Lucas Matana Luza

A CONTRIBUTION TO THE IN-ORBIT VALIDATION OF A RADIATION-HARDENED COMMUNICATION PLATFORM TO BE USED IN SMALL SATELLITES

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Engenharia Elétrica.
Orientador: Prof. Eduardo Augusto Bezerra, PhD.
Coorientador: Luigi Dilillo, PhD.

Florianópolis

2019

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Luza, Lucas Matana
A contribution to the in-orbit validation of a radiation-hardened communication platform to be used in small satellites / Lucas Matana Luza ; orientador, Eduardo Augusto Bezerra, coorientador, Luigi Dilillo, 2019.
142 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Engenharia Elétrica, Florianópolis, 2019.

Inclui referências.

1. Engenharia Elétrica. 2. CCSDS. 3. Nanossatélite. 4. Módulo de comunicação. 5. FPGA resistente a radiação. I. Bezerra, Eduardo Augusto. II. Dilillo, Luigi. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

Lucas Matana Luza

A CONTRIBUTION TO THE IN-ORBIT VALIDATION OF A RADIATION-HARDENED COMMUNICATION PLATFORM TO BE USED IN SMALL SATELLITES

Esta Dissertação foi julgada adequada para obtenção do Título de "Mestre em Engenharia Elétrica" e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica.

Florianópolis, 18 de Março de 2019.

Prof. Bartolomeu Ferreira Uchoa-Filho, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

---

Prof. Eduardo Augusto Bezerra, PhD.
Orientador
Universidade Federal de Santa Catarina

Luigi Dilillo, PhD.
Coorientador
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier.

Banca Examinadora:

Laio Oriel Seman, Dr.
Universidade Federal de Santa Catarina

Leonardo Kessler Slongo, Dr.
Universidade Federal de Santa Catarina

Prof. Marcelo Daniel Berejuck, Dr.
Universidade Federal de Santa Catarina

To my family, especially my parents who give me support for the fulfillment of my dreams.

ACKNOWLEDGEMENTS

First of all, I would like to thank my family, especially my parents Milton and Néli, who with love and hard work have given me the necessary tools to achieve my goals, I will never be able to give you back this effort.

I want to thank my advisor Prof. Eduardo Augusto Bezerra for the possibility of working under his guidance, for the knowledge transmitted and the attention received during this work. I would also like to thank my co-advisor Prof. Luigi Dilillo who has always been willing to give the necessary support for the proper development of the work.

I am grateful to my friends and Masters colleagues Elder Dominghini Tramontin and Cézar Antônio Rigo, where they were not mere co-workers, but they were great companies during these last years. Your support was essential for this work.

Also, I want to thank my dear friends Felipe Andreis and Alex Colussi for all the support, companionship and for making me feel at home even though I am several kilometers away.

I thank Intel for making available, through the student license, the ModelSim tool that is indispensable in the simulation of digital circuits.

This work was conducted during a scholarship supported by the CAPES at the Federal University of Santa Catarina.

"Forget your lust for the rich man's gold
All that you need is in your soul
And you can do this, if you try
All that I want for you, my son, is to be satisfied".

