi version one or the Pi Zero and the other is compatible with the Pi version two only. After test the two available options with the Pi Zero and the Pi 3B+, in the case of the Pi Zero, the most reliable and fastest driver was the bcm2835 and in the case of the Pi 3B+, the most suitable and fastest driver was the sysfs. The details and differences of the both drivers are presented in [43].

Regarding the target files, by default, the OpenOCD repository includes the target files for the Xilinx Spartan 6 and 7 family FPGAs in the "cpld" folder. For this implementation the default files worked successfully without any modification.

Until now, OpenOCD with the default configuration allows the possibility to debug and reprogram the FPGA logic, however, it is not possible to reprogram the SPI flash memory attached to the FPGA device. In order to achieve that, a custom interface configuration file should be created, including the TAP instructions to program the SPI flash memory through the FPGA itself, programing in the logic a bscan-SPI bitstream controller to manage the flash memory programing via the SPI protocol. The custom configuration file used in

this implementation was created on the base of the configuration files included in the netv2mvp-scripts repository [44], and the final version used for all the implementation and test are shown in the appendix 2 with respective comments explaining the functionalities. The bscan-SPI bitstream files used to program the FPGA logic temporarily for performing the SPI flash memory programing task can be found in the Quartiq repository [45] ("bscan_spi_xc7k160t.bit" for the Kintex 7 FPGA and the "bscan_spi_xc6slx75.bit" for the Spartan 6 FPGA).

The test and results of the FPGA reprogramming system are shown in the next chapter.

CHAPTER 5 - RESULTS AND DISCUSSION

In order to test and characterize the Ten-Koh system and the proposed SDR architecture, some key parameters were chosen to be measured and compared.

The parameters are: for the receiver part, the performance for different signal strengths in order to obtain the practical sensitivity values and the maximum number of received bytes per AX.25 packet; for the transmitter part, the power output and the maximum number of bytes allowed to send per AX.25 packet.

A power consumption test was performed on the two systems executing the receiving and transmitting functions at the same time and in the specific case of the proposed SDR implementation, it was implemented in a different Raspberry Pi model in order to compare the different power consumptions depending the used model.

On the other hand, some simulations were performed in order to have an idea about the performance of the proposed SDR implementation for the signal-to-noise ratio (SNR) and the Packet Error Rate (PER) modeling an Additive White Gaussian Noise (AWGN) channel between the transmitter and the receiver using the GNU Radio Companion graphical interface and the signal processing blocks available to perform that kind of simulations. Also, it is possible to observe the modulated and demodulated signals in the presence of controlled noise level in the channel and how it affects the constellation diagram and the entire system performance.

Finally, for the FPGA reprogramming system implemented on the Raspberry Pi using OpenOCD, a test was performed in order to verify the required time for programing the different modulators implemented for the Kintex 7 and Spartan 6 FPGAs, to compare the results for the Zero and the 3B+ models and define if the system can be suitable for using on-board in a nanosatellite mission.

5.1. Receiver sensitivity test

In the receiver part, one of the most important parameters for characterization is the sensitivity. This value, usually varies depending the modulation scheme, power supply quality, component tolerances, etc. In order to perform this test, the configuration shown in the Figure 45 was used. In this case, a base band signal was generated using the same TNC used in the Ten-Koh ground station, the Kantronics KPC9100+, in the case of the 1,200bps AFSK signal, it is obtained from the port 1 and in the case of the 9,600bps GMSK signal, it is obtained from the port 2. That signal is connected to the modulation input port into the RF signal generator (Hewlett Packard 8656B) which is in charge to put the base band signal in the desired RF carrier wave. The frequency value was fixed at 435.2 MHz and the power of the carrier wave varies from -100 dBm to -120 dBm. Finally, the RF output signal is connected directly to the input of the receiver module using a coaxial cable and the received data packets can be observed in the GNU Radio companion in the case of the proposed SDR system and in the HyperTerminal in the case of the Ten-Koh system via UART serial communication port. The test consists in send 250 AX.25 packets from the TNC to the receiver at different power level steps and monitoring how many packets were successfully decoded into the receiver. At the power level when no packets are received, it is the sensitivity value of the receiver module.

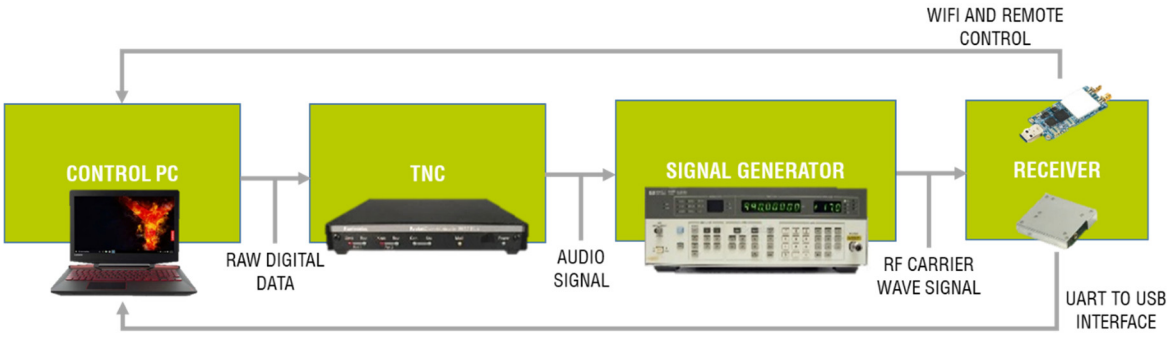


Figure 45 - Receiver sensitivity test configuration.

The results of this test are shown in the Figure 46 and Figure 47, in the first one, it is possible to observe the number of receiving packets in function of the

power input strength into the receiver and in which it is possible to determine the sensitivity threshold which is the RF power level where the receiver start to receive valid packets. Those values are -119.5 dBm for the Ten-Koh AFSK receiver, -112 dBm and -110 dBm for the proposed SDR AFSK and GMSK receivers respectively. However, for defining a practical receiver sensitivity value usable for link budget calculations and for determining the receiver performance, is better to calculate the Packet Error Rate (PER) in percentage which is calculated using the Equation 15.

$$PER_{\%} = \left(\frac{N_{Error\_packets}}{N_{Total\_packets}}\right) \cdot 100$$

Equation 15 - Packet Error Rate calculation.

The PER performance for Ten-Koh and SDR proposed systems are shown in the Figure 47. For optimal packet communication performance in a nano-satellite mission like Ten-Koh, the practical receiver sensitivity threshold value can be chosen when the PER is 1%. Then, in the case of the Ten-Koh AFSK receiver, the measured value is -115.5 dBm and in the case of the proposed SDR AFSK and GMSK receivers the values are -108.5 dBm and -106 dBm respectively.

The testing was performed using the ground station already implemented and in operation with the Ten-Koh satellite. The software used to send and receive the data is exactly the same used to do the same duty in the daily satellite operations. For receiving the data at the ground station, the AGW Online KISS decoder [46] was used and for sending data, the control software development for the Ten-Koh team was used.

5.2. Receiver Signal-to-noise ratio (SNR) simulation

The signal-to-noise ratio (SNR) is the relation between the received RF signal strength and the noise signal strength presented at the input of the communication receiver.

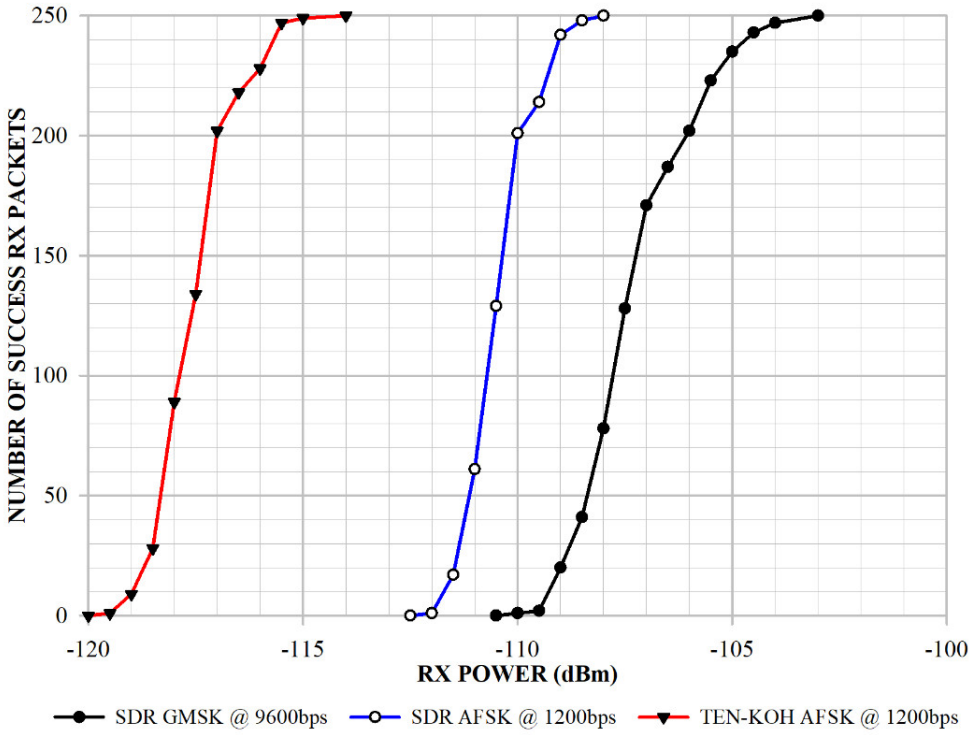


Figure 46 - Number of received packets in function of the received RF power.

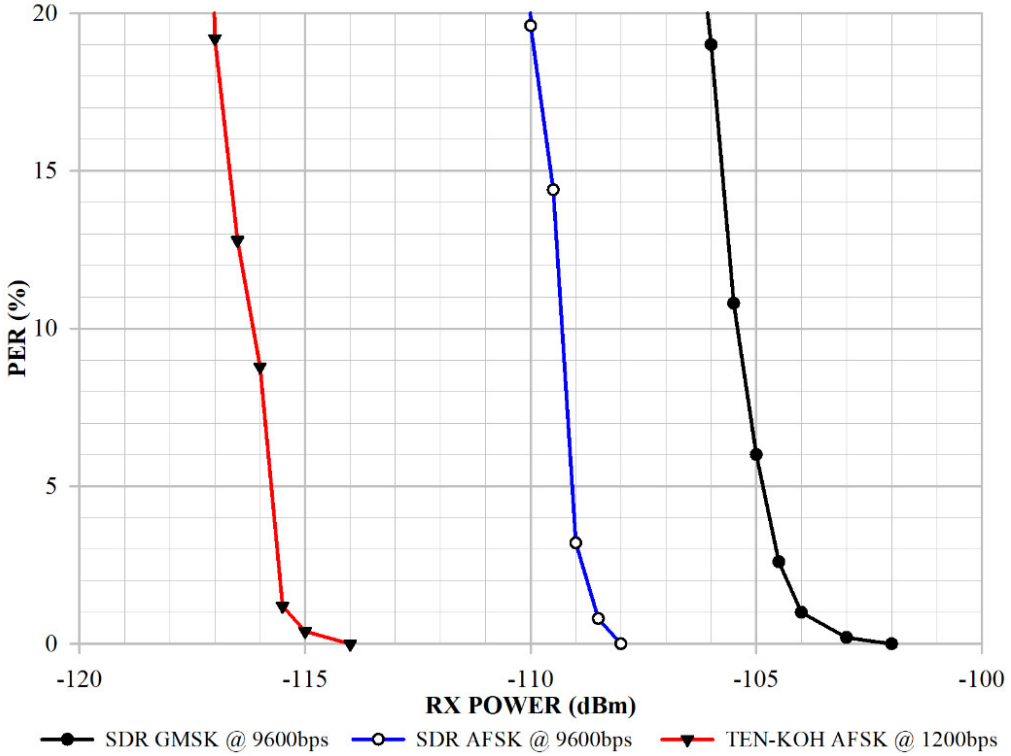


Figure 47 - Packet Error Rate (PER) in function of the received RF power.

When the noise signal strength becomes comparable with the received signal strength, the system performance starts to decrease which means that the probability to lose data packets starts to increase. In a satellite communication

link, the noise is produced by different sources, e.g. by the atmosphere that generates thermal noise and by interfering signals generated from other communication systems working in near frequency bands. For that reason, it is difficult to measure the SNR and the performance of the communication system due to this effect. However, GNU radio includes a special signal blocks to generate Gaussian noise which makes possible the simulation of the effects of an Additive white Gaussian noise (AGWN) channel in the system. For this simulation, the implemented diagrams in GNU radio are shown in the Figure 48 (for the GMSK transceiver) and in the Figure 49 (for the AFSK transceiver). The methodology is to create a loopback between the transmitter and the receiver, excluding the LimeSuite RF blocks to avoid undesirable/uncontrollable noise sources and including the respective signal blocks to add the AGWN channel effects. In the real case, the noise is introduced to the system via the RF signal and the expression for the SNR is given by the Equation 16.

$$SNR(dB) = 10 \cdot \log_{10}\left(\frac{P_{Signal}}{P_{Noise}}\right)$$

Equation 16 - Signal-to-Noise Ratio calculation.

However, in the simulation, the RF modules cannot be included because the GNU radio blocks can introduce random Gaussian noise only for baseband signals, then, the method for simulating the same effects in the system is to add the noise to the baseband modulated signals. In the case of the GMSK system, the modulator output signal is complex type; then, it is possible to use the Channel Model block to include the noise effects in the system as showed in the Figure 48. On the other hand, in the AFSK system case, the modulator output signal is float type and it is necessary to use a Noise Source block to add the AGWN channel effects to the system as shown in the Figure 49 since the Channel Model block only works for complex signals.

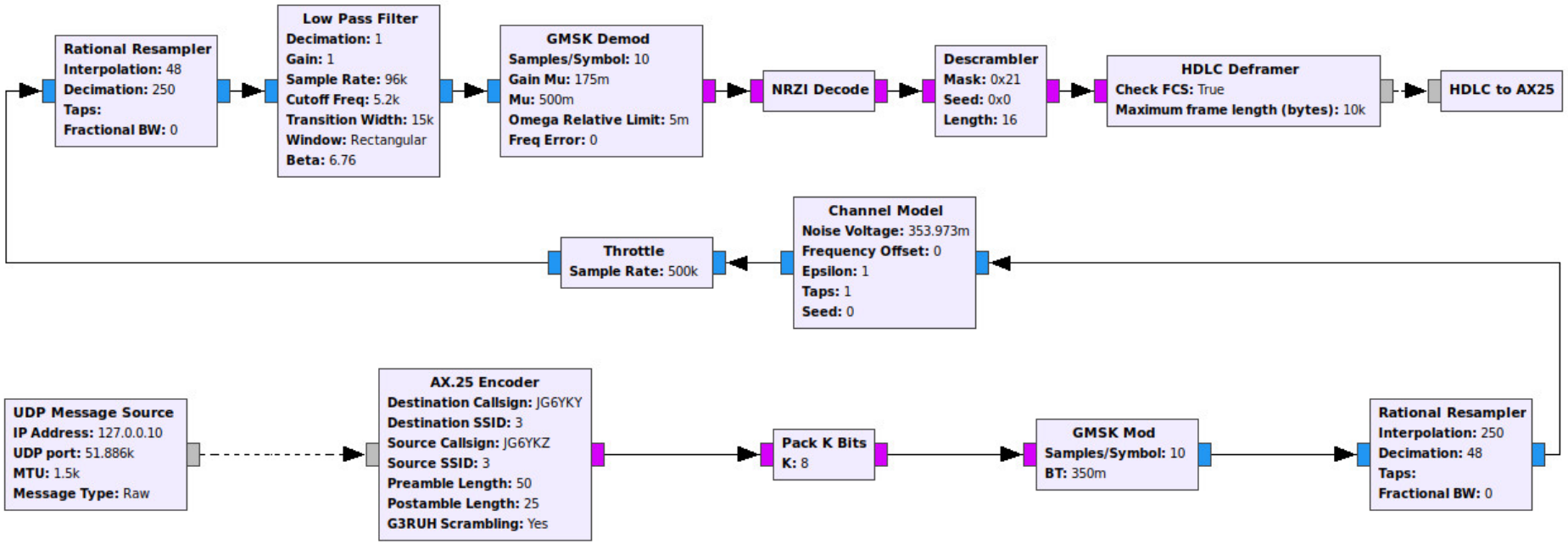


Figure 48 - GMSK SNR simulation diagram.

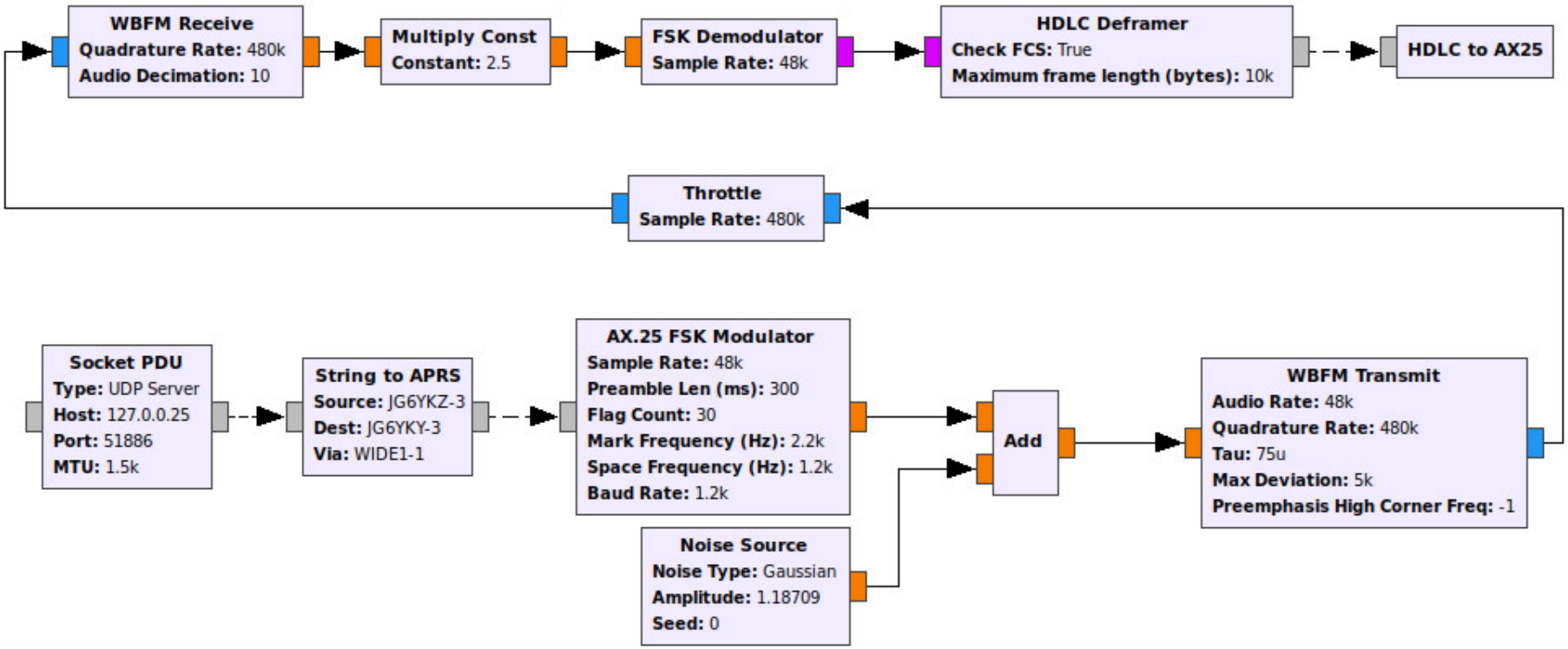


Figure 49 - AFSK SNR simulation diagram.

The expression to calculate the noise voltage amplitude parameter depending of the desirable SNR is given by the following formula:

$$V_{Noise} = \left(\sqrt{2 \cdot Bit_{Symbol} \cdot 10^{\left(\frac{SNR(dB)}{10}\right)}} \right)^{-1}$$

Equation 17 - Noise voltage calculation.

Were Bit_{Symbol} is the number of bits transmitted by one modulation symbol (1 for AFSK and 2 for GMSK).

In the plot is clear to observe how the constellation points are distorted and displaced as the SNR value is decreasing. In the case of the AFSK transceiver, a QT time sink block is connected at the output of the AFSK modulator block and after the addition of the Gaussian Noise to observe how the noise interferes with the modulated signal. In the Figure 52, it is possible to observe how the AFSK modulated signal is affected when the SNR value decreases. Additionally, in both systems, a Throttle block is used between the transmitter and receiver in order to establish the sample rate used by the LimeSuite RF blocks and to avoid consuming the entire CPU resources since there are not hardware included in the simulation.

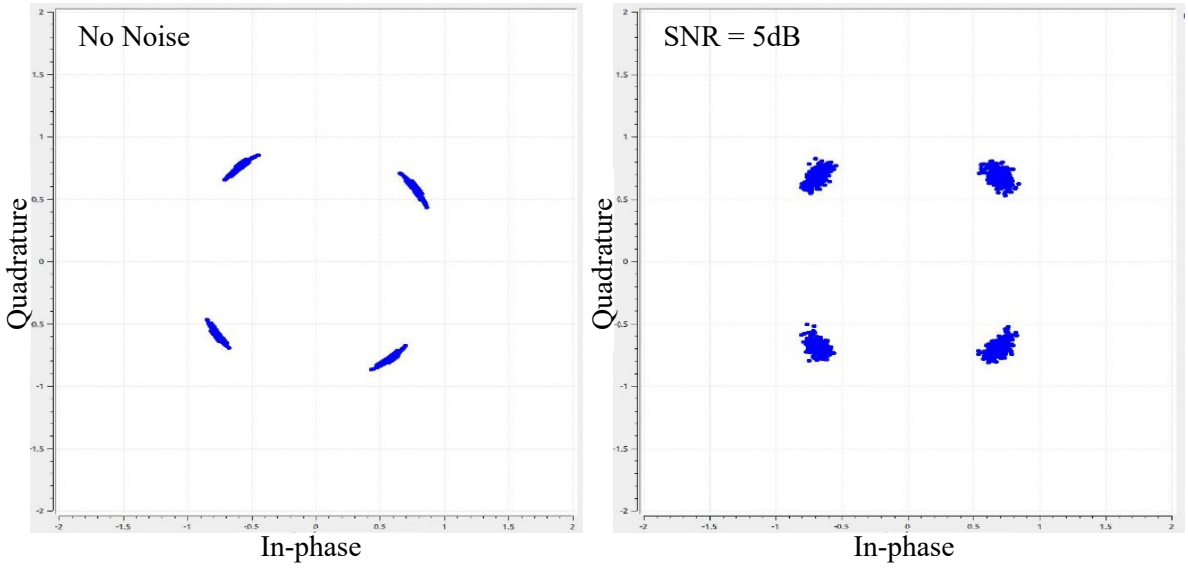


Figure 50. Noise effects in GSMK transceiver constellation diagram I.

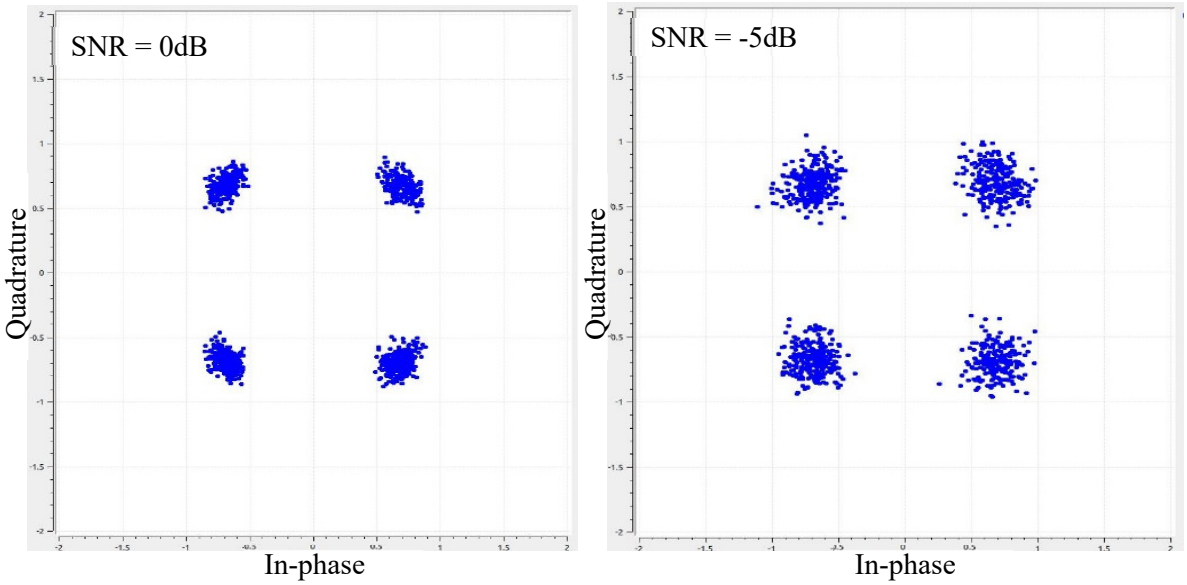


Figure 51. Noise effects in GSMK transceiver constellation diagram II.

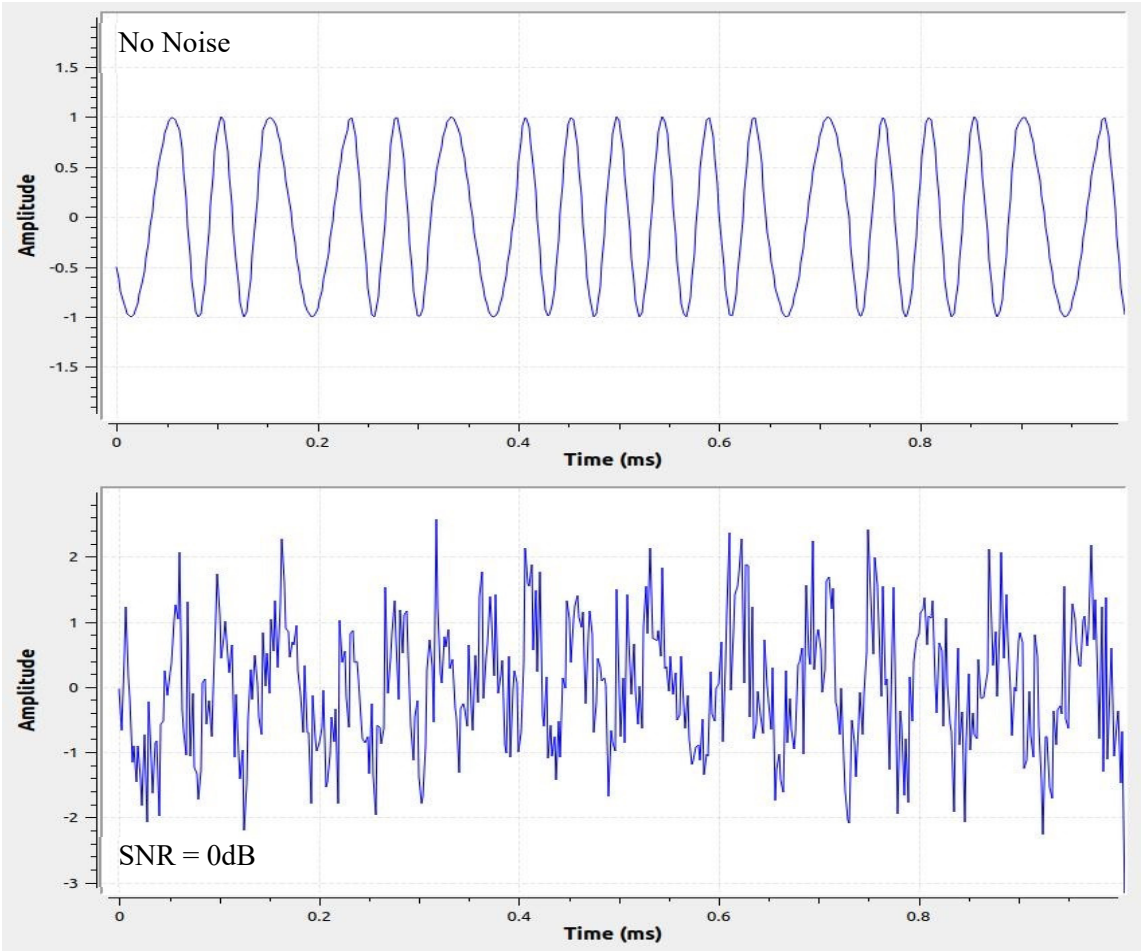


Figure 52 - Noise effects on AFSK modulated signal.

After verifying that the noise effects are included with the system correctly, it is possible to run the simulations to obtain the PER vs SNR performance of the system. In order to do that, a simulation in which 10,000 packets were sent from the transmitter to the receiver for every SNR variation step (from -5 to 4 dB in steps of 0.5 dB) were performed. For every step, the HDLC Deframer block in the receiver part decodes the AX.25 packets and decides which packets are correct or not. With this information, it is possible to calculate the PER using the Equation 15 used previously in the numeral 5.1. The time interval used to send the packets were 0.25ms for the GSMK transceiver and 1s for the AFSK transceiver. The results are shown in the Figure 53, Figure 54 and summarized in the Table 12. Here, it is possible to observe the SNR performance for the both transceivers are similar and it is possible to deduct that for values of SNR greater than 3.5 dB, the packet error rate is less than 10^{-4} (0.01%) which is a typical value used for ensuring correct communication in nano-satellite link budgets.

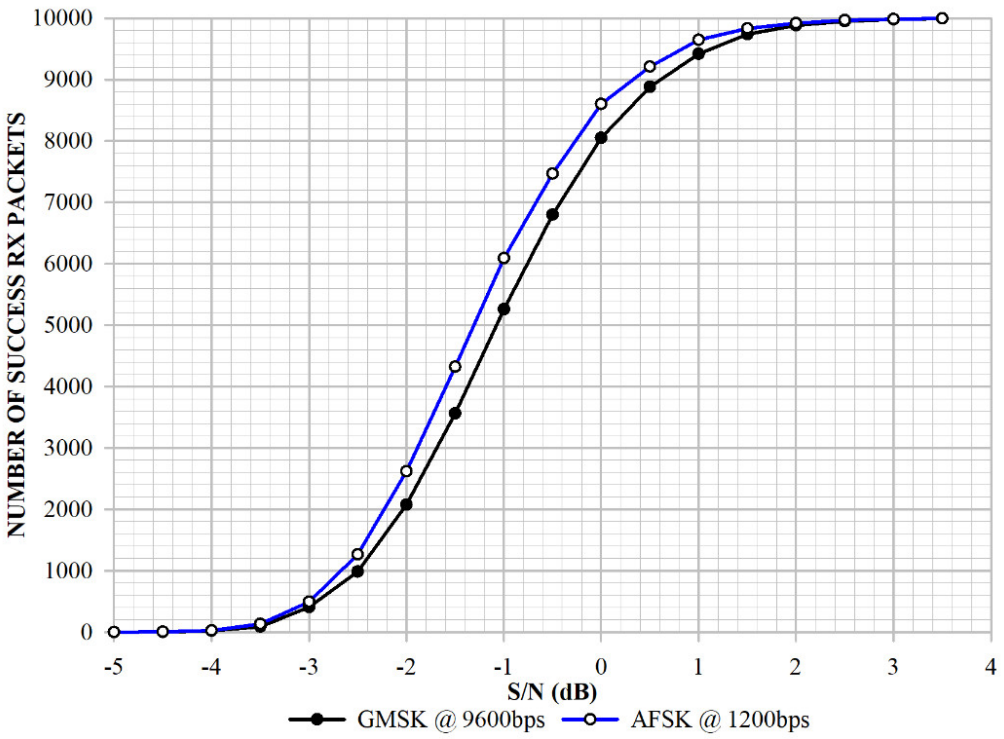


Figure 53 - Number of received packets in function of the Signal-to-noise ratio SNR.

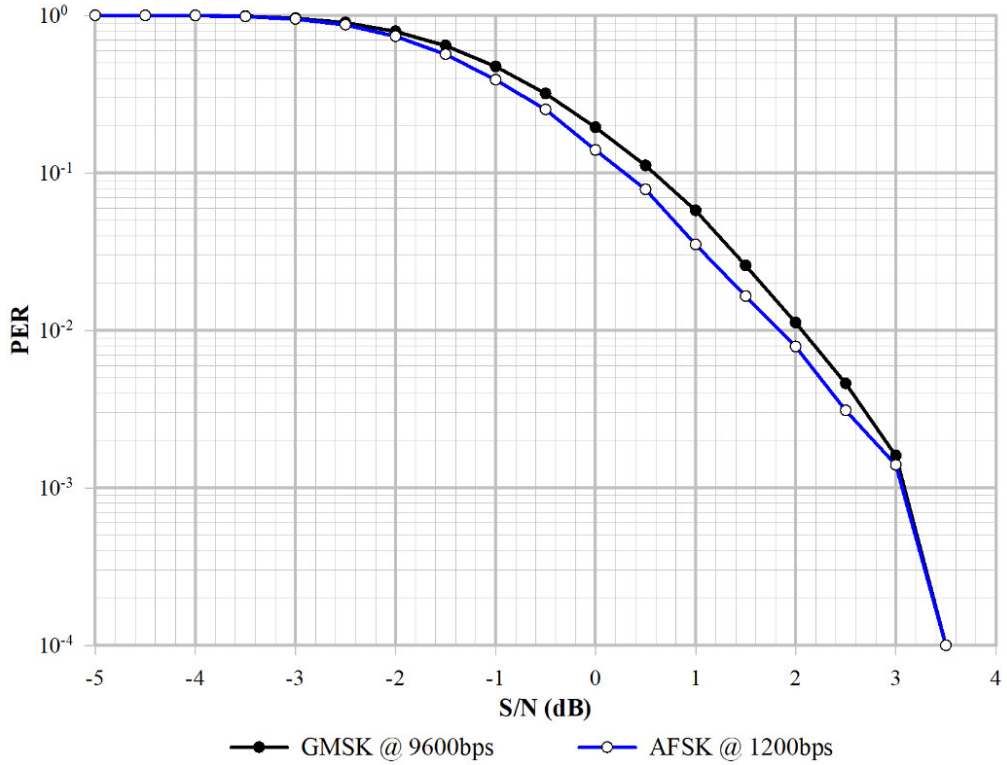


Figure 54 - Packet Error Rate (PER) in function of the Signal-to-noise ratio SNR.

Table 12 - Signal-to-noise ratio performance for AFSK and GMSK modulations.

PER (%)	AFSK SNR (dB)	GMSK SNR (dB)
0.01	3.50	3.50
0.10	3.10	3.10
1.00	1.90	2.10
10.00	0.20	0.60
50.00	-1.16	-1.05

5.3. Transmitter RF output power test

For the transmitter part, the maximum RF output power was measured using a spectrum analyzer. In the case of the Ten-Koh Nishimusen transmitter, it only operates in the UHF band, the power output is factory fixed and cannot be modified by software. In the case of the proposed SDR system, the maximum power output allowed by the LimeSuite Source (TX) block was configured (Gain dB = 60) and it was measured in the VHF, UHF and S bands since those are

typically assigned for satellite radio amateur communications. The test set-up is shown in the Figure 55, the transmitter (LimeSDR mini and Nishimusen) is connected to the spectrum analyzer through an attenuator in order to decrease the power level for protection. The power output level is measured when valid AX.25 packets are sent. The results of the test are shown in the Table 13.

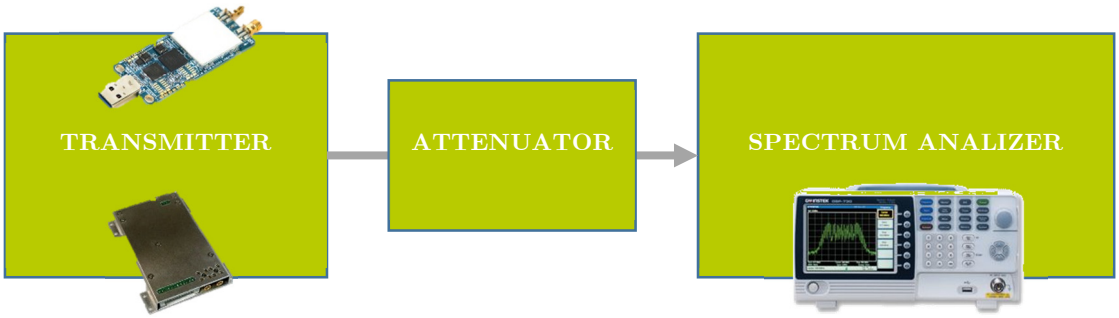


Figure 55 - Transmitter RF power output set-up

Table 13 - Maximum transmitter RF power output measurements for both systems transmitters.

Module	Power (dBm)	Power (mW)
Ten-Koh @ 437.385 MHz	29.0	794.00
SDR @ 437.385 MHz	2.0	1.58
SDR @ 145.980 MHz	3.5	2.23
SDR @ 2.4 GHz	-4.5	0.35

5.4. Power consumption test

In a nano-satellite mission, the power consumption is one of the biggest constraints in the bus subsystem design, for that reason is important to measure the power performance of the proposed SDR architecture for comparing it with the existing ones. In this test, the power consumption was measured for the Ten-Koh and proposed architecture monitoring the current when the systems were operating in different modes and at the same time the processor usage. In the case of the SDR, it was implemented in three different Raspberry Pi model variations. The results of the test are shown in the Table 14. Here, it is possible

to observe that the Ten-Koh system has the less power consumption both in the idle mode as in the transmission mode.