Gary Rossington / Ron Van Zant

RESUMO

Durante a última década, o número de missões espaciais utilizando nanossatélites aumentou consideravelmente devido a utilização do padrão CubeSat. O padrão CubeSat trouxe uma opção viável tanto para a comunidade acadêmica quanto para as empresas, uma vez que traz um orçamento consideravelmente reduzido quando comparado a modelos desenvolvidos por grandes corporações. Entretanto, diferentes protocolos de comunicações são empregados, diminuindo a integração entre as missões espaciais. Contudo, os protocolos definidos pelo CCSDS surgem como uma forte alternativa para quebrar essa barreira de integração, uma vez que já foram utilizados em mais de 900 missões espaciais. Com isso, este trabalho descreve o desenvolvimento de um módulo de comunicação para nanossatélites seguindo as recomendações propostas pelo CCSDS. Este trabalho tem como dispositivo alvo um novo FPGA endurecido contra os efeitos da radiação (*radiation-hardened*). Esse FPGA foi desenvolvido por uma companhia francesa e tem um grande potencial de utilização em novas missões espaciais devido ao fato de não ter as restrições provindas das regulamentações ITAR e EAR. Para alcançar os objetivos, o trabalho faz uso de uma unidade de telemetria e telecomandos e desenvolve uma arquitetura para um OBDH com o propósito de validar os telecomandos recebidos e gerar telemetria, tanto no aspecto de dados de status quanto para dados científicos. A implementação proposta por este trabalho tem como base a arquitetura do Payload-X, que foi desenvolvido para ser operado como carga útil juntamente com o nanossatélite FloripaSat-I. Testes em *hardware-in-the-loop* aplicados a casos de usos do sistema mostraram a validade dos fluxos de telecomando e telemetria, sendo também verificadas a alocação de recursos do FPGA pela implementação e a utilização do *Static Time Analisys* para definir a viabilidade da aplicação.

Palavras-chave: CCSDS; Nanossatélite; Módulo de Comunicação; FPGA resistente a radiação.

RESUMO EXPANDIDO

Introdução

Nos últimos anos, o Grupo de Sistemas Embarcados (GSE) da UFSC vem realizando atividades de pesquisa e desenvolvimento visando a concepção de uma missão completa de um CubeSat. O primeiro satélite desenvolvido no âmbito dessa iniciativa recebeu a denominação FloripaSat-I. Levando em consideração os trabalhos do grupo, pode-se observar que FPGAs são comumente utilizados no desenvolvimento de componentes imprescindíveis nas missões, por exemplo, os computadores de bordo e unidades de comunicação. Entretanto, técnicas de tolerância à radiação são necessárias uma vez que prótons e elétrons presos nos cinturões de radiação da Terra, tal qual os raios cósmicos, representam um dano real para a eletrônica, que deve operar de maneira confiável no ambiente espacial. Nesse ponto, o FPGA tolerante à radiação desenvolvido pela empresa francesa NanoXplore, apresenta a confiabilidade necessária para este tipo de missão e ultrapassa as barreiras impostas pelas regulações do ITAR e EAR que trazem dificuldades nas aquisições de componentes para missões espaciais providos por empresas estadunidenses. Além do mais, a padronização dos protocolos de comunicação para nanosatélites tem o objetivo de remover barreiras entre missõs espaciais, neste aspecto, as recomendações propostas pelo CCSDS (do inglês *The Consultative Committee for Space Data Systems*) trazem uma padronização para comunicações espaciais, sendo que as mesmas já foram utilizadas em mais de 900 missões.

Objetivos

O trabalho tem como objetivo geral o desenvolvimento de um módulo de comunicação para CubeSat utilizando as recomendações CCSDS aplicado em um FPGA tolerante à radiação. Os objetivos específicos deste trabalho contemplam: a utilização de uma unidade de telemetria e telecomandos e o desenvolvimento do componente de manipulação de dados (OBDH) compatível com as recomendações CCSDS para validar os telecommandos e gerar telemetria; E a aplicação desta unidade em um estudo de caso baseado na arquitutera do Payload-X, que foi desenvolvido para ser uma carga útil na missão FloripaSat-I.

Metodologia

Primeiramente, esse trabalho implementa o módulo comunicação uti-

lizando a linguagem de descrição de hardware VHDL, sendo aplicadas máquinas de estados para controle dos blocos lógicos desenvolvidos. A codificação das máquinas de estado segue o modelo proposto pela Xilinx no *XTS User Guide* e a implementação é fisicamente testada com o auxílio da placa de desenvolvimento da empresa NanoXplore. Foi possível testar o sistema utilizando a ferramenta Cortex que é amplamente utilizada na área espacial para sistemas de comunicação. O módulo de comunicação é composto pela unidade de telemetria e telecomandos (UTMC) e de um manipulador de dados (OBDH). A UTMC utilizada é fruto das pesquisas e desenvolvimento do grupo GSE e, como contribuição desse trabalho de mestrado, recebeu alterações para que pudesse ser utilizada em conjunto com o BRAVE FPGA, com o foco em modificações de processos para atingir a frequência de operação desejada e reformulação da descrição para evitar a inferência de latches na aplicação. A UTMC é responsável pela codificação e decodificação dos dados e realiza a manipulação dos dados na camada de *Data Link* proposta pelo CCSDS. O OBDH proposto neste trabalho tem como objetivo a validação dos telecomandos recebidos, e a geração de telemetria com dados de status e científicos. Para isso a arquitetura possui diferentes camadas que realizam processos de verificação, codificação, decodificação e empacotamento dos dados. O módulo de comunicação foi aplicado em um estudo de caso utilizando a plataforma denominada Payload-X, que foi desenvolvida durante esse trabalho de mestrado em conjunto com outros mestrados realizados por colegas do grupo de pesquisa. O Payload-X tem como uma de suas características permitir a modificação da configuração presente no FPGA por meio de um mecanismo de reconfiguração que utiliza dados recebidos durante o voo. O Payload-X será integrado a missão FloripaSat na forma de carga útil. E é previsto a utilização de dois diferentes modos de operação focando a validação da estrutura desenvolvida.

Resultados e Discussão

Este trabalho apresentou o desenvolvimento de um módulo de comunicação para pequenos satélites baseado em tecnologia de hardware reconfigurável (FPGA). O foco principal deste trabalho foi a construção de uma arquitetura baseada nas recomendações do CCSDS. A arquitetura proposta visa um equilíbrio para atender às demandas específicas do estudo de caso e ser genérica o suficiente para ser aplicada em outros cenários. Diante disso, todos os blocos OBDH foram construídos para integrar este modelo com novos blocos que atenderão às demandas de uma missão futura. Além disso, a implementação foi simulada prin-

cipalmente usando a ferramenta ModelSim, e foi testada fisicamente com a ajuda da placa de desenvolvimento da empresa NanoXplore, fabricante do FPGA utilizado. Como os resultados apresentados são satisfatórios para a aplicação proposta, é possível concluir que o trabalho alcançou os objetivos pretendidos. É importante ressaltar que o trabalho tem continuidade no enfrentamento da missão FloripaSat-I, uma vez que está embutido na arquitetura desenvolvida para o Payload-X, que é o resultado do trabalho conjunto com os colegas do grupo de pesquisa. Durante o desenvolvimento do trabalho foi possível deteminar a possibilidade de trabalhos futuros: como a validação da arquitetura fazendo uso da placa desenvolvida para o Payload-X que tem previsão de ficar pronta nos dias que precedem a finalização deste trabalho; a implementação do algoritmo de controle e teste das memórias SRAM; aplicação dos serviços propostos pelo padrão pelo ECSS PUS; e o estudo e implementação de um barramento de comunicação *On-Chip*, como por exemplo, o padrão AMBA desenvolvido pela ARM.

Palavras-chave: CCSDS; Nanossatélite; Módulo de Comunicação; FPGA resistente a radiação.

ABSTRACT

During the last decade, the number of space missions using nanosatellites has increased considerably, this is due to the use of the CubeSat standard that has brought a viable option for both the academic community and the companies since it brings a considerably reduced budget when compared to models developed by large corporations. However, different communication protocols are employed, reducing the integration between these space missions. Though, the protocols defined by the CCSDS appear as a strong alternative to break this integration barrier since they have already been used in more than 900 space missions. As a result, this work describes the development of a communication module for nanosatellites following the recommendations proposed by the CCSDS. This work has as its target device a new radiation-hardened FPGA developed by a French company that has a high potential usage in new space missions since it does not have the restrictions coming from the ITAR and EAR regulations. The work makes use of a telemetry unit and remote controls and develops a platform for an OBDH to validate the incoming telecommands and to generate telemetry, both in terms of status data and scientific data. The implementation proposed by this work is based on the Payload-X architecture, which was developed to be used as a payload in the FloripaSat-I nanosatellite. Hardware-in-the-loop tests applied to system use cases showed the validity of telecommand and telemetry flows, as well as the allocation of FPGA resources through implementation and the use of Static Time Analysis to define the feasibility of the application.