Regarding the SDR, the implementation in the Raspberry Pi Zero model has the less power consumption in all modes followed by the Pi 3B and the Pi 3B+ model respectively, which was expected since the Zero model includes a single core processor ARMv6 running at 1GHz and the 3B and 3B+ models include a quad core processor ARMv8 running at 1.2 and 1.4 GHz respectively. On the other hand, regarding the processor usage, it is possible to note the advantage to use a quad core processor in the 3B and 3B+ modules, running the corresponding python scripts, the maximum usage was just 25% in comparison with the 100% usage presented in the Pi Zero model running the applications in a single core.

Table 14 - Power consumption of Ten-Koh system and SDR implementations on different Raspberry Pi modules.

PARAMETTER	**R-PI ZERO W**		**R-PI3 B**		**R-PI3 B+**		**Ten-Koh**	
	I (mA)	**Power (W)**	**I (mA)**	**Power (W)**	**I (mA)**	**Power (W)**	**I (mA)**	**Power (W)**
IDDLE	110	0.550	280	1.400	490	2.450	30	0.150
100% CPU USAGE	195	0.975	770	3.850	685	3.425	--	--
IDDLE + LIMESDR	475	2.375	655	3.275	885	4.425	--	--
FSK TRANSMITTER	650	3.250	775	3.875	1050	5.250	--	--
GMSK TRANSMITTER	580	2.900	875	4.375	985	4.925	550	2.750
PROCESSOR USAGE	100%		25%		25%		--	

5.5. FPGA reprogramming system test

The purpose of this test is to verify the correct functionality of the reprogramming system implemented into the Raspberry Pi module for the FPGA logic and the SPI flash memory attached to the device and also to establish the required time to perform the reprogramming task for the Kintex 7 family as well as for the Spartan 6. As explained in the numeral 4.4.3, the JTAG controller uses

the GPIOs on the Raspberry Pi to create a TAP to be connected directly to the JTAG port into the FPGA device. The JTAG port consists in four connections defined as follow: TDI (Test Data In), TDO (Test Data Out), TCK (Test Clock) and TMS (Test Mode Select).

The connections between the Raspberry Pi and the FPGA JTAG port used in the test are shown in the Figure 56 and described in the Table 15. The GPIO port number into the Raspberry Pi can be chosen in a different way changing those into the configuration file presented in the appendix 2.

The test was performed using the Raspberry Pi Zero and the 3B+ models to define if there is a representative difference in performance between the both models. The configuration bit files used for performing the FPGA programming where the files resulting of every modulator implementation described in the numeral 4.4.2 (BPSK, FSK, QPSK, MSK and the integration of those modulators).

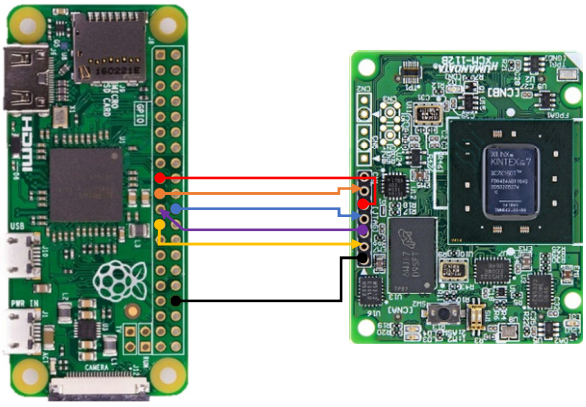


Figure 56 - JTAG Connections

Table 15 - JTAG Connections

Raspberry Pi		FPGA
Header Pin	GPIO Driver	JTAG Port
19	10	TDI
21	9	TDO
22	25	TMS
23	11	TCK
VCC	VCC	VCC
GND	GND	GND

The results of the test are shown in the Table 16 and Table 17, the required time for programming the different modulators into the SPI flash memory and the FPGA logic for the Spartan 6 and Kintex 7 families. It is possible to observe that the difference in time between the simplest modulator (BPSK) and the integration of all modulators has been just 3 seconds, then, for that reason for the Raspberry Pi Zero the test were performed only for the modulator implementations mentioned above. To program the Kintex 7 FPGA logic takes 10 seconds less than the Spartan 6 case.

Table 16 – R-Pi 3B+ FPGA system reprogramming time (sysfsgpio driver)

MODULATOR	SPARTAN 6		KINTEX 7	
	SPI MEMORY	FPGA LOGIC	SPI MEMORY	FPGA LOGIC
BPSK	02:04.7	00:25.3	01:15.2	00:15.5
FSK	02:04.7	00:25.5	01:15.8	00:15.3
QPSK	02:09.9	00:25.9	01:16.6	00:15.7
MSK	02:10.6	00:26.5	01:21.1	00:16.3
ALL MODULATORS	02:14.8	00:28.0	01:21.9	00:16.6

Table 17 – R-Pi Zero FPGA system reprogramming time (bmc2835gpio driver)

MODULATOR	SPARTAN 6		KINTEX 7	
	SPI MEMORY	FPGA LOGIC	SPI MEMORY	FPGA LOGIC
BPSK	02:05.3	00:32.7	01:16.5	00:19.3
ALL MODULATORS	02:14.8	00:36.0	01:22.0	00:21.2

For the SPI flash memories, the Kintex 7 takes 50 seconds less than the Spartan 6 case which shown that the Kintex architecture can be programmed faster using this reprogramming system implementation.

Comparing the results between the two Raspberry Pi modules, we can observe that the required reprogramming time in every case is almost the same which shows that the programming speed does not depend on the processor used in the Raspberry Pi module.

5.6. Discussion

As mentioned in the introduction section, this research intends to show an optimization of the Ten-Koh communication system, though an SDR implementation using a Raspberry Pi module in conjunction with Linux, Python and GNU Radio tools. The above sections showed the Ten-Koh system architecture, the SDR design procedures, testing and simulations performed for the both systems. In this section we will discuss and analyze the obtained results.

As shown in the chapter 4CHAPTER 4 -, the proposed SDR implementation offers the possibility to send and receive data in AFSK (1,200bps) and GMSK (9,600bps) modulations at a tunable frequency (VHF, UHF and S bands) by

software. The transmitter output power and the receiver sensitivity can be also modified by software and additionally, it is possible to implement other kind of modulation schemes using the available library blocks in GNU radio without the need to modify the hardware. Comparing the above with the characteristics of the Ten-Koh architecture mentioned in the chapter 3; clearly, it is possible to conclude that the proposed SDR architecture offers more flexibility and reusability which is very desirable for future nano-satellite missions.

Regarding the transmitter performance, about the maximum number of data bytes allowed per packet, is clear to observe that the proposed SDR system improves the value almost 3 times since it is possible to send the complete 255 bytes allowed by the AX.25 protocol in comparison with the 65 bytes allowed by the Ten-Koh transmitter due to the PIC microcontroller RAM memory limitations mentioned in the section 3.3. Regarding the maximum RF power available in the both systems, it is possible to observe that the proposed SDR has a disadvantage due to the value is 27 dBm lower than the Ten-Koh system for the same frequency band as showed in the table 2. However, it can be improved, including and RF power amplifier or choosing another RF module which can provide the required RF power output level.

In the case of the receiver performance, in the numeral 5.1, the packet error rate for different signal strengths were analyzed. Here, it is possible to observe that the Ten-Koh system has slightly better sensitivity performance (-115.5 dBm against -108.5 dBm for a packet error rate of 1%), however, the system offers a good performance comparing it with commercial/space heritage systems available in the market e.g. ISIS VHF/UHF Duplex Transceiver (-104 dBm) [47]. Also, the maximum amount of data allowed by received AX.25 packets was increased by almost 8 times, since the proposed SDR is able to receive up to 255 bytes while the Ten-Koh receiver can only receive up to 32 bytes transmitter due to the PIC microcontroller RAM memory limitations mentioned in the numeral 3.3.

Regarding the power consumption of the systems, the proposed system is more

power hungry as expected since we are making an upgrade in the processor side, we can observe that using a powerful Raspberry Pi module increases the processing performance while the power consumption also increases up to 2 times more comparing it with the Ten-Koh system. It is clear to observe that the implementation using the Zero model can be more convenient for nanosatellite missions since the power consumption is not significantly higher compared with the Ten-Koh system. However, having the possibility to use different Raspberry Pi models offers flexibility and it is possible to make a trade-off between the processing capabilities versus the power consumption depending the mission constraints without need to modify the implemented software modules.

We can observe how using Linux and GNU Radio Companion as development tool offer the possibility to simulate and monitoring several signals into the design to analyze the performance of the system without needing external equipment. A good example of that is the simulations preformed in the numeral 5.2 to estimate the SNR performance, adding the noise effects into the system and measuring the packet error rate. We obtained reasonable results in comparison with other simulation cases like shown in [48] or [49] for similar modulation schemes. Also, it is possible to make changes to the design only modifying the parameters into the signal processing blocks and observe the changes immediately, which is a big advantage to using an SDR platform over hardware fixed systems like Ten-Koh implementation.

Finally, the FPGA reprogramming system implemented into the Raspberry Pi was tested, we could verify that the time required to perform an FPGA programming is not high in comparison with the typical time required to program the device using a normal JTAG interface, then the system could be used in a nanosatellite mission in order to allow maintenance and error correction due to SEEs generated by radiation effects or if the system hangs unselectively.

CHAPTER 6 - RADIATION TEST

This chapter describes the procedures and details of a Single Event Effects (SEE) testing by using a beam of energetic protons in a synchrotron facility. The key definitions, test facility description, procedure steps, materials, methods and instrumentation needed for performing the test are included. Finally, the analysis of the obtained results is discussed.

6.1. Purpose of the test

The main objectives of the SEE testing are:

- To verify if the Ten-Koh on-flight failures discussed in the CHAPTER 2 - 2 (numeral 2.2) are produced by the radiation environment effects or perhaps by other causes.
- To verify the behavior and operation of the Raspberry Pi devices used in the proposed SDR platform described in the chapter 4 in a high-energy radiation environment such as the scenarios encountered in space. From the results of the tests, it will be possible to conclude if the Raspberry Pi could be selected as a processor candidate for future missions and in such a case, mitigation strategies can be deducted as well as operation margins.
- To estimate and compare the SEE sensitivity of the processor used in the Ten-Koh subsystem (PIC16F877) with on-ground testing versus in-orbit testing.

6.2. Test facility description: WERC Synchrotron accelerator

The facility used for this testing is the synchrotron located at the Wakasa Wan Energy Research Center in Fukui prefecture, Japan. The synchrotron can produce protons, helium and carbon ions with the characteristics presented in Table 18.

Table 18 - Synchrotron protons and ion characteristics [50].

Incident energy	H^+(protons): 10 MeV (B = 0.46 Tm) He^{2+}, C^{6+}: 2.08 MeV (B = 0.42 Tm)
Outgoing energy	H^+(protons): 10 to 200 MeV/u (B = 2.15 Tm) He^{2+}, C^{6+}: 2.08 to 55 MeV/u (B = 2.15 Tm)
Operation repetition frequency	0.5 Hz
Format	Function separation time.
Beam injection method	Multiple rotation injection method.
High frequency acceleration cavity	Multi-Feed Untuned Accelerating Cavity.
Beam extraction method	Resonance extraction method (diffuse resonance method) that applies high frequency with constant electromagnet strength.

The Figure 57 shows a diagram of the synchrotron facility used for the SEE testing. Here, the particles (ions or protons) are generated by the ion source and are injected into the Tandem accelerator which is in charge to accelerate the particles at an energy of around 5 MeV; then, the pre-accelerated particles can be provided in the irradiation rooms 1 and 2 or can be used to feed the synchrotron accelerator. The synchrotron is a ring that consist in 8 synchronized coils (red blocks) that are in charge of accelerate the particle beam at an energy of around 200 MeV. Afterwards, the accelerated beam is ready to be provided to the irradiation rooms 3 and 4 through the high energy transport system. In the case of this experiment, the irradiation room 4 was used in which all the test equipment for data acquisition set-up was configured. The synchrotron facilities are located underground for avoiding external interferences and for avoiding undesirable particle radiation to the external environment. For obvious security reasons, when the synchrotron is in operation, there should be no people inside of any irradiation room.

Figure 57 - Synchrotron accelerator facility diagram from [51].

6.2.1. Ion beam conditions

The objective of this test is to obtain the SEE cross-section for the PIC16F877 and for the Raspberry Pi module processor. As mentioned in the numeral 6.4.1, depending on the particles used to radiate the device. In the Figure 7, it is possible to see that the typical LET range to obtain the SEE cross-section using ions is from about $10\ to\ 100\ MeV \cdot cm^2/mg$. As shown in Table 18, the beam energy range that the synchrotron can supply is from $10\ to\ 100\ MeV$. By using helium and carbon ions, different levels of LET can be estimated in the selected energy range using the information provided by the NASA Space Radiation Laboratory shown in the Figure 58. The LET values are shown in the Table 19. From that values, it is possible to observe that the LET range available for this synchrotron is in the order from $0.014\ to\ 1.1\ MeV \cdot cm^2/mg$ which is not enough to obtain the desired SEE cross-section from the devices.

From the Figure 58, it is possible to observe that to be able to achieve the required LET using ions in silicon, heavy ions as Xenon (Xe), Tantalum (Ta), Gold (Au) and Thorium (Th) are needed, however, this type of heavy ions are not available in the WERC synchrotron.

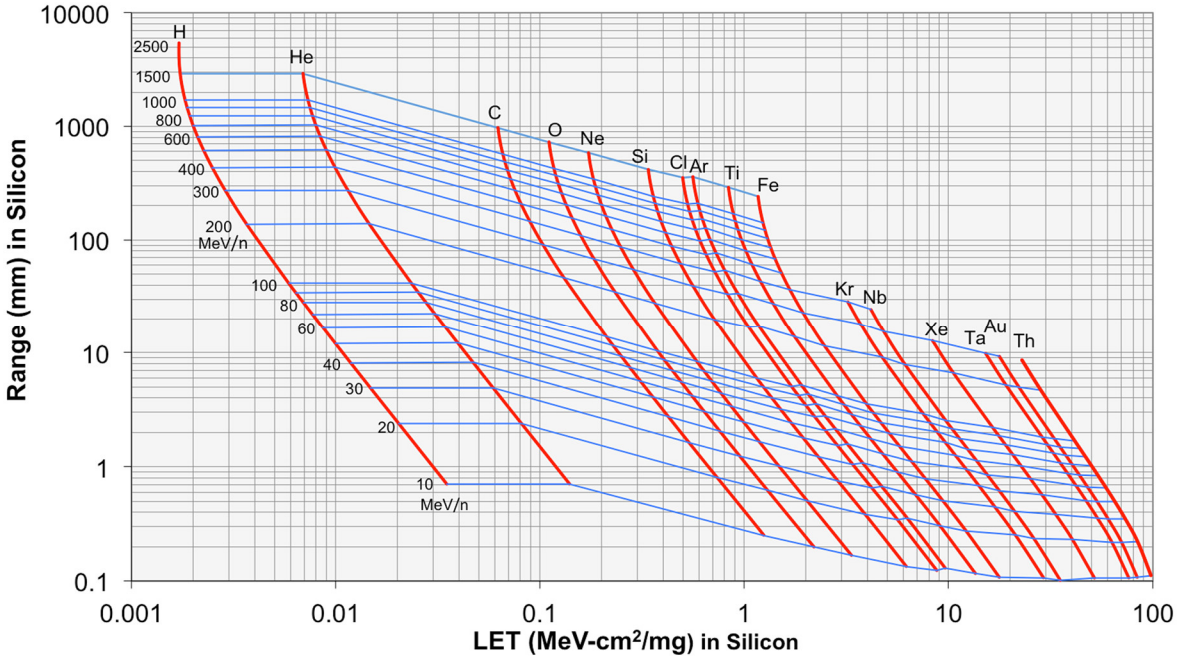


Figure 58 - LET in silicon for different ions [52].

Table 19 - LET in silicon for He and C ions for WERC synchrotron beam energy range.

Beam energy [MeV]	**He^{2+} LET in Si [MeV·cm²/mg]**	**C^{6+} LET in Si [MeV·cm²/mg]**
10	0.12	1.10
50	0.04	0.31
100	0.011	0.21
150	0.019	0.17
200	0.014	0.13

6.2.2. Proton beam conditions

Because the LET of the recoil nucleus is unknown when using protons for SEE,

the characterization of the device cross-section is performed with the primary incident proton beam energy as shown in the Figure 8. According to [53], the delivered energy by the accelerator shall be in the range from 20 to 200 MeV with a variable flux ranging from 10^5 to $10^8\ proton/cm^2/s$ on the device under test.

It is also important for considering the total ionizing dose (TID) damage. Some devices can show an increase in SEU susceptibility of up to two orders of magnitude as a result of TID effects from proton irradiation [2]. In this regard, the TID received by the device shall be accounted by adjusting the irradiation time of each device to the lowest possible that allows the SEE cross-section characterization. The total dose received by a device from a high-energy proton beam can be estimated from Equation 18:

$$D[Gy] = \Phi \cdot k(1 \times 10^{-3})(1/\rho)dE/dx$$

Equation 18 - Total dose received from a proton beam.

Where D is the dose in gray (Gy) units, Φ is the proton fluence in $protons/cm^2$, k is a constant to convert MeV into J (joule) with a value of $1.602\ \times 10^{-13}$, ρ is the target material density in $g/cm3$, dE/dx is the deposited energy in the material of thickness in MeV/cm. The value of 1×10^{-3} converts g into kg. D can be converted from Gy to rad by multiplying the result of Equation 18 by a factor of 100.

The deposited energy can be replaced by the energy loss per unit path length, also known as the stopping power, which is approximately true if the energy along the particle path remains constant (i.e. long-range protons).

The proton beam provided for the WERC synchrotron consists of protons (H+) with the following characteristics:

1. The beam energy starts at the level of 20 MeV, the following is desired to be set at 50 MeV and then increasing in steps of 50 MeV until getting the

maximum beam energy of 200 MeV (20, 50, 100, 150, 200).

2. The flux of protons is in the order of $10^5\ to\ 10^8\ proton/cm^2/s$ so enough protons produced recoil atoms. The exact flux has to be confirmed during the test at the facility. The upper value $(10^8\ proton/cm^2/s)$ is preferable, however, for very sensitive devices, the flux can be adjusted to the lower value $(10^5\ proton/cm^2/s)$.

In base of the discussed above and in the numeral 6.2.1, the proton beam conditions are the most suitable to perform the test using the WERC synchrotron and where the conditions chosen for testing the devices.

6.3. Device Under Test (DUT) preparation

The devices chosen for the radiation test are the following:

- PIC16F877 which was used for all the subsystems included on-board of Ten-Koh satellite.
- Raspberry Pi Zero and Raspberry Pi 3B+ used in the proposed SDR architecture.

6.3.1. PIC16F877

As discussed in the CHAPTER 2 - 2, some failures were present on-board during the Ten-Koh operation phase. It included several reset events in the OBC system and several failures into the EEPROM/RAM memories. For that reason, the microcontroller was programmed with a specific software which permits monitoring the mentioned parameters during the irradiation time. The flow chart that describes the software implemented for the microcontroller is shown in the Figure 59. The idea is to configure the entire EEPROM and RAM memories with a known default value; afterwards, the program enters in a loop that read and send the entire memory data via UART protocol every second in order to detect if the values are modified due to the proton irradiation beam.

Additionally, for the EEPROM memory, some specific positions are written with another known value (15 bytes in specific) in every loop iteration in order to check if the radiation beam affects the writing operation. It is not necessary for the RAM memory to perform a specific writing operation because of the data UART transmission, a buffer (57 bytes) is used which is constantly written whit the values read from the memory positions.

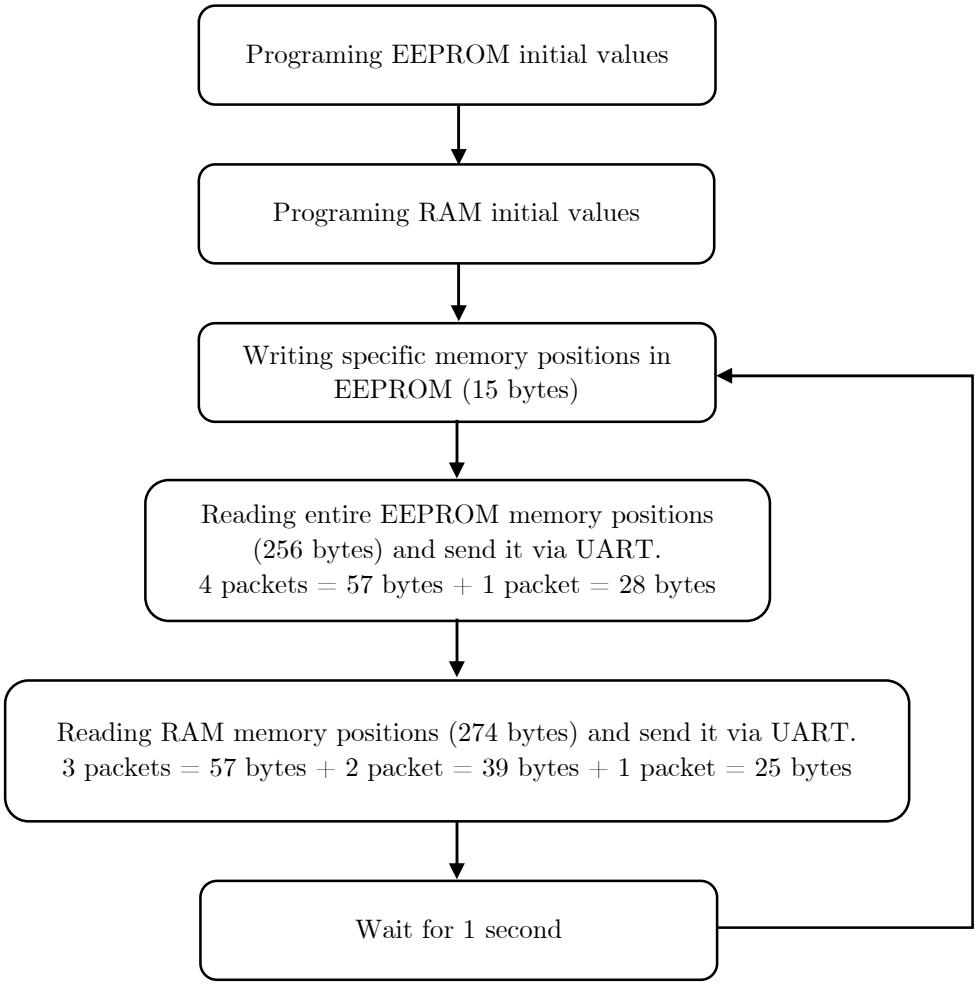


Figure 59 - PIC16F877 radiation test software flow chart.

The format of the data received from the DUT via the UART protocol is divided in two parts, the EEPROM memory data and the RAM memory data.

The EEPROM memory data format is shown in the Figure 60. It consists of 4 packets of 57 bytes and 1 packet of 28 bytes to complete a total of 256 bytes which is the entire size available on the device. For every packet, the first three

bytes “F” are the packet header values that are written in every loop iteration, the next 53 bytes are the default memory values “A” which are the values programmed permanently in the EEPROM memory and the software does not modify it in the entire execution and the last byte “X” which is the packet ending value and acts as same as the previous default values.

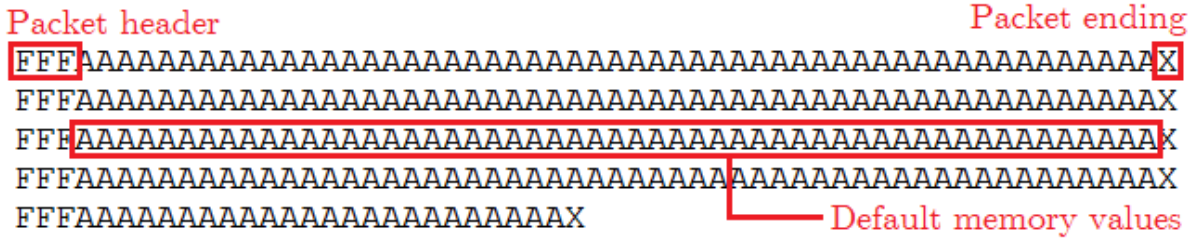


Figure 60 - EEPROM memory data format.

The RAM memory data format is shown in the Figure 61. The RAM memory in divided in 3 regions identified by the default values “1”, “2” and “3” respectively. The regions 1 and 2 are divided in 1 packet of 57 bytes plus 1 packet of 39 bytes and the region 3 is divided of 1 packet of 57 bytes plus 1 packet of 25 bytes. The other identifiers into the packet work as same as mentioned in the EEPROM packets with the difference that no values are written during any loop iteration. The total amount of RAM data received via UART is 274 bytes.

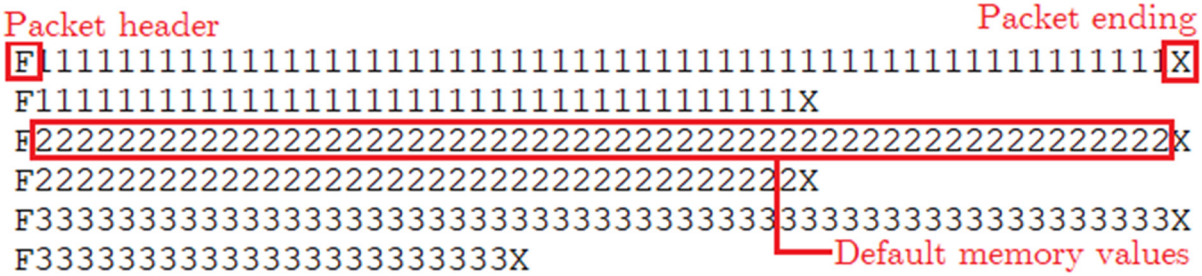


Figure 61 - RAM memory data format.

As mentioned before, in the case of the RAM memory, it is not necessary to write data in every loop iteration because the UART transmission buffer is part of the RAM memory and it is written automatically in every UART data

transaction; then, if the data corruption due to the radiation effect is in the buffer memory region, the data received will be corrupted temporary and will be recovered in the next loop iteration. On the other hand, if the data corruption is in the other memory positions, the data received will be corrupted permanently due to the other positions on the RAM are not modified in any part of the loop iteration.

6.3.2. Raspberry Pi 3B+ and Zero

As discussed in the previous chapters, the Raspberry Pi seems to be a good candidate to be used in the design of subsystems for nano-satellite applications. However, this single-board computer still has not enough space-heritage to become a widely used device for space applications as discussed in the CHAPTER 1 - 1.

One of the most important characteristics to define if the device can be used or not safely in space applications is its performance in the presence of radiation environment. For that reason, it is very useful to be able to perform a radiation test in the ground in order to characterize the radiation performance of the Raspberry Pi; however, there are not too much information regarding it. The most relevant test found in the literature is the case of the radiation experiment performed by the University of Surrey [54] in which a Raspberry Pi Compute Module 3 was tested for TID conditions, the result of the test is the evidence that the module can work without failure under beta ray irradiation of up to 130 krad.

On the other hand, there are not so much information about the radiation performance of the Raspberry PI in satellite missions. The most relevant is Astro Pi, a mission which allows to run a code aboard the International Space Station (ISS) [55]. They included a CCD sensor in order to measure random ionizing radiation events and also a reset counter to detect if the module experiments unexpected resets due to ionizing radiation; however, the initial conclusion was that the radiation sensor experiment was not successful and no unexpected resets

were counted because the experiment was inside of an aluminum case and probably the thickness keep out much the radiation levels.

Due to the Raspberry Pi is a module that runs an operative system which runs over an SD card memory (used as a FLASH/ROM memory) and includes an external dedicated RAM memory, it will not be possible to detect errors at memory/byte level. In this case, the idea is to monitoring the correct functionality of the operating system at kernel level using the UART interface activating the kernel serial log functionality. At his way, the operating system will notify any error presented at the system/kernel level (e.g. CPU execution error, memory error, etc.) and error/device events can be monitored instead of error/byte events monitored in the PIC microcontroller case.

The preparation and prerequisites of the Raspberry Pi in this case are the same as mentioned in the CHAPTER 3 - and additionally a python script executing an algorithm following the flow chart shown in the Figure 59 as for the PIC test, reading and transmitting memory data in the same format as explained in the Figure 61 via UART interface. In this case, it is not necessary to write/read EEPROM memory because the SD card is treated as a ROM memory and it will not be radiated in the test.

6.4. Radiation test set-up

Before starting the test, the synchrotron proton beam has to be calibrated in order to measure the precise energy and fluence values for every desired energy step as well as the radiation field dimensions and characteristics. After, the DUT has to be located in the irradiation room, correctly aligned to the beam according the radiation field characteristics obtained during the calibration process and finally, connect the DUT to the control and data acquisition system providing the controlled power supply line and data interfaces.

In the following numerals, the process to perform the beam calibration, the DTU alignment and the test set-up conditions are presented.

6.4.1. Proton beam calibration

The proton bean always works at a desired maximum energy and nominal fluence values; however, a different energy values are needed for this test. The way to achieve that is locating a degrader material between the beam output and the DUT as shown in the Figure 62. The material used is called "Solid Water" which is a solid block that attenuates electrons and protons same as the normal water. Different lower beam energies can be obtained increasing the thickness of the solid water blocks, then, for every energy step that need to be attenuated, a different thickness of a solid water material should be located between the DUT and the proton beam output.

Before the test operation, the proton beam must be calibrated for every desired energy step level. The calibration procedure was performed with the support of the WERC staff and it includes the following steps:

- The WERC staff set-up the maximum bean energy level and the nominal fluence.
- In the DUT position, a dummy target is aligned to the beam at the same desired distance from the beam output.
- The beam is activated in order to measure the following parameters:
  - The beam energy (in MeV).
  - The degrader thickness if it is present (in mm).
  - The beam flux (in $proton/cm^2/s$).
  - The beam radiation field (beam spot size in cm).
  - The beam efficiency.

To reduce the energy to the next step, it is necessary to enter into the irradiation room and locate the adequate solid water material thickness between

the dummy DUT and the output of the proton beam as shown in the Figure 62. After that, the same procedure should be performed for every energy level step (150, 100, 50 and 20 MeV) obtaining the corresponding beam characteristics.

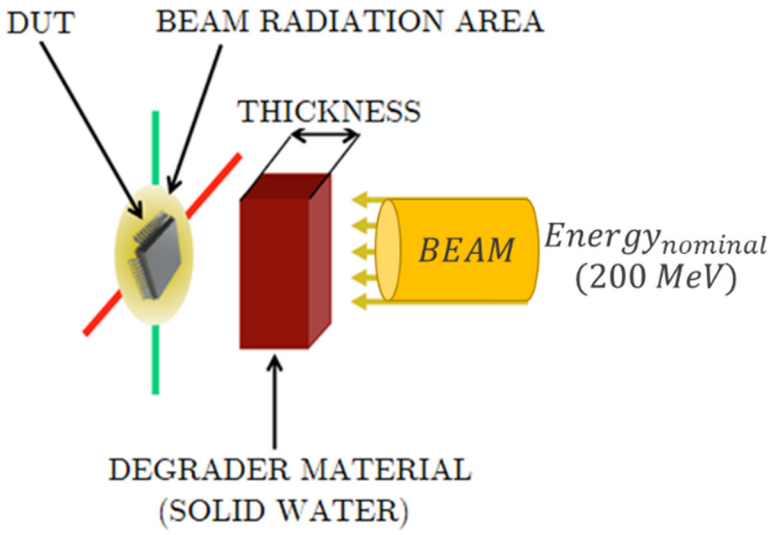


Figure 62 - Proton beam attenuation process.

6.4.2. Alignment of the DUT in the proton beam

The spot or irradiation field of the beam was provided by the facility team after performing the calibration procedure providing the exact dimensions of the radiation field; then, the DUT can be aligned before starting the tests. It must to be located orthogonal and perfectly centered in the beam as shown in the Figure 64. In order to do that accurately, a laser source provides the guide for the vertical and horizontal axis and indicates the exact center point of the target where the beam will be irradiated. The final alignment for the PIC microcontroller and for the Raspberry Pi Zero is shown in the Figure 63.

6.4.3. Facility set-up

As shown in the Wakasa Wan Research Center facility description in the numeral 6.2. The test was performed in the irradiation room 4 which is located underground where there is no possibility to control the acquisition equipment

inside the room and it was necessary to control everything remotely. The irradiation room final set-up is shown in the Figure 65 in which the DUT is located in front of the proton beam and aligned using the laser source guides.

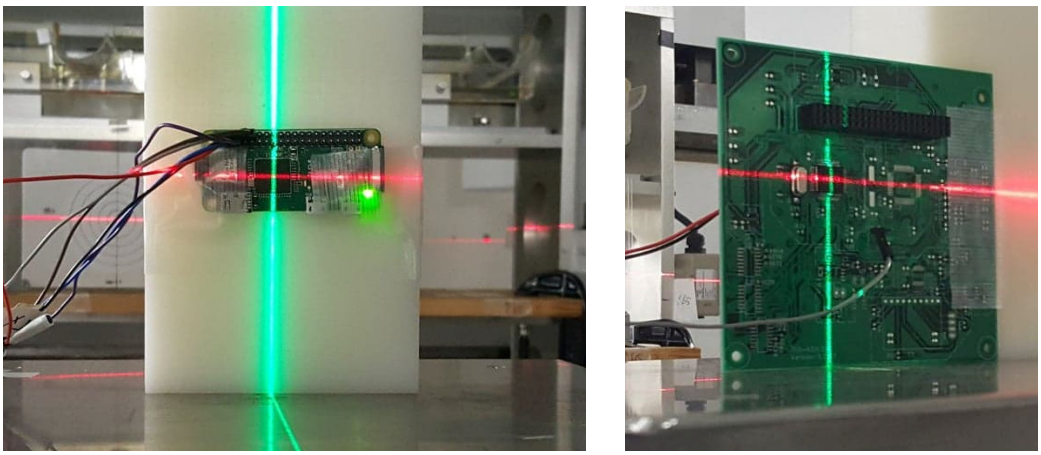


Figure 63 - Final beam alignment for Raspberry Pi Zero and PIC microcontroller.

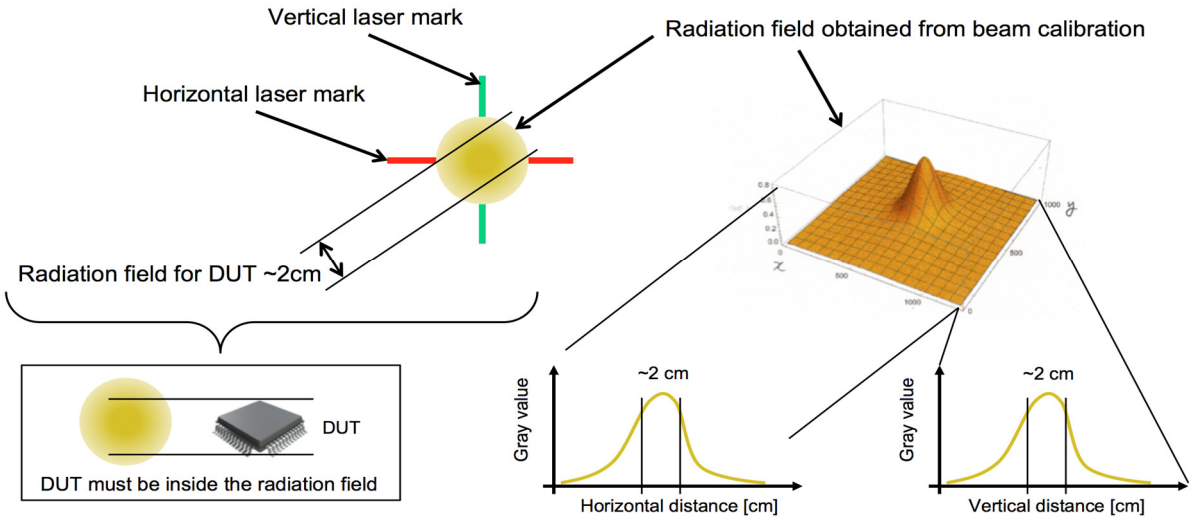


Figure 64 - Sketch of the DUT alignment with the proton beam.

The Figure 66 shows the experiment diagram at radiation facility. In the control room, the equipment to control the irradiation beam is located which is completely controlled by the WERC staff. Also, a computer executing a remote desktop software is used to control remotely the data acquisition desktop located in the irradiation room via the Local Access Network (LAN) available in the

radiation facilities.

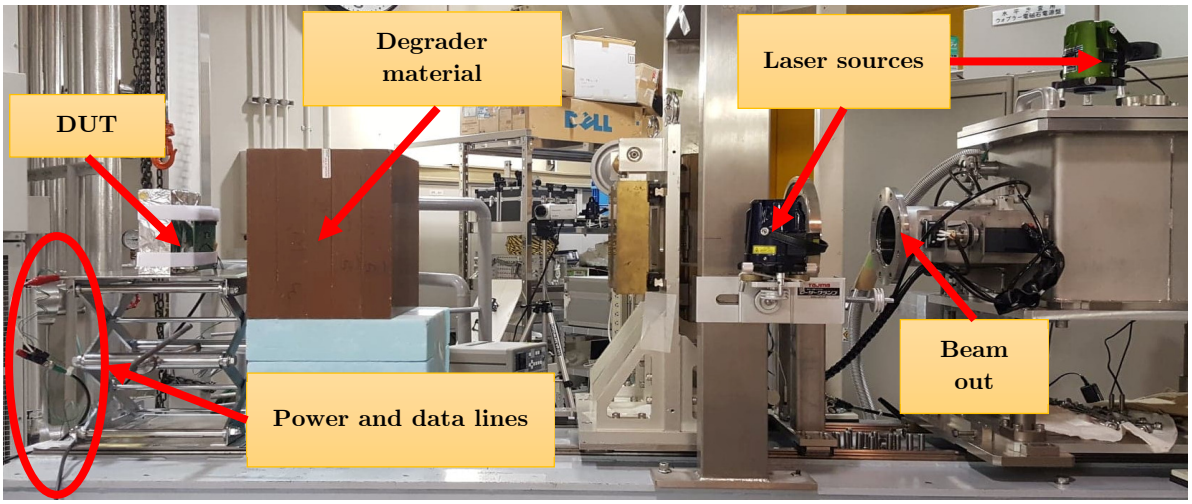


Figure 65 - Synchrotron irradiation room DUT set up.