Keywords: CCSDS; Nanosatellite; Communication Module; Radiation-Hardened FPGA.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABREVIATIONS AND SYMBOLS

ACG	Automatic Control Gain
ARM	Advanced RISC Machine
BCH	Bose–Chaudhuri–Hocquenghem
BRAVE	Big Re-programmable Array for Versatile Environments
CAD	Computer Aided Design
CADU	Channel Access Data Unit
CCSDS	Consultive Committee for Space Data Systems
CDMS	Command and Data Management System
CDS	CCSDS Day Segmented
CGRS	Cosmic Galactic Rays
CLBs	Configurable Logic Blocks
CLCW	Communications Link Control Word
CLTU	Comunications Link Transmission Unit
cm^3	Cubic centimeter
CME	Coronal Mass Ejection
CMIC	Configuration Memory Integrity Check
COP	Communications Operation Procedure
CPDU	Command Pulse Distribution Unit
CRC	Cyclic Redundancy Check
CUC	CCSDS Unsegmented
DD	Displacement Damage
DMR	Dual Modular Redundancy
DPRAM	Dual-Port RAM
DUT	Device Under Test
EDAC	Error Detection and Correction
EPS	Electrical Power System
ESA	European Space Agency
eV	Electronvolt
FARM	Frame Acceptance and Reporting Mechanism
FEC	Forward Error Correction
FOP	Frame Operational Procedure
FPGA	Field Programmable Gate Arrays
GeV	Giga Electronvolt
GEO	Geostationary Orbit
HDL	Hardware Description Language
HEO	High Earth Orbit
HUMAN	Housekeeper and Update MANager

IP	Intellectual Property
keV	Kilo Electronvolt
km	Kilometer
LDPC	Low-Density Parity-Check
LEO	Low Earth Orbit
LFSR	Linear Feedback Shift Register
LUT	Look-up Table
MBU	Multiple-Bit Upset
MCU	Multiple-Cell Upset
MEO	Medium Earth Orbit
MeV	Mega Electronvolt
MOS	Metal Oxide Semiconductor
NGHam	Next Generation Ham
OBC	On-Board Computer
OBDH	On-Board Data Handling
I/O	Input/Output
PUS	Packet Utilization Standard
RHBD	Radiation-Hardened By Design
RS	Reed-Solomon
RTL	Register-Transfer Level
SAA	South Atlantic Anomaly
SBU	Single-Bit Upset
SEB	Single Event Burnout
SEEs	Single Event Effects
SEFI	Single Event Functional Interruption
SEGR	Single Event Gate Rupture
SEL	Single Event Latch-up
SET	Single Event Transient
SRAM	Static Random Access Memory
SSI	Synchronous Serial Interface
TET-R	Total-Elapsed-Time Recorder
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
UART	Universal Asynchronous Receiver/Transmitter
UTMC	Unit of Telemetry and Telecommand
VHDL	VHSIC Hardware Description Language

CONTENTS

1 INTRODUCTION

Since its set up in 2002, the activities performed at the the Embedded Systems Group (GSE) are in the context of research and development in the aerospace area. For the last 10 years, the research focus has been on CubeSat nano-satellites, which have a mass between 1 and 10 kg. These small sized satellites are considered a viable option for the academic community, as they require reduced budgets when comparing to the models developed by large corporations (RAZZAGHI, 2012). GSE currently works on a project called FloripaSat-I that aims the development of a complete space mission and proposes a CubeSat 1U with the following subsystems: the onboard computer; electrical power system; communications systems; pure passive attitude control; and payloads (VILLA et al., 2014; SLONGO et al., 2016).