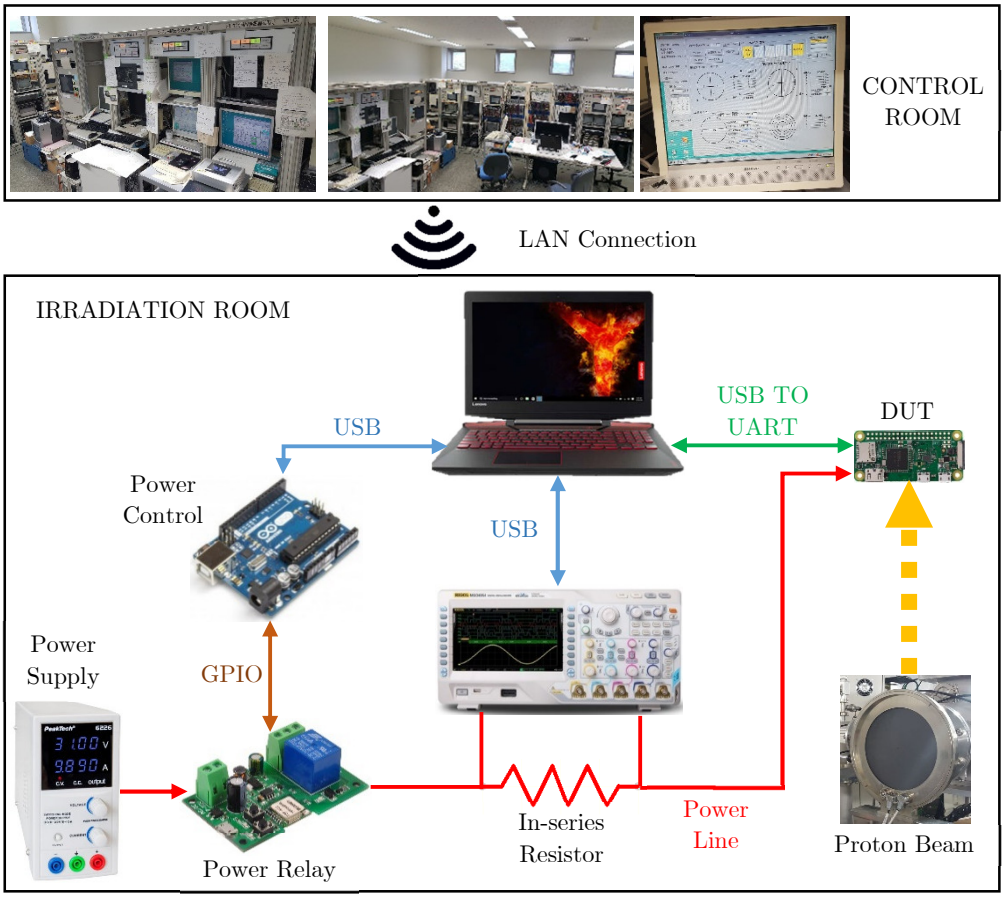


Figure 66 - Radiation facility set-up

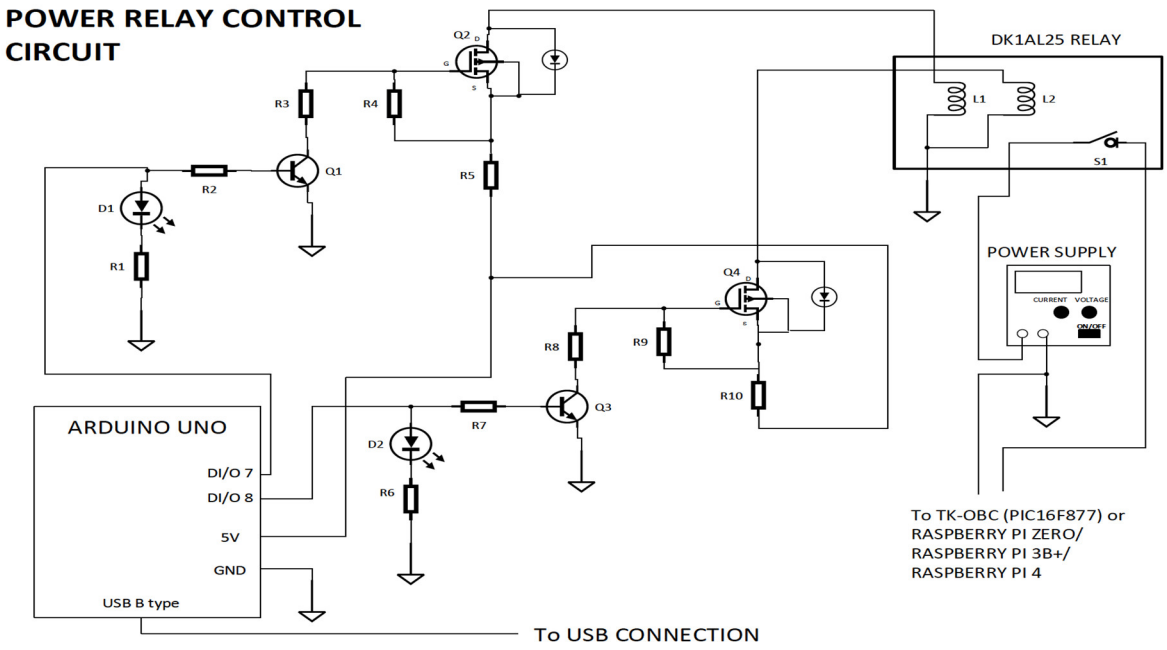


Figure 67 - Arduino-based power control relay circuit schematic.

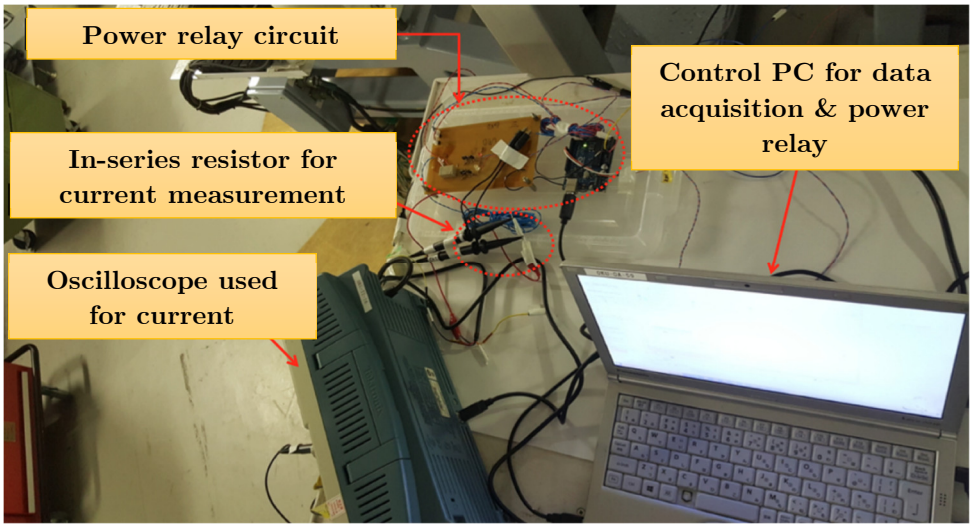


Figure 68 – Radiation test data acquisition set up.

In the irradiation room, the DUT is connected to the desktop via USB to UART converter transmitting the test data continuously every second. The power line to the DUT is controlled by the control laptop using an Arduino board which is in charge of controlling a power relay board in order to connect or disconnect the power supply remotely, the Arduino board is controlled by the control laptop via USB interface. The complete power relay control circuit is shown in the Figure 67. The current on the DUT is monitored via an in-series

resistor and one oscilloscope which is connected to the control laptop via USB interface, the control desktop executes a proprietary acquisition software in charge of receiving and store the current consumption. The data acquisition set-up is shown in the Figure 68.

6.5. DUT irradiation procedure

Once the proton beam has been calibrated, the DUT has been aligned correctly and the control & data acquisition system set-up is done, then everything is ready to start the test. The obtained calibration values provided from the WERC staff and used to conduct the test are summarized in the Table 20.

Table 20 - Proton beam calibration values.

Beam Energy (MeV)	**Beam Flux ($proton/cm^2/s$)**	**Degrader Thickness (mm)**	**Beam Spot Size (mm)**	**Beam Efficiency**
200	10^8	0	20 x 20	0.0946
150	10^8	96	20 x 20	0.0946
100	10^8	172	20 x 20	0.0946
50	10^8	224	20 x 20	0.0946
20	10^8	240	20 x 20	0.0946

The test process steps followed for all the DUT were:

1. Inside the irradiation room, turn on the DUT board and data acquisition system, perform a device initialization and functional test to confirm all systems are operating as expected.
2. From the control room, turn on the DUT power supply and initialize the software that control and store the current consumption data verifying the nominal values and correct functionality.
3. Start to store the data received via UART interface verifying the correct functionality.
4. Start the beam irradiation with the desired fluence, the synchrotron system will calculate the irradiation time and will automatically stop the beam

when the time is elapsed.

5. During the beam irradiation time, collect the data and make annotations about relevant and unexpected events and the beam irradiation time.
6. If necessary, change the desire fluence and start again the process from the step 2.
7. When finish the test, inform to the WERC staff in order to turn off completely the beam and to be able to enter into the irradiation room to include the degrader material to attenuate to the next energy level.
8. Repeat the steps from 1 to 7 for every energy level.
9. Repeat the procedure for every DUT.

6.6. Test results

6.6.1. PIC16F877 Results

The PIC microcontroller was subjected to two irradiation time conditions, in the first experiment the device was radiated during 30 seconds and in the second one it was radiated during 60 seconds.

The irradiation parameters for the first experiment are presented in the Table 21 and for the second experiment are presented in the Table 23. As discussed previously, the proton beam energy is varying (1^{st} column) whit a constant flux (2^{nd} column). The desired fluence (4^{th} column) is calculated multiplying the flux by the desired irradiation time (3^{rd} column).

In order to set the beam at that value, it is necessary to preset the synchrotron with the fluence value expressed in dose unit values (5^{th} column) which were obtained previously as a result of the calibration process. The preset value configured in the synchrotron (6^{th} column) is calculated dividing the fluence value by the dose unit value.

During the irradiating time, a real time measurement of the fluence (7^{th} column) expressed in dose units is provided by the synchrotron and is the value used to calculate the real fluence irradiated to the device (8^{th} column).

Table 21 - PIC16F877 proton irradiation parameters (1st experiment).

Beam energy [MeV]	**Flux [$proton/cm^2/s$]**	**Irradiation time preset [s]**	**Desired Fluence [$proton/cm^2$]**	**Dose (1-unit value)**	**Dose Preset [Gy]**	**Measured Dose [Gy]**	**Real applied fluence [$proton/cm^2$]**
20	1.00E+08	30	3.00E+09	1.62E+04	185185.19	185350.00	3.00E+09
50	1.00E+08	30	3.00E+09	1.96E+04	153061.22	153584.00	3.01E+09
100	1.00E+08	30	3.00E+09	2.38E+04	126050.42	126830.00	3.02E+09
150	1.00E+08	30	3.00E+09	2.74E+04	109489.05	109987.00	3.01E+09
200	1.00E+08	30	3.00E+09	4.25E+04	70588.24	70691.00	3.00E+09

Table 22 - PIC16F877 cross-section calculation (1st experiment).

Beam energy [MeV]	**No errors in EEPROM bytes**	**No errors in FLASH/RAM bytes**	**Cross-section of (EEPROM + FLASH/RAM [$cm^2/byte$]**	**Cross-section of EEPROM [$cm^2/byte$]**	**Cross-section of FLASH/RAM [$cm^2/byte$]**
20	0	0	0.00E+00	0.00E+00	0.00E+00
50	0	1	3.32E-10	0.00E+00	3.32E-10
100	1	0	3.31E-10	3.31E-10	0.00E+00
150	0	5	1.66E-09	0.00E+00	1.66E-09
200	0	7	2.33E-09	0.00E+00	2.33E-09

Table 23 - PIC16F877 proton irradiation parameters (2nd experiment).

Beam energy [MeV]	**Flux [$proton/cm^2/s$]**	**Dose Monitor (1-unit value)**	**Irradiation time preset [s]**	**Desired Fluence [$proton/cm^2/s$]**	**Dose Monitor Preset [Gy]**	**Measured Dose [Gy]**	**Real applied fluence [$proton/cm^2/s$]**
20	1.00E+08	1.62E+04	60	6.00E+09	370370.37	370908.00	6.01E+09
50	1.00E+08	1.96E+04	60	6.00E+09	306122.45	306787.00	6.01E+09
100	1.00E+08	2.38E+04	60	6.00E+09	252100.84	252793.00	6.02E+09
150	1.00E+08	2.74E+04	60	6.00E+09	218978.10	219321.00	6.01E+09
200	1.00E+08	4.25E+04	60	6.00E+09	141176.47	141176.00	6.00E+09

Table 24 - PIC16F877 cross-section calculation (2nd experiment).

Beam energy [MeV]	**No errors in EEPROM bytes**	**No errors in FLASH/RAM bytes**	**Cross-section of (EEPROM + FLASH/RAM [$cm^2/byte$]**	**Cross-section of EEPROM [$cm^2/byte$]**	**Cross-section of FLASH/RAM [$cm^2/byte$]**
20	0	1	1.66E-10	0.00E+00	1.66E-10
50	2	1	4.99E-10	3.33E-10	1.66E-10
100	0	9	1.50E-09	0.00E+00	1.50E-09
150	25	12	6.16E-09	4.16E-09	2.00E-09
200	0	13	2.17E-09	0.00E+00	2.17E-09

It is possible to observe that the measured values are close to the calculated ones which indicate that the beam is giving the desired fluence correctly. The results of the both experiments are presented in the Table 22 and Table 24.

The number of byte error events presented in the EEPROM and in the FLASH/RAM memory of the microcontroller are counted in the 2nd and 3rd column respectively. With those values, it is possible to calculate the SEE cross-section values using the Equation 6 presented in the numeral 1.6.4.

As the calculation of the SEE cross-section is a result from nuclear reactions caused by protons as in this case, the experimental values obtained in the test can be fitted in order to obtain more accurate results and to be able to estimate the threshold kinetic energy E_{th} and the saturation cross-section σ_{sat} values as presented in the Figure 8 in the numeral 1.6.4. As mentioned in the Space Environment Information System (SPENVIS) help on single events upsets section [56], the experimental values can be fitted using a 2-parameter function or a 4-parameter Weibull function. The former is a good approximation when a few experimental data are obtained and when there are a few close to the threshold kinetic energy and the latter provides a better approximation when there are several experimental data, especially near to the threshold kinetic energy point, however it cannot be used when the experimental data available is obtained from four or less beam energy levels [57].

For performing the data fit, the OMERE 5.3.2 radiation software developed by Test & Radiations (TRAD) [58] was used. On its SEE rate estimation module, it is possible to introduce the proton cross-section experimental data and the software calculates the parameters of the Bendel or the Weibull function that best fit to the given data.

In the case of the RAM memory results, errors occurred both in the writing and reading operations. In the Figure 69, the SEE cross-section experimental values and the fitted curves using 2-parameters Bendel and 4-parameters Weibull functions for the FLASH/RAM memory are shown for the second experiment.

Here, it is possible to see how the probability of errors is growing as the proton kinetic energy also grows as expected. Comparing the data fit curves obtained by Bendel and Weibull functions, we can observe that both fit well the data; however, the Weibull fit assumes that the threshold kinetic energy point is the minimum beam energy level used which may not be correct because we got errors at that energy point. In the case of the Bendel fit, it is possible to see how the function estimates the threshold kinetic energy point which seems to be more adjusted to the expected behavior of the cross-section curve.

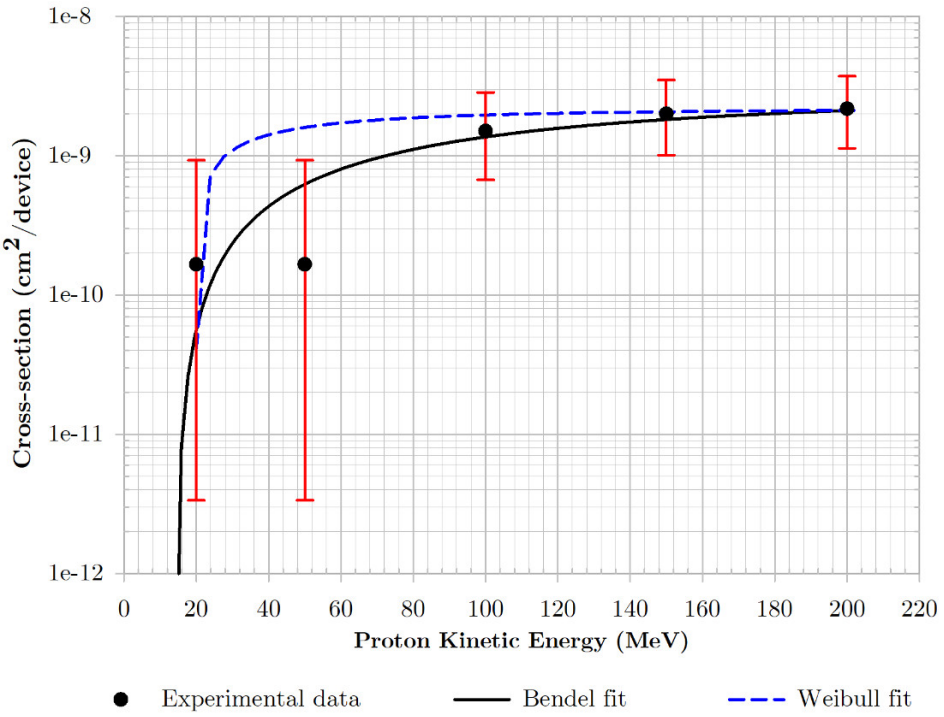


Figure 69 - PIC16F877 FLASH/RAM cross-section.