Therefore, FPGAs with features of high performance, high density and hardened to radiation are in great demand. In this direction, NanoXplore SAS implemented a 65 nm complementary metal-oxide-semiconductor (CMOS) FPGA with radiation protection up to 100 000 rads and 60 $MeV \cdot cm^2 \cdot mg^{-1}$, multiple voltage level inputs and outputs, and double data rate type 2 (DDR2) memory support (NANOXPLORE, 2018a). Those devices are called Big Re-programmable Array for Versatile Environments (BRAVE) and because of their recent generation have never been tested in orbit.

This new radiation-hardened FPGA developed by NanoXplore can open new possibilities to new space missions since it does not imply in the regulations from the ITAR (International Traffic in Arms Regulations) and EAR (Export Administration Regulations). Then, it is valid to explicit that the ITAR and EAR are two important United States export control regulations which cover from military products to software and hardware, including space-related technology. These regulations may involve difficult access to components such as radiation-hardened FPGAs manufactured in the USA due to the bureaucratic procedures (COOK, 2010).

Furthermore, the standardization of communications protocol for nanosatellites has the aim to remove integration barriers between space missions. On this aspect, the international recommendations of the Consultative Committee for Space Data Systems (CCSDS) bring a standardization for space communications. These recommendations have already been used in more than 900 space missions (CCSDS, 2019).

The detection and correction of errors aim at bringing reliability

and integrity to the telemetry and remote control system of a space mission. Different error detection and correction codes (EDACs) can be applied in the frame structures that are used with the communications modules such as the BCH (Bose–Chaudhuri–Hocquenghem), Reed-Solomon, Convolutional, CRC (Cyclic Redundancy Check) and so forth (MOON, 2005).

1.1 OBJECTIVE

The global objective of this work is to perform the research and development of a communication module for CubeSats using the CCSDS protocols, targeting their use in a radiation-tolerant FPGA.

Also, this work proposes the utilization of a Unit of Telemetry and Telecommand (UTMC) previously developed to the Brazilian Institute of Space Research (INPE), in the context of a partnership between GSE and the companies Innalogics and AEL Systems. An important contribution of the developed research, was the adaptations performed in the UTMC source code, in order to port it to the new radiation-hardened FPGA considering the necessary modifications to achieve the expected performance. Also, this work should provide the connection between UTMC and an OBDH with reduced functionality, aiming the validation of the received Telecommands and the generation of telemetry data from sensors by using an I^2C communication interface.

A study case was developed targeting the integration of the communication module with the remaining FloripaSat boards, as a payload. To allow this case study, this work targets the development of of the FPGA functionality provided by the Payload-X architecture.

The specific objectives to achieve the general goal include:

- UTMC porting to the BRAVE FPGA;
- Development of an OBDH based on the ECSS PUS recommendations;
- Integration between the UTCM and the OBDH;
- Application of the communication module in a case study based on the Payload-X platform;
- Development of a test setup for the application;
- Simulations and physical test to validate the communication module;

1.2 DOCUMENT ORGANIZATION

Chapter 2 gives an overview of the space radiation environment and the types of effects that are caused on electronic devices. The chapter also includes topics about reconfigurable hardware, communications protocols in small satellites and the CCSDS applications in this scope. Next, Chapter 3 presents related works that focus on different communications protocols applied in CubeSat's missions.

Chapter 4 describes the development process to create the application ported into the BRAVE FPGA and its simulation framework. It also describes how the integration with the FloripaSat-I mission and its use cases is performed. After, in Chapter 5, the results are introduced and discussed. Finally, Chapter 6 presents the conclusions and suggestions for future works.

2 BACKGROUND

As the on-board systems of CubeSats are sensitive to radiation effects, in this chapter there is an overview of radiation sources and their effects on electronic devices. Since the work involves the use of an FPGA as the main application device, its architecture will be presented. Next, a section is dedicated to the CubeSat standard and the nano-satellites classifications. This section also presents a review of the communication protocols used in small satellites focusing on their structures and capabilities.

2.1 RADIATION

When exposed to a radiation space environment, the effects caused on electronic devices are one of the main concern for all type of satellites missions, from the Low Earth Orbit (LEO) to deep space (VELAZCO; FOUILLAT; REIS, 2007; NICOLAIDIS, 2011; FLEETWOOD; WINOKUR; DODD, 2000).

2.1.1 Space Radiation Environment

The space environment is classified into four categories accordingly with the origins: radiation belts, solar flares, solar wind, and galactic cosmic rays (VELAZCO; FOUILLAT; REIS, 2007).

The Earth radiations belts, known as Van Allen Belts, are composed of trapped electrons and protons and are divided into two zones. The inner belts contain electrons with energy between 1 and 10 MeV and high-energy protons (>100 MeV). The outer belt is mainly composed of high-energy electrons (VELAZCO; FOUILLAT; REIS, 2007; DYER, 2002). Figure 1 presents the radiation belts fluxes with respect to the Earth radius.

There is a region located in the South Atlantic area, where a high flux of energetic protons is present. This anomaly is caused because of displacement between the magnetic and geographic axes. This region is known under the name of South Atlantic Anomaly (SAA) and a spacecraft is exposed to its radiation effects at altitudes about 1000 km (PETERSEN, 2011; DYER, 2002).

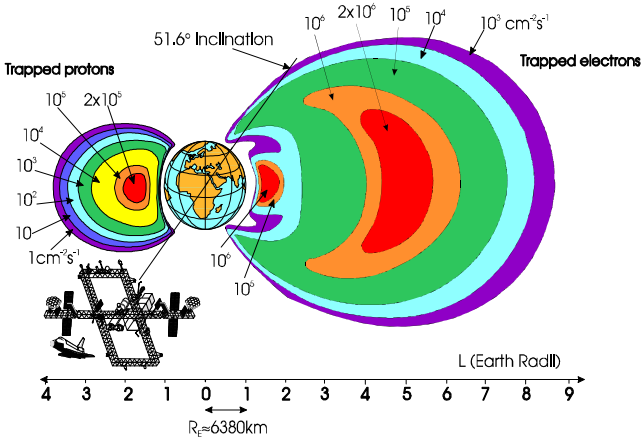
During the years close to the solar maximum, the solar flares

are a sporadic source of radiation and are defined by two types of events. The first one is the Coronal Mass Ejection (CME) that could remain for several days and emits high-energy protons with energies of hundreds MeV. The second type is the emission of heavy-ions with a energy range starting from several tens of MeV up to hundreds GeV per nucleon (VELAZCO; FOUILLAT; REIS, 2007; DYER, 2002; PETERSEN, 2011).

The solar wind particles have not significant energy when compared with solar flares and in the Geostationary Orbit (GEO) leads to electrons with energies around 2 keV and ions at 10 keV, while in the LEO the energies are even lower and incapable to inducing significant charge deposition (VELAZCO; FOUILLAT; REIS, 2007; PETERSEN, 2011).

Cosmic Galactic Rays (CGRs) are composed of high energetic heavy-ions (1%), protons (83%), helium nuclei (13%) and electrons (3%). The ions' energy is very high, with the most energetic ion ever detected having an energy of $3x10^{20}$ eV. Therefore, the primary concern in space applications is the effect of ionizing particles in the energy range of 1-20 GeV/nucleon (VELAZCO; FOUILLAT; REIS, 2007).

Figure 1 – Van Allen belts with respect to the flux and Earth radius. In the left is shown the proton contribution and the right, the electron contribution.



Source: (DYER, 2002).