Finally, in accordance with the recommendations on the numeral 1.6 of the "Single Event Test Method And Guidelines" European Space Components Coordination (ESCC) basic specification No 25100 [53], when a few experimental cross-section data is obtained, error bars must be plotted with the experimental data for showing the uncertainly of the cross-section values. Following the procedure showed in the guideline, we calculated the uncertainly of our experimental data and the error bars are shown in red color on the Figure 69.

The parameters of the Bendel function obtained in OMERE 5.2 are $A = 14.00\,MeV$, $B = 24.72\,MeV$ and the cross-section representative values are:

- Proton Kinetic Energy Threshold $E_{th} = 14.00\,Mev$.
- Saturation cross-section $\sigma_{sat} = 2.83 \times 10^{-9} cm^2/device$.

On the other hand, in the case of the EEPROM memory, errors only occurred in the memory positions where the data were written in every loop iteration. In the first experiment, errors only occurred at 100 MeV proton kinetic energy which proves that the EEPROM memory is more radiation tolerant than the RAM memory. However, in the second experiment when the beam energy was 150 MeV, the mentioned memory positions were written with an unknown value. After that event, the programmed software were not able to write the correct value "F" in those memory positions anymore, which could mean that those memory positions could be damaged permanently. However, we tested the DUT after the radiation experiment and we found that the EEPROM values were changed to unknown value "F" due to the program memory was corrupted. We tested that connecting a PICKit programmer in order to verify the integrity of the FLASH program memory values comparing the values that was read from the DUT and the values of the programing file used to program the device before the radiation test.

Afterwards, we proceeded to reprogramming the device in order to verify if the EEPROM memory or the FLASH memory were completely damaged by the proton beam; however, the device were successfully reprogrammed and the application worked normally as expected which indicates that the device can be recovered of the failures generated by SEEs reprogramming completely the device.

In the Figure 70, the SEE cross-section for the EEPROM memory is presented. For the second experiment, errors were only detected at 50 and 150 MeV beam energies; then, for that reason, the data can be fitted only using the Bendel distribution because the 4-parameter Weibull fit requires at least the cross-section

value for 4 energy levels. The parameters of the Bendel function obtained in OMERE 5.2 are $A = 20.00\,MeV$, $B = 36.27\,MeV$ and the cross-section representative values are:

- Proton Kinetic Energy Threshold $E_{th} = 20.00\,Mev$.
- Saturation cross-section $\sigma_{sat} = 4.09 \times 10^{-9} cm^2/device$.

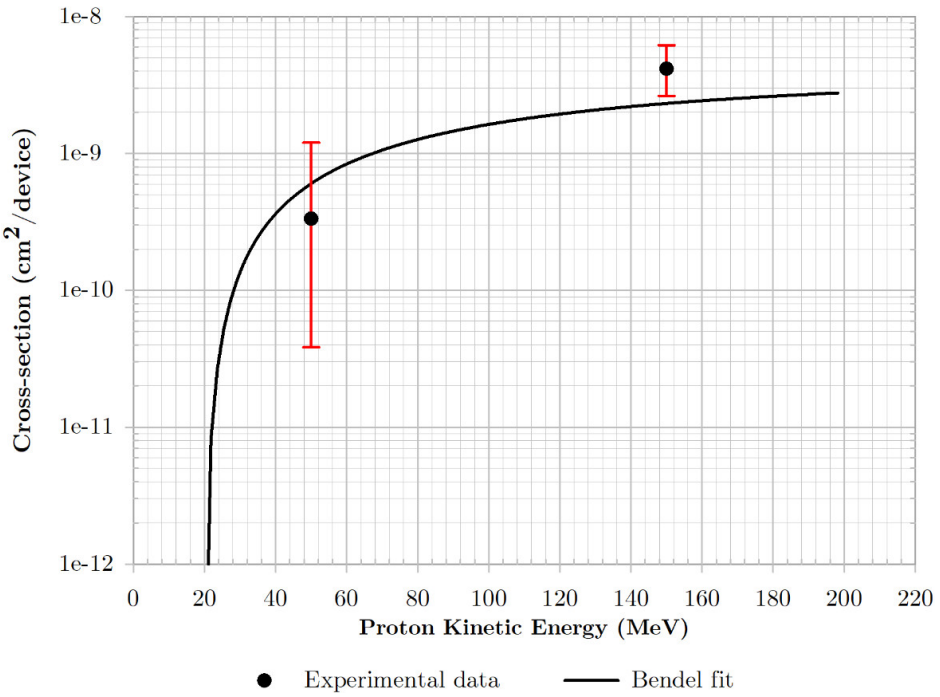


Figure 70 - PIC16F877 EEPROM cross-section.

The summary of the PIC radiation test results is:

- The EEPROM is less sensitive than the flash-based SRAM for reading operations.
- The EEPROM is more sensitive than the flash-based SRAM for writing operations.
- The flash-based SRAM got corrupted values after radiation with beam energy of 150 MeV. However, it could be recovered only after a full re-programming process via PICKit programmer.

6.6.2. Raspberry Pi Results

The purpose of this test is to perform the experiment with the same parameters and characteristics performed in the PIC microcontroller in order to compare their performance. However, at the end the device was tested in a different condition due to when it was tested at the same fluence value $(3 \times 10^9\, proton/cm^2/s)$, the device stopped completely to work. It happened because the Raspberry Pi uses an operative system that constantly is monitoring the CPU and memory regarding errors and if a critical error is detected in the execution, the system notifies it via UART interface and try to fix it automatically. However, if the system is not able to recover from the error, it stops the functionality until a power reset is executed. It is good for the reliability standpoint since the operative system automatically protects the device about errors produced by radiation upsets. Nevertheless, it indicates that the Raspberry Pi is more sensitive to radiation events when is executing its own operative system. Then, in order to perform the test in a meaningful way, the fluence was decreased until the system could continue working without losing its complete functionality. After finding that fluence level, then, the test was performed in the same way followed for the PIC microcontroller. The irradiation parameters for the R-Pi Zero experiment are presented in the Table 25 as well as the corresponding ones for the R-Pi 3B+ in the Table 26. The parameters in the table are calculated as the same way as in the case of the PIC microcontroller case in the numeral 6.6.1, the difference is that in order to change the fluence level, the beam flux is changed (note the differences in the values of the second column in the Table 21, Table 25 and Table 26). The experimental data results with their respective uncertainly error bars are plotted in the Figure 71 and Figure 72; here, it is possible to appreciate how the R-Pi Zero cross-section grows as the proton kinetic energy is increasing following the trend of the expected cross-section curve; however, in the case of the R-Pi 3B+, the errors seem to be more random for every proton energy steps.

Table 25 - Raspberry Pi Zero proton irradiation parameters.

Beam energy [MeV]	**Flux [$proton/cm^2/s$]**	**Irradiation time [s]**	**Desired Fluence [$proton/cm^2$]**	**Dose (1-unit value)**	**Dose Preset [Gy]**	**Measured Dose [Gy]**	**Real applied fluence [$proton/cm^2$]**
20	1.00E+08	30	3.000E+09	1.62E+04	185185.19	185185.00	3.000E+09
50	1.00E+07	30	3.000E+08	1.96E+04	15306.12	15760.00	3.089E+08
100	5.00E+06	30	1.500E+08	2.38E+04	6302.52	6324.00	1.505E+08
150	7.00E+06	30	2.100E+08	2.74E+04	7664.23	7784.00	2.133E+08
200	5.00E+06	50	2.500E+08	4.25E+04	5882.35	5942.00	2.525E+08

Table 26 - Raspberry Pi 3B+ proton irradiation parameters.

Beam energy [MeV]	**Flux [$proton/cm^2/s$]**	**Irradiation time [s]**	**Desired Fluence [$proton/cm^2$]**	**Dose (1-unit value)**	**Dose Preset [Gy]**	**Measured Dose [Gy]**	**Real applied fluence [$proton/cm^2$]**
20	1.00E+06	30	3.000E+07	1.62E+04	1851.85	1921.00	3.112E+07
50	1.00E+06	30	3.000E+07	1.96E+04	1530.61	1607.00	3.150E+07
100	1.00E+06	30	3.000E+07	2.38E+04	1260.50	1298.00	3.089E+07
150	1.00E+06	30	3.000E+07	2.74E+04	1094.89	1141.00	3.126E+07
200	1.00E+06	30	3.000E+07	4.25E+04	705.88	771.00	3.277E+07

Probably, it is due to the Raspberry Pi 3B+ is a quad-core processor and the operating system monitors every core as a different CPU; then, there is a more probability to get a system error than in the case of the Raspberry Pi Zero which is a single-core processor only.

Same as in the case of the PIC, the cross-section experimental values can be fitted using the OMERE 5.2 radiation software with a 2-parameter Bendel and 4-parameter Weibull functions. As shown in the Figure 71 and Figure 72 the Bendel fit provides more accuracy in the estimation of the threshold kinetic energy point than the Weibull fit, even more than the observed in the PIC case. For that reason, the relevant parameters of the cross-section will be obtained from the Bendel fit. For the Raspberry PI Zero, the Bendel fit parameters are: $A = 8.00\,MeV$ and $B = 16.14\,MeV$. The result is shown in the Figure 71 and the cross-section parameters are:

- Proton Kinetic Energy Threshold $E_{th} = 8.00\,MeV$.
- Saturation cross-section $\sigma_{sat} = 1.831 \times 10^{-8} cm^2/device$

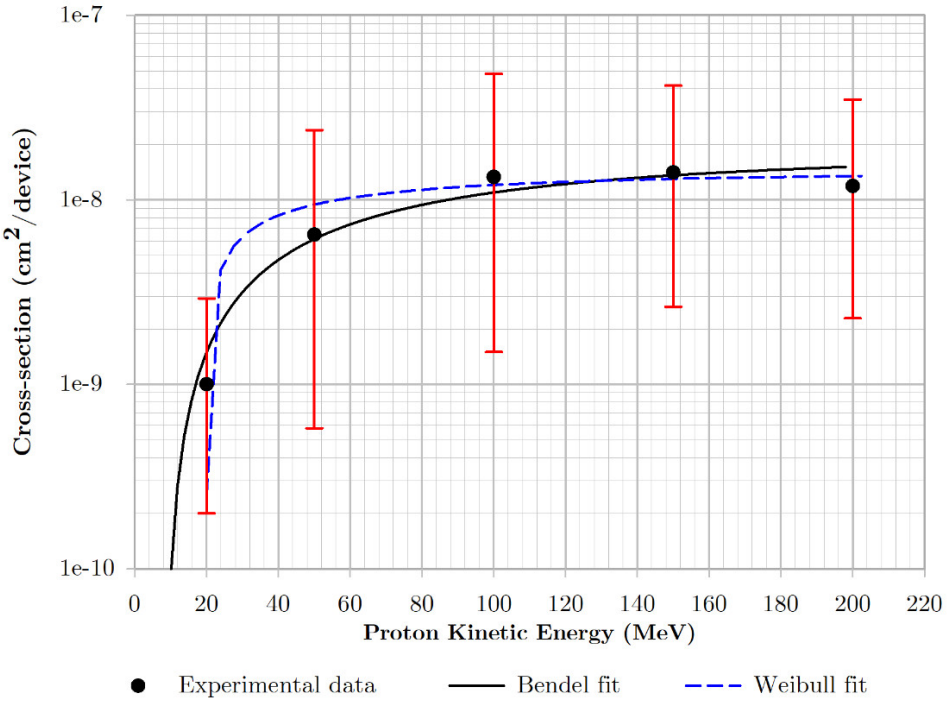


Figure 71 - Raspberry Pi Zero device cross-section.

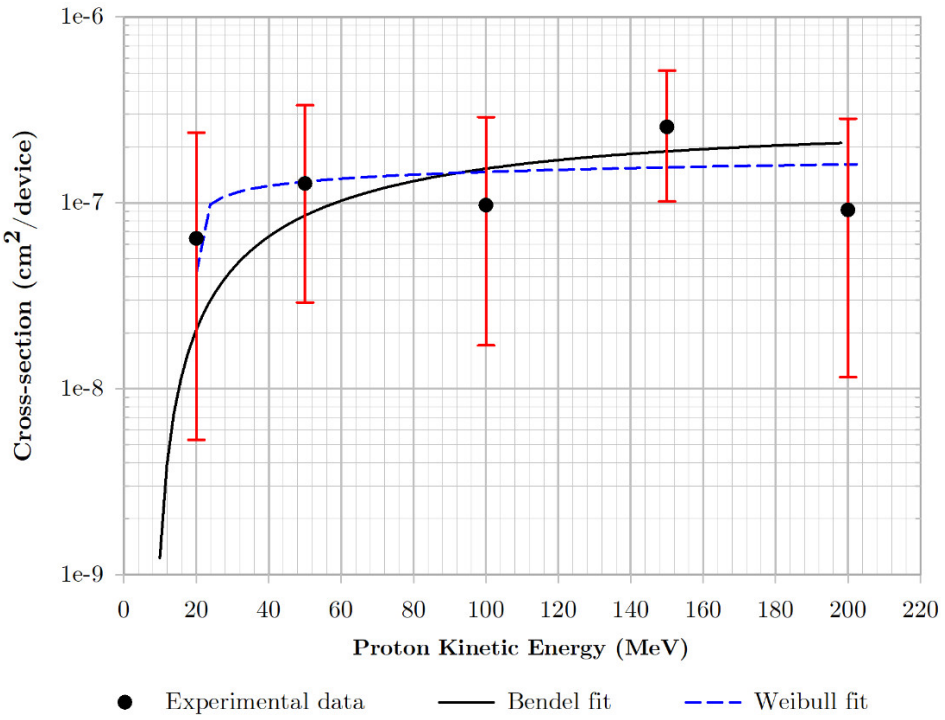


Figure 72- Raspberry Pi 3B+ device cross-section.

For the Raspberry PI 3B+, the Bendel fit parameters are: $A = 8.00\,MeV$ and $B = 19.47\,MeV$. The result is shown in the Figure 72 and the cross-section parameters are:

- Proton Kinetic Energy Threshold $E_{th} = 8.00\,Mev$.
- Saturation cross-section $\sigma_{sat} = 2.54 \times 10^{-7} cm^2/device$

The above shows that the probability of errors in the Raspberry Pi 3B+ is a one-order of magnitude greater than in the case of the Raspberry Pi Zero which indicates that the R-Pi Zero is more tolerant to the radiation effects. It is compressible since the R-Pi Zero worked without a system failure for fluences below $1.5 \times 10^8\,proton/cm^2$ while the Pi 3B+ worked for fluences below $3 \times 10^7\,proton/cm^2$.

To summarize the obtained results and for comparing them for all the tested devices, in the Figure 73, the obtained cross-section for every device is shown and

in the Table 27. Cross-section parameters for all tested devices., the relevant cross-section parameters are presented.

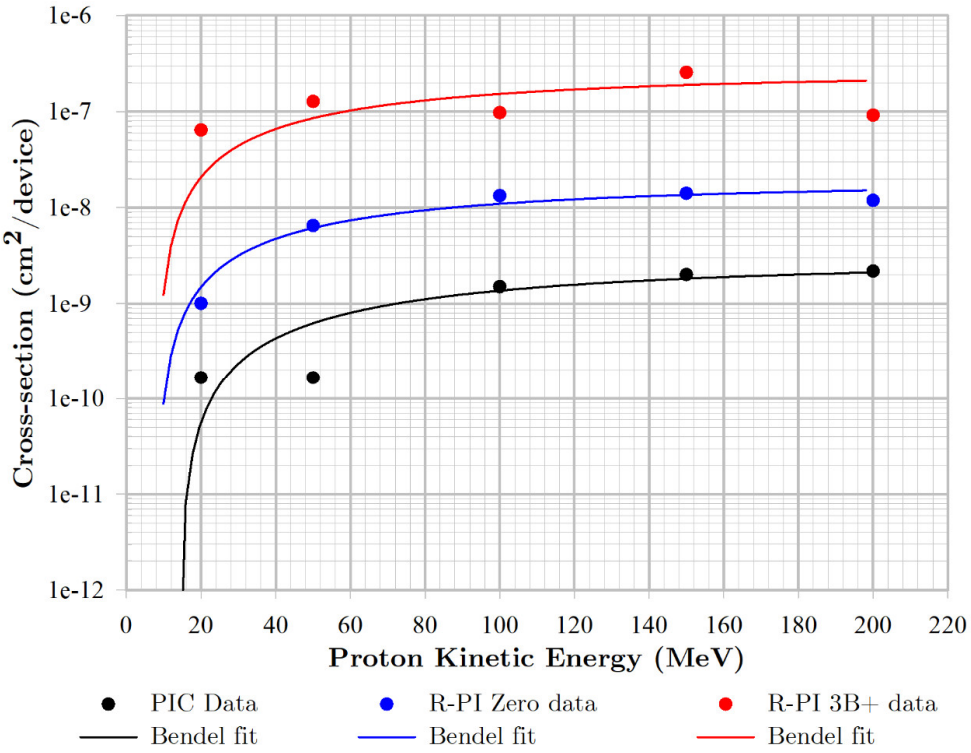


Figure 73. Cross-section data comparison for all tested devices.

Table 27. Cross-section parameters for all tested devices.

	PIC	**R-PI ZERO**	**R-PI 3B+**
$E_{th}\,(MeV)$	14	8	8
$\sigma_{sat}\left(\frac{cm^2}{device}\right)$	2.83E-09	1.83E-08	2.54E-07
Fluence	3.00E+09	2.00E+08	3.0E+07

Finally, the cross-section parameters can be used to estimate the Single Event Upset (SEU) rate on a specific satellite orbit. In order to make that estimation the SPENVIS environment is used. With the software, the first step is to create a project with the mission orbital parameters (for this case, the Ten-Koh orbital parameters) as shown in the Figure 74. Afterwards, the trapped proton flux, the

solar particle mission fluxes and the galactic cosmic ray fluxes should be chosen and executed. For this calculation in particular the selected models were:

- AP-8 MIN trapped proton model
- CREME96 (peak 5min) solar particles model (H - U)
- CREME96 Galactic Cosmic Rays (GCR) particles model (H - U)

Finally, the Short-term SEU rates and LET spectra module should be selected for performing the SEU rate calculation. The module was configured with the parameters shown in the Figure 75 used for the calculation on the Raspberry Pi Zero as an example. As a result, running the module, the SEU rate upsets/device/day can be obtained.

The results of the SEU rate for the three tested devices are shown in the Table 28 with the respective sensitive area used for assumed for each device. Here, it is possible to see that the PIC is the device which has the less SEU rate and the Raspberry Pi 3B+ is the device which has the major value. It was expected since the cross-section values shown that the PIC is less sensitive than the R-Pi Zero and the R-Pi 3B+ in four and five orders of magnitude respectively.

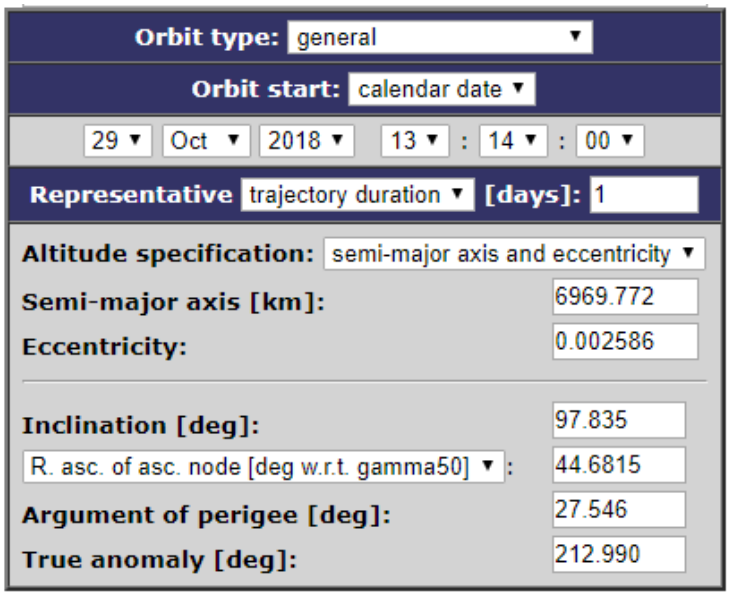


Figure 74. Ten-Koh orbital parameters configured on SPENVIS

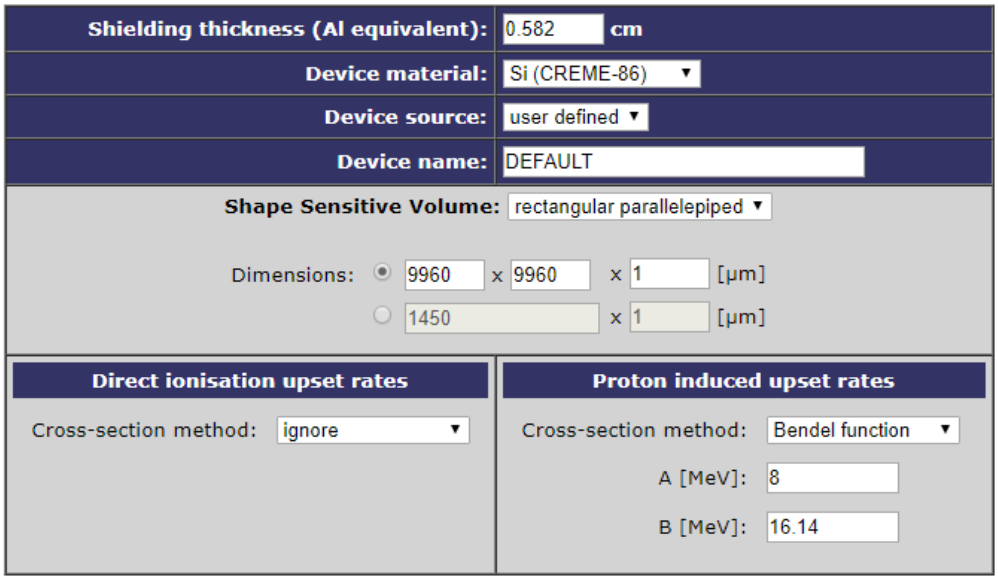


Figure 75. SEU rate calculation parameters configured in SPENVIS

Table 28. SEU rate estimation results.

	PIC	R-PI ZERO	R-PI 3B+
Sensitive Area (cm^2)	0.5	1	1
$SEU_{rate}\left(^{device}/_{day}\right)$	5.84E-1	5.6E+00	7.8E+01

6.7. Discussion

In this numeral, an analysis of the obtained results will be discussed in the base of the test objectives presented in the numeral 6.1.

In the first place, regarding the results presented in the numeral 6.6.1, it is possible to observe that the EEPROM memory in the PIC16F877 microcontroller used in the Ten-Koh subsystems are sensitive to the proton radiation events when write operations are performed. The results show that no SEE were presented in the memory allocations were only reading operations were performed. Also, in the proton energy of 150 MeV, the allocations where writing operations were presented a complete corruption and those were not corrected even after a hard reset of the microcontroller. It means that the EEPROM memory can be

corrupted due to proton radiation events which proves that the issues presented in the numeral 2.2.2 possibly were caused by the LEO radiation environment. We verified the functionality of the DUTs after the test and we found that the corrupted EEPROM memory positions could be reprogrammed successfully which means that the EEPROM were just corrupted temporarily due to the proton radiation effects.

On the other hand, no soft resets were presented during the entire test; however, taking into account that the RAM memory presented several errors during the test and considering that the OBC reset management is commanded by the EPS microcontroller via I2C communication time-out (which is a counter stored temporarily in the RAM memory), if the counter bytes are changed to a random value that exceed the defined time-out, then the EPS will reset the OBC unexpectedly which can explain the resets events analyzed in the numeral 2.2.1.

Regarding the Raspberry Pi modules, the first aspect to take into account is that the test could not be performed whit the same parameters used for the PIC test because using the same amount of proton fluence, the Raspberry Pi modules stopped the operation due to the operating system kernel error protection which indicates that the Raspberry Pi modules running the official operative system (Raspbian) are more sensitive than the PIC microcontroller due to radiation events. Due the above fact, the calculation of the cross-section was more difficult since to for evaluating the error events occurred in the system, it was necessary to analyzing the kernel logs provided by the system via UART interface as shown in this document provided by Texas Instruments [59] which is a difficult task that may produce inaccurate counting of the real error events. However, for this research purposes that is acceptable since the idea was to evaluate the performance of the Raspberry Pi modules running the official Linux distribution and open source tools for nano-satellites applications.

In contrast with the above, the Raspberry Pi modules always recovered the nominal functionality after a power reset due to the kernel error management

mentioned previously, it is beneficial because it reduces the risk to have permanent damage as presented in the EEPROM in the PIC microcontroller. It makes the Raspberry Pi modules a suitable candidate for use in applications that operated during a reduced time like payloads.

Finally, the results also shown that the Raspberry Pi Zero module is less sensitive than the 3B+ for radiation events, aspect to make the Zero module more reliable to be a candidate for future designs.

CHAPTER 7 - CONCLUSION

7.1. Conclusion

In this research an improvement of a Ten-Koh satellite communication system using an SDR architecture is presented. Using the Raspberry Pi module in conjunction with the LimeSDR mini RF module, we achieve more flexibility and less limitations comparing with the Ten-Koh presented system. The possibility to use a Linux distribution in conjunction with Python and GNU radio suite, gives to developers several tools to develop flexible communications systems reducing the prototyping and software development time. GNU radio offers a bunch of precompiled signal processing blocks and even it is possible to find blocks developed for enthusiasms and specialists (e.g. SatNOGS) that can be used to facilitate the SDR design for satellite applications. In addition, several SDR hardware manufacturers offer precompiled blocks on the GNU radio to configure their modules easily, for example, the LimeSDR mini module used in the proposed architecture.

Additionally, the research proved that it is possible to develop a significantly low-cost, functional SDR system with the trade-off of compromising parameters like the receiver sensitivity and the power consumption mainly. It became an interesting option for missions, which the budget is very limited and that are able to overcome with the presented limitations, especially for non-developed countries.

On the other hand, there are some aspects to keep in mind which merits more researching. For example, the Raspberry Pi is a module created for general purpose projects and the Linux Raspbian distribution is a good operating system, but it does not focus on real time applications and can be not suitable enough for satellite applications. Also, the Raspberry Pi power consumption can be high for small satellite applications (specially CubeSats), even in the Zero model is slightly

high in comparison with typical low power consumption microcontrollers used in small satellite applications.

As experimented in Ten-Koh mission, satellite systems are becoming more complex because of the diversification of the missions and payloads. For that reason, new satellite communication systems have to be flexible, reconfigurable and adopt functionalities in higher frequency bands e.g. S and K bands that enable large capacity communications. Although old PICs are good microcomputers with space heritage, they cannot cope with these requirements very well. That is why in the future, we will be forced to use new microcomputers such as a Raspberry Pi that can supply satisfactorily the requirements discussed. Also, mitigation techniques against radiation effects and reduction of power consumption are challenges to be faced.

Regarding the performed radiation test for the PIC16F877 used in the Ten-Koh subsystems, it is possible to conclude that some of the EEPROM memory failures and the reset events presented in some of the subsystems were caused due to radiation events. The PIC microcontroller results shown that the RAM memory is more sensitive to the radiation effects than the EEPROM and the FLASH memory was permanently corrupted using fluences under $6.00 \times 10^{-9}\ \text{proton/cm}^2/\text{s}$ but recovered after a full chip re-programming externally. It explains why in the Ten-Koh mission, the subsystems started to present undesired functionalities that could not be recovered even after performing a hard reset in the system.

Regarding the Raspberry Pi radiation performance, it is possible to conclude that the obtained tolerances are lower than the obtained from the PIC which means that probably is not a recommended device for a critical satellite mission application e.g. bus subsystems. However, the device always recovered the normal functionality after a power reset which indicates that the use of the native operating system ensure a reliable protection against permanent damage as presented in the PIC microcontroller EEPROM. It makes the Raspberry Pi a

good candidate for developing satellite systems which do not have to work continuously during a long time such as payloads.

Finally, regarding the FPGA reprogramming system implemented for the SDR transmitter optimization, we probed the correct functionality for the Spartan 6 and Kintex 7 Xilinx families and we tested that the reprogramming times are suitable to be considered in using for future nanosatellite missions. The system can be considered for using in another subsystem or payload for maintenance and fault recovery purpose.

7.2. Future perspectives

In this research, we described a methodology to develop an SDR for nanosatellite applications using a Raspberry Pi and a LimeSDR mini modules optimizing the hardware development, demonstrating that the use of the GNU Radio tools in conjunction with Linux and Python facilitating the complexity that the developing of software requires in this type of systems and performing functional and radiation testing in order to prove that the proposed system performance can meet with Ten-Koh requirements. However, the system can be improved taking into account the following issues found in this research:

- Creating a library with different transceiver modulation combinations in order to be used for the nanosatellite community interested in it.

- Following the proposed methodology using GNU Radio plus python in another specialized operating system for embedded systems applications for improving the overall performance. The Raspberry Pi is compatible with several Linux distributions and real time operating systems and probably will be possible to overcome the issues presented using the stock Linux distribution (Raspbian) used in this research disabling or not including unnecessary modules into the system. It could improve the processing

performance, the real time response and even the power consumption.

- Implementing the proposed SDR architecture in another single processor board, e.g. Zynqberry, beagleboard, etc. There are several options that even include ARM processors and FPGA logic in a single chip which can optimize the power consumption, the processing performance and the software development due the easy integration between the processor part and the FPGA part that this kind of modules offer.

- Developing a safe protocol system to upload the required files for the FPGA reprogramming functionalities, in this research the possibility of reprogramming the transmitter was tested locally, but the protocol used to upload the file is an important aspect that have to be improved in future applications.

- Regarding the radiation test, performing the test using a low-level software in the Raspberry Pi in order to be able to obtain the cross-section in cm^2/bit or $cm^2/byte$ same as performed for the PIC microcontroller. We tested the Raspberry Pi running the test software under the operating system and due to the kernel execution protection, we could find only errors at the system level.

- Another possibility for improving the obtained cross-section results on the Raspberry Pi is to test only the microcontroller externally because we tested the device in the board as is shipped from factory and there are several components near to the processor that received radiation and it could generate undesired system errors. Another recommendation is to isolate/protect the components situated near to the processor using a proper proton degrader e.g. aluminum or solid water with the proper thickness.

- If there is a special requirement of using the PIC16F77 or similar in future missions even knowing the presented limitations in this research. A recommendation for using it is to implement a self-reprogramming system in order to recover the FLASH memory due corruptions produced by radiation effects. We proved that reprogramming externally the device, it recovers the normal functionality after a radiation issue.

APPENDIX 1 - Integrated modulators internal block diagrams

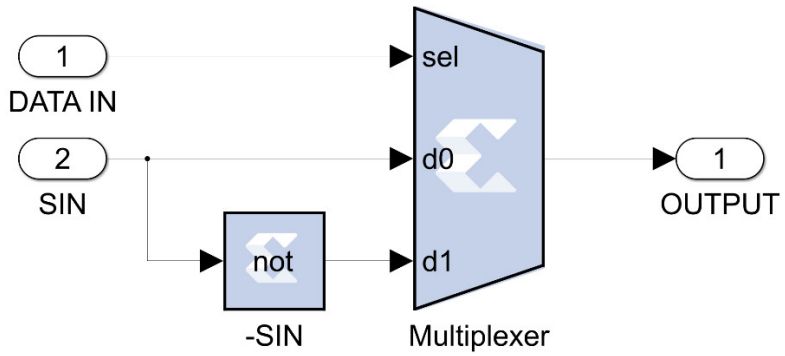


Figure 76 - BPSK Modulator

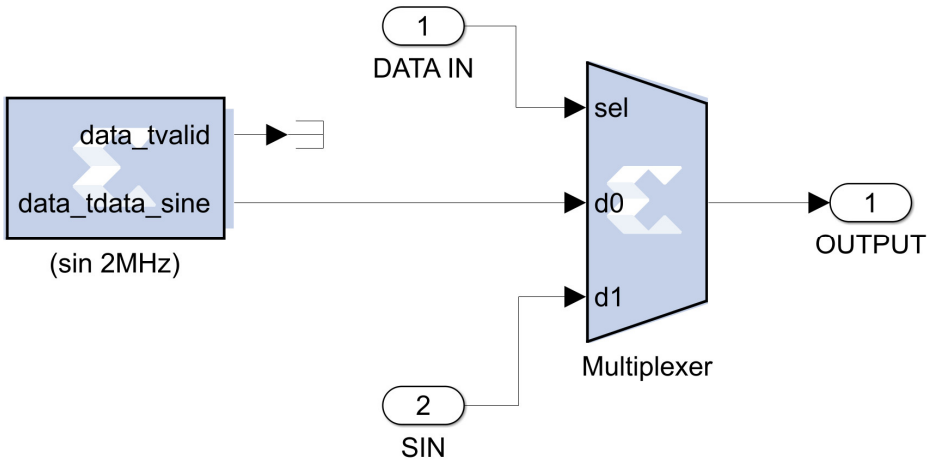


Figure 77 - FSK Modulator

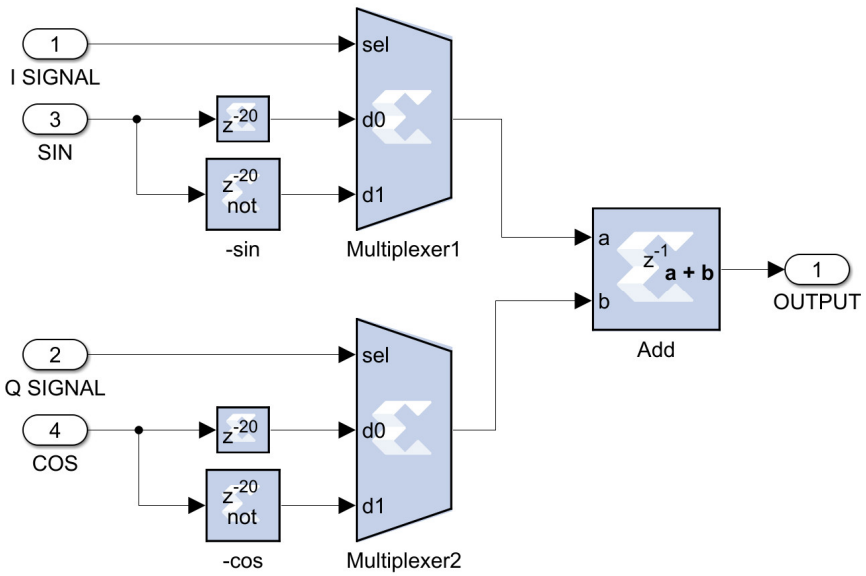


Figure 78 - QPSK Modulator

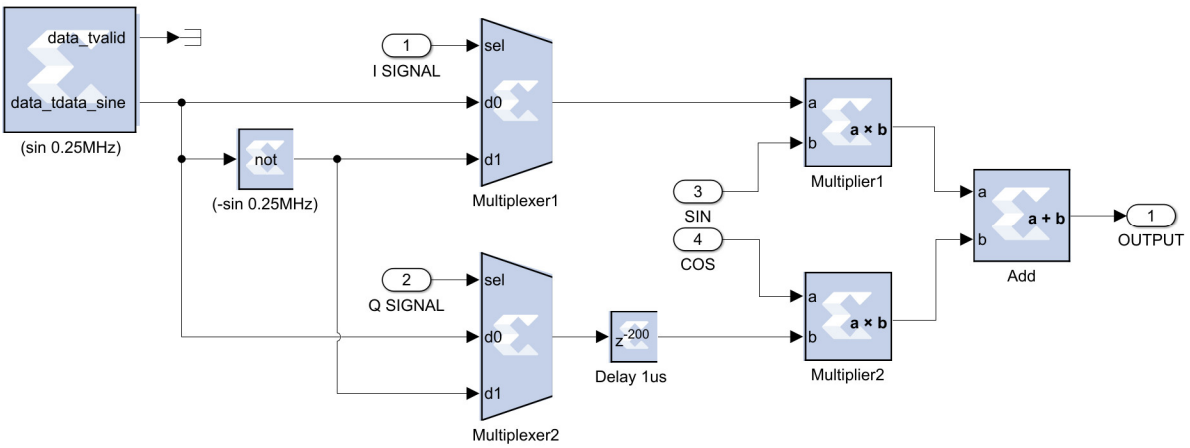


Figure 79 - MSK Modulator

APPENDIX 2 - OpenOCD configuration files description

Raspberry Pi 3B+ and Kintex 7 configuration file example

```
#############################################################################################
# This file is a modification of the original configuration file located in the OpenOCD root
# directory /interface/sysfsgpio-raspberrypi.cfg
#############################################################################################

# Config for using RaspberryPi's expansion header
#
# This is best used with a fast-enough buffer but also
# is suitable for direct connection if the target voltage
# matches RPi's 3.3V
#
# Do not forget the GND connection, pin 6 of the expansion header.
#

#############################################################################################
# Raspberry Pi JTAG GPIO interface configurations and definitions
#############################################################################################

interface sysfsgpio          # Define the interface (sysfs) to control the R-Pi GPIOs

transport select jtag        # Select the JTAG mode into the OpenOCD

# Each of the JTAG lines need a gpio number set: TCK TMS TDI TDO
# Header pin numbers: 23 22 19 21   # Corresponding pins designators into the R-Pi header
sysfsgpio_jtag_nums 11 25 10 9      # Defining the corresponding Linux GPIO designators

#############################################################################################
# FPGA configuration for SPI flash memory programming
#############################################################################################

source [find cpld/xilinx-xc7.cfg]  # Selecting the Xilinx 7 family configuration file located
                                   # into the OpenOCD root directory “cpld” folder

set _CHIPNAME xc7      # Setting the FPGA name identificator into the JTAG TAP

set _USER1 0x02        # Setting the USER name identificator into the JTAG TAP

if {[info exists JTAGSPI_IR]} {
   set _JTAGSPI_IR $JTAGSPI_IR
} else {
   set _JTAGSPI_IR $_USER1
}

if {[info exists TARGETNAME]} {
   set _TARGETNAME $TARGETNAME
} else {
   set _TARGETNAME $_CHIPNAME.proxy
}

if {[info exists FLASHNAME]} {
   set _FLASHNAME $FLASHNAME
} else {
   set _FLASHNAME $_CHIPNAME.spi
}

target create $_TARGETNAME testee -chain-position $_CHIPNAME.tap
flash bank $_FLASHNAME jtagspi 0 0 0 0 $_TARGETNAME $_JTAGSPI_IR

proc jtagspi_init {chain_id proxy_bit} {
   # load proxy bitstream $proxy_bit and probe spi flash
   global _FLASHNAME
   pld load $chain_id $proxy_bit
   reset halt
   flash probe $_FLASHNAME
}

proc jtagspi_program {bin addr} {
```

```
   # write and verify binary file $bin at offset $addr
   global _FLASHNAME
   flash write_image erase $bin $addr
   flash verify_bank $_FLASHNAME $bin $addr
}

#############################################################################################
# OpenOCD commands to programming the SPI flash memory and the FPGA logic
#############################################################################################
Init     # Initializing the OpenOCD interface

pld load 0 /home/pi/Documents/bscan_spi_bitstreams/bscan_spi_xc7k160t.bit
# Loading the SPI bscan configuration file for programing the flash memory via the Kintex 7 FPGA

reset halt                   # Resetting the FPGA after bscan configuration file programming

flash probe xc7.spi          # Starting to programming the SPI flash memory trough the FPGA

jtagspi_program bpsk.bin 0   # Programming the BPSK modulator via the JTAG interface

shutdown                     # Shutdown the OpenOCD interface
```