2.1.2 Radiation Effects

The space radiation environment affects microelectronics devices with cumulative effects that are characterized by the Total Ionizing Dose (TID) and Displacement Damage (DD), but also by instantaneous and temporary effects defined as Single Event Effects (SEEs), which are due to the penetration of a single ionizing particle that causes issues in sensitive nodes of the circuits (PETERSEN, 2011; DYER, 2002).

2.1.2.1 Cumulative Effects

The TID is a long-term failure mechanism and is defined as the energy deposited per mass unit of material and that is generally expressed by the unit rad (in the SI) and that degrades the function of the device until its operation is definitely compromised (DYER, 2002; DOWD, 2003). The cumulative dose can cause shift in the transistor threshold voltage, increase of the leakage currents (power consumption), and also the degradation of the gain in bipolar devices (DYER, 2002).

According to Schwank (1994) and Johnston (2010), the DD effect is caused by the collision between high-energy protons and one atom generating a recoil of the same from its lattice site. When the transfer energy of the particle is high enough, the hitted atom can be freed from its interstitial site. The minimum energy required for this is called the displacement threshold energy. The accumulation of this effect can create a large fault cluster. Dyer (2002) cites examples of damage caused by this effect, as a reduction in bipolar transistor gain, reduction of the efficiency in solar cells, light emitting diodes and photodetectors, charge transfer inefficiency in charge coupled devices and resolution degradation in solid-state detectors.

2.1.2.2 Single Event Effects

A single ionizing particle (heavy ion, electrons, and protons) strike may induce a Single Event Effect, that occurs instantaneously and is due to a deposed charge that is collected by transistor electrodes of the device (NICOLAIDIS, 2011; VELAZCO; FOUILLAT; REIS, 2007; PETERSEN, 2011).

The SSEs are classified as soft-errors, since they cause data cor-

ruption that can be mitigated for example by a power cycle, a reset, or other erase operations. However, in some cases, they may lead to hard-errors that cause permanent device failure. The SSEs are divided into the following categories (NICOLAIDIS, 2011; VELAZCO; FOUILLAT; REIS, 2007; PETERSEN, 2011):

- Single-bit Upset (SBU) [Soft-error]: a transient event that causes a bit-flip (upset) in a memory cell or a latch;
- Multiple-Bit Upset (MBU) [Soft-error]: transient event caused by a single ionizing particle that generates two or more bit-flips in the same word;
- Multiple-Cell Upset (MCU) [Soft-error]: transient event caused by a single ionizing particle that generates two or more bit-flips in the cell array;
- Single Event Transient (SET) [Soft-error]: the collected free carriers generated by an ionizing particle may generate a current or voltage transient that may lead to an error, especially in combinational logic circuits;
- Single Event Functional Interruption (SEFI) [Soft-error]: this event is due a perturbation in control signals (e.g. registers, clock signals, reset signals) and cause a loss of functionality;
- Single Event Latch-up (SEL) [Hard/Soft-error]: when a particle hit triggers a parasitic thyristor that involves high-current consumption. This event requires a power cycle and may cause permanent damage due to thermal dissipation of large current;
- Single Event Gate Rupture (SEGR) [Hard-error]: this event occurs when a particle strike is capable of breaking the gate dielectric of a MOS transistor;
- Single Event Burnout (SEB) [Hard-error]: occurs when a strike causes a destructive burnout due to high-current and Joule effects.

2.2 RECONFIGURABLE HARDWARE

The reconfigurable hardware combines the flexibility of a software implementation with the hardware performance, which is potentially higher than a microcontroller solution, since the microcontroller

must read each instruction from memory, interpret it and then execute the instruction (COMPTON; HAUCK, 2002). This is confirmed by Bezerra e Gough (2000) and Garcia et al. (2006), that show how the hardware-based implementations avoid the software overhead of fetch/decode/execute and use the resources to increase the parallelism.