Raspberry Pi Zero and Spartan 6 configuration file example

```
#############################################################################################
# This file is a modification of the original configuration file located in the OpenOCD root
# directory /interface/raspberrypi-native.cfg
#############################################################################################

# Config for using RaspberryPi's expansion header
#
# This is best used with a fast-enough buffer but also
# is suitable for direct connection if the target voltage
# matches RPi's 3.3V
#
# Do not forget the GND connection, pin 6 of the expansion header.
#

#############################################################################################
# Raspberry Pi JTAG GPIO interface configurations and definitions
#############################################################################################

interface bcm2835gpio        # Define the interface (bcm2835) to control the R-Pi GPIOs

transport select jtag        # Select the JTAG mode into the OpenOCD

bcm2835gpio_peripheral_base 0x20000000    # GPIO peripheral base value for Raspberry Pi ZERO


# Transition delay calculation: SPEED_COEFF/khz - SPEED_OFFSET
# These depend on system clock, calibrated for stock 700MHz
# bcm2835gpio_speed SPEED_COEFF SPEED_OFFSET
bcm2835gpio_speed_coeffs 113714 28        # GPIO speed coefficients for Raspberry Pi ZERO

# Each of the JTAG lines need a gpio number set: TCK TMS TDI TDO
# Header pin numbers: 23 22 19 21   # Corresponding pins designators into the R-Pi header
bcm2835gpio_jtag_nums 11 25 10 9    # Defining the corresponding Linux GPIO designators

adapter_khz 200        # JTAG adapter frequency for Raspberry Pi ZERO

#############################################################################################
# FPGA configuration for SPI flash memory programming
#############################################################################################

source [find cpld/xilinx-xc6s.cfg]  # Selecting the Xilinx Spartan 6 family configuration file
                                    # located into the OpenOCD root directory “cpld” folder

set _CHIPNAME xc6s     # Setting the FPGA name identificator into the JTAG TAP

set _USER1 0x02        # Setting the USER name identificator into the JTAG TAP
```

```
if {[info exists JTAGSPI_IR]} {
   set _JTAGSPI_IR $JTAGSPI_IR
} else {
   set _JTAGSPI_IR $_USER1
}

if {[info exists TARGETNAME]} {
   set _TARGETNAME $TARGETNAME
} else {
   set _TARGETNAME $_CHIPNAME.proxy
}

if {[info exists FLASHNAME]} {
   set _FLASHNAME $FLASHNAME
} else {
   set _FLASHNAME $_CHIPNAME.spi
}

target create $_TARGETNAME testee -chain-position $_CHIPNAME.tap
flash bank $_FLASHNAME jtagspi 0 0 0 0 $_TARGETNAME $_JTAGSPI_IR

proc jtagspi_init {chain_id proxy_bit} {
   # load proxy bitstream $proxy_bit and probe spi flash
   global _FLASHNAME
   pld load $chain_id $proxy_bit
   reset halt
   flash probe $_FLASHNAME
}

proc jtagspi_program {bin addr} {
   # write and verify binary file $bin at offset $addr
   global _FLASHNAME
   flash write_image erase $bin $addr
   flash verify_bank $_FLASHNAME $bin $addr
}

#############################################################################################
# OpenOCD commands to programming the SPI flash memory and the FPGA logic
#############################################################################################
Init     # Initializing the OpenOCD interface

pld load 0 /home/pi/Documents/bscan_spi_bitstreams/ bscan_spi_xc6slx75.bit
# Loading the SPI bscan configuration file for programing the flash memory via the Spartan 6
# FPGA

reset halt                  # Resetting the FPGA after bscan configuration file programming

flash probe xc6s.spi        # Starting to programming the SPI flash memory trough the FPGA

jtagspi_program qpsk.bin 0  # Programming the QPSK modulator via the JTAG interface

shutdown                    # Shutdown the OpenOCD interface
```

REFERENCES

[1] SatNOGS.org, "SatNOGS WIKI," 2019. [Online]. Available: https://wiki.satnogs.org/Main. [Accessed 14 May 2019].

[2] G. Quintana-Díaz and R. Birkeland, "Software-Defined Radios In Satellite Communications," in *The 4S Symposium*, Sorrento - Italy, 2018.

[3] J. Mitola, "The Software Radio Architecture," *IEEE Communications Magazine,* vol. 33, no. 5, pp. 26-38, 1995.

[4] M. R. Maheshwarappa and C. P. Bridges, "Software defined radios for small satellites," in *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Leicester, 2014.

[5] R. C. Reinhart, T. J. Kacpura, S. K. Johnson and J. P. Lux, "NASA's space communications and navigation test bed aboard the international space station," *IEEE Aerospace and Electronic Systems Magazine,* vol. 28, pp. 4-15, 2013.

[6] E. Kulu, "Nanosats Database," 06 January 2020. [Online]. Available: https://www.nanosats.eu/img/fig/Nanosats_years_2020-01-06_large.png. [Accessed 20 January 2020].

[7] B. Corrado, W. Ebel and S. Jayaram, "Cubesat software defined radio project," in *26th AIAA International Communications Satellite Systems Conference (ICSSC)*, San Diego, 2008.

[8] S. J. Olivieri, J. Aarestad, L. H. Pollard, A. M. Wyglinski, C. Kief and R. S. Erwin, "Modular FPGA-based software defined radio for CubeSats," in *2012 IEEE International Conference on Communications (ICC)*, Ottawa, 2012.

[9] O. Ceylan, A. Caglar, H. B. Tugrel, H. O. Cakar, A. O. Kislal, K. Kula and H. B. Yagci, "Small Satellites Rock A Software-Defined Radio Modem and Ground Station Design for Cube Satellite Communication," *IEEE Microwave magazine,* vol. 17, no. 3, pp. 26-33, 2016.

[10] GomSpace, "GomSpace Software Defined Radio," 2018. [Online]. Available: https://gomspace.com/UserFiles/Subsystems/datasheet/gs-ds-nanocom-sdr-10.pdf. [Accessed 15 January 2020].

[11] Vulcan Wireless INC., "Products," [Online]. Available: http://www.vulcanwireless.com/products/mbt. [Accessed 15 January 2020].

[12] Tethers Unlimited INC., "About SWIFT Radios," [Online]. Available: http://www.tethers.com/SWIFT-RADIOS/. [Accessed 15 January 2020].

[13] Allen Space, "Hardware for nanosatellites," [Online]. Available: https://alen.space/hardware-for-nanosatellites/. [Accessed 15 January 2020].

[14] R. Dewitt, D. Duston and A. K. Hyder, "PHYSICS OF ENERGETIC PARTICLE INTERACTIONS," in *The Behavior of Systems in the Space Environment*, Netherlands, Springer Netherlands, 1993, pp. 353-381.

[15] European Cooperation for Space Standardization (ECSS), "ECSS-E-HB-10-12A – Calculation of Radiation and Its Effects and Margin Policy Handbook," 17 December 2010. [Online]. Available: http://ecss.nl/get_attachment.php?file=handbooks/ecss-e-hb/ECSS-E-HB-10-12A17December2010.pdf. [Accessed 28 June 2019].

[16] T. Dachev, B. Tomov, P. Dimitrov, Y. Matviichuk, K. Fujitaka, Y. Uchihori and H. Kitamura, "Calibration of LIULIN-4 type system at HIMAC with heavy ions (11P-084)," 2001.

[17] F. Kuroiwa, K. Okuyama, M. Nishio, H. Morita, B. Szasz, S. Bendoukha, P. Saganti and D. Holland, "A Design Method of an Autonomous Control System for a Deep-Space Probe," *of the Japan Society for Aeronautical and Space Sciences, Aerospace Technology,* vol. 14, pp. 105-112, 2016.

[18] F. Kuroiwa, S. Bendoukha, K. Okuyama, H. Morita and M. Nishio, "A Redundancy and Operation of Power Control System for a Deep-space Small Probe," *Journal of Automation and Control Engineering,* pp. 353-359, 2016.

[19] R. Funase, Y. Nakamura, M. Nagai, T. Eishima, K. Nakada, A. Enokuchi, C. Yuliang, E. Takei and S. Nakasuka, "University of Tokyo's Student Nano-Satellite Project CubeSat-XI and Its On-Orbit Experiment Results," in *IFAC Proceedings*, Saint Petersburg, 2004.

[20] H. Kawakubo, "Hardware Development of a Microcontroller Board for a Small Satellite," in *Proceedings of the 16th AIAA/USU Conference on Small Satellites*, Utah, 2002.

[21] Microchip Technology Inc., "28/40-Pin 8-Bit CMOS FLASH Microcontrollers," 2013. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/30292D.pdf. [Accessed 14 May 2019].

[22] T. Pann, "The South Atlantic Anomaly," 2007. [Online]. Available: https://www.bibliotecapleyades.net/ciencia/ciencia_earthchanges49.htm. [Accessed 15 April 2019].

[23] T. Dachev, B. Tomov, Y. Matviichuk, P. Dimitrov, S. Vadawale, J. Goswami, G. De Angelis and V. Girish, "An overview of RADOM results for earth and moon radiation environment on Chandrayaan-1 satellite," *Advances in Space Research,* vol. 48, no. 5, pp. 779-791, 2011.

[24] T. Dachev, J. Semkova, B. Tomov, Y. Matviichuk, P. Dimitrov, R. Koleva, S. Malchev, N. Bankov, V. Shurshakov, V. Benghin, E. Yarmanova, O. Ivanova, D. Häder, M. Lebert, M. Schuster, G. Reitz, G. Horneck, Y. Uchihori, H. Kitamura, O. Ploc, J. Cubancak and I. Nikolaev, "Overview of the Liulin type instruments for space radiation measurement and their scientific results," *Life Sciences in Space Research,* vol. 4, pp. 92-114, 2015.

[25] Nishimusen CO., "TXE430MFMCW-302A," 2016. [Online]. Available: http://www.nishimusen.co.jp/eisei2016/eisei2016.htm. [Accessed 14 May 2019].

[26] GNU Radio.org, "GNU Radio Manual and C++ API Reference," 2019. [Online]. Available: https://www.gnuradio.org/doc/doxygen/build_guide.html. [Accessed 14 May 2019].

[27] SatNOGS.org, "SatNOGS GNU Radio Out-Of-Tree module," 2019. [Online]. Available: https://gitlab.com/librespacefoundation/satnogs/gr-satnogs. [Accessed 14 May 2019].

[28] D. Estevez, "GNU Radio decoders for several Amateur satellites," 2019. [Online]. Available: https://github.com/daniestevez/gr-satellites. [Accessed 14 May 2019].

[29] T. Kuester, "gr-bruninga GNU Radio Out-Of-Tree module," 2017. [Online]. Available: https://github.com/tkuester/gr-bruninga. [Accessed 14 May 2019].

[30] D. Smith, "Digital Modulation Techniques," in *Digital Transmission Systems*, Boston, Springer, 2004, p. 378.

[31] IEEE Standard Association, "1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture," [Online]. Available: https://standards.ieee.org/standard/1149_1-2013.html. [Accessed 23 February 2020].

[32] Lauterbach GmbH, "Training JTAG Interface," 06 November 2019. [Online]. Available: https://www2.lauterbach.com/pdf/training_jtag.pdf. [Accessed 23 February 2020].

[33] Xilinx Inc., "Platform Cable USB II," 6 August 2018. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds593.pdf. [Accessed 23 February 2020].

[34] Xilinx Inc, "Xilinx In-System Programming Using an Embedded Microcontroller (ISE Tools)," 18 May 2017. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp058.pdf. [Accessed 23 February 2020].

[35] UrJTAG.org, "UrJTAG - Universal JTAG library, server and tools," [Online]. Available: http://urjtag.org/. [Accessed 23 February 2020].

[36] The OpenOCD Project, "Open On-Chip Debugger," [Online]. Available: http://openocd.org/. [Accessed 23 February 2020].

[37] HuMANDATA LTD., "XCM-110 シリーズ," [Online]. Available: https://www.hdl.co.jp/XCM-110/. [Accessed 23 February 2020].

[38] HuMANDATA LTD, "XCM-112 シリーズ," [Online]. Available: https://www.hdl.co.jp/XCM-112/. [Accessed 23 February 2020].

[39] Xilinx Inc., "Using SPI Flash with 7 Series FPGAs," 18 October 2016. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp586-spi-flash.pdf. [Accessed 23 February 2020].

[40] The OpenOCD Project, "OpenOCD - Open On-Chip Debugger Repository," [Online]. Available: https://sourceforge.net/p/openocd/code/ci/master/tree/. [Accessed 24 February 2020].

[41] Ardafruit Industries, "Compiling OpenOCD," 16 March 2016. [Online]. Available: https://learn.adafruit.com/programming-microcontrollers-using-openocd-on-raspberry-pi/compiling-openocd. [Accessed 23 February 2020].

[42] The OpenOCD Project, "Open On-Chip Debugger: OpenOCD User's Guide," 20 April 2020. [Online]. Available: http://openocd.org/doc/pdf/openocd.pdf. [Accessed 22 April 2020].

[43] eLinux.org, "RPi GPIO Code Samples," 8 April 2019. [Online]. Available: https://elinux.org/RPi_GPIO_Code_Samples. [Accessed 23 February 2020].

[44] A. ". Huang, "netv2mvp-scripts - Various scripts for NeTV2MVP," 6 October 2018. [Online]. Available: https://github.com/bunnie/netv2mvp-scripts. [Accessed 24 February 2020].

[45] QUARTIQ, "bscan_spi_bitstreams repository," 18 December 2019. [Online]. Available: https://github.com/quartiq/bscan_spi_bitstreams. [Accessed 24 February 2020].

[46] M. Rupprecht – DK3WN, "Amateur Radio – PEØSAT," 2019. [Online]. Available: https://www.pe0sat.vgnet.nl/decoding/tlm-decoding-software/dk3wn/. [Accessed 14 May 2019].

[47] ISIS – Innovative Solutions In Space, "VHF/UHF duplex transceiver," 2016. [Online]. Available: https://www.isispace.nl/wp-content/uploads/2016/02/VHF-UHF-Full-Duplex-Transceiver-Brochure-web.pdf. [Accessed 14 May 2019].

[48] A. I. Perez-Neira and M. R. Campalans, "Chapter 2 - Different views of spectral efficiency," in *Cross-Layer Resource Allocation in Wireless Communications*, Academic Press, 2008, pp. 13-33.

[49] R. Patidar, S. Roy and T. R. Henderson, "Technical report on validation of error models for 802.11n," University of Washington Seattle, Seattle, 2017.

[50] The Wakasawan Energy Research Center, "Multipurpose Synchrotron Tandem Accelerator W-MAST," [Online]. Available: http://www.werc.or.jp/outline/shisetsu/gaiyo/sinkuro.html. [Accessed 24 November 2019].

[51] The Wakasa Wan Energy Research Center, "About WERC," July 2011. [Online]. Available: http://www.werc.or.jp/enenews/pdf/pamphlet_english.pdf. [Accessed 21 November 2019].

[52] BNL - NASA Space Radiation Laboratory, "NSRL User Guide," [Online]. Available: https://www.bnl.gov/nsrl/userguide/let-range-plots.php#silicon. [Accessed 26 November 2019].

[53] European Space Components Coordination (ESCC), "ESCC BasicSpecification 25100: Single Event Effects Test Method and Guidelines," October 2014. [Online]. Available: https://escies.org/download/webDocumentFile?id=62690. [Accessed 1 July 2019].

[54] G. Toumbas, "Raspberry Pi - Radiation Experiment," B.S. Thesis, University of Surrey, May 2018. [Online]. Available: http://personal.ee.surrey.ac.uk/Personal/C.Bridges/AAReST/Files/2018,%20George%20Toumbas,%20BSc%20Thesis.pdf. [Accessed 15 December 2019].

[55] D. Honess and O. Quinlan, "Astro Pi: Running your code aboard the International Space Station," *Acta Astronautica,* vol. 138, pp. 43-52, 2017.

[56] The Space Environment Information System - SPENVIS, "Single Event Upsets (SEUs)," 12 March 2018. [Online]. Available: https://www.spenvis.oma.be/help/background/creme/creme.html#SEU. [Accessed 20 December 2019].

[57] S. Buchner, P. Marshall, S. Kniffin and K. LaBel, "Proton Test Guideline Development - Lessons Learned," 8 August 2002. [Online]. Available: https://radhome.gsfc.nasa.gov/radhome/papers/Proton_testing_guidelines_2002.pdf. [Accessed 20 January 2020].

[58] TRAD Tests & Radiations, "Our Radiation Software - OMERE," [Online]. Available: https://www.trad.fr/en/space/omere-software/. [Accessed 20 January 2020].

[59] Texas Instruments Incorporated, "Debugging Embedded Linux Systems: Understand Kernel Oops Logs," 2017. [Online]. Available: https://training.ti.com/sites/default/files/docs/Kernel-Debug-Series-Part6-understand-kernel-oops.pdf. [Accessed 15 September 2019].

[60] European Cooperation for Space Standardization (ECSS), "Radiation Standards and Guidelines," October 2014. [Online]. Available: https://escies.org/download/webDocumentFile?id=62690. [Accessed 20 January 2020].