The FPGA is an integrated circuit composed of an array of Configurable Logic Blocks (CLBs) and configurable interconnections that brings the reconfigurable hardware concept since their capability to be reprogrammed in a matter of microseconds. This reconfigurable logic blocks can implement combinational and sequential logic circuits. This implementation uses a Hardware Description Language (HDL) to define functionality in Register-Transfer Level (RTL) that will be synthesized into a netlist with the connectivity between the elements of the circuit (WAIN et al., 2006; FAROOQ; MARRAKCHI; MEHREZ, 2012).

2.2.1 FPGA Architecture

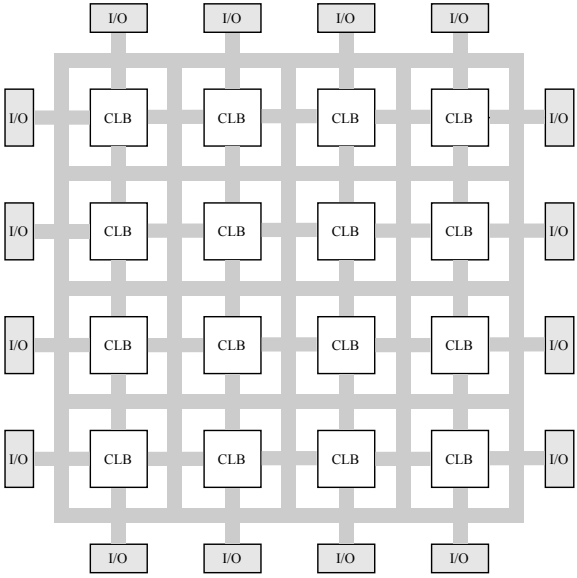
FPGA are devices composed of an array of circuitry's blocks that are generally comprised of three main components: configurable logic blocks (CLBs); configurable routing connections; I/O (inputs/outputs) blocks to provide external access to the chip. As presented in Figure 2, the logic blocks are distributed in an array format, the I/O blocks are arranged in the periphery of the grid, and the routing elements interconnect all these blocks (BEZERRA; LETTNIN, 2014; FAROOQ; MARRAKCHI; MEHREZ, 2012).

According to Farooq, Marrakchi e Mehrez (2012), the SRAM technology is the most used by commercial vendors since it allows easy re-programmability and use CMOS standard processes. For the SRAM-based FPGAs, the memory cells are used to store data in the logic blocks and to program the multiplexers that are used in the interconnections elements.

Most FPGAs use Look-up Tables (LUTs) in the CLB implementation. The LUTs implement any n-input truth table and are complemented with the use of a Flip-Flop, Figure 3 presents this generic structure in a SRAM-based FPGA (BEZERRA; LETTNIN, 2014; FAROOQ; MARRAKCHI; MEHREZ, 2012).

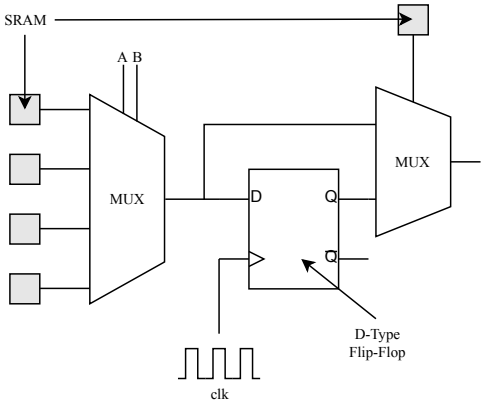
Wires and programmable switches composing the interconnections between the CLBs and I/Os and the basic FPGA architectures are defined by island-style and hierarchical interconnections schemes (FAROOQ; MARRAKCHI; MEHREZ, 2012).

Figure 2 – A typical FPGA internal components.



Source: based on (BEZERRA; LETTNIN, 2014; FAROOQ; MARRAKCHI; MEHREZ, 2012).

Figure 3 – An FPGA basic logic element.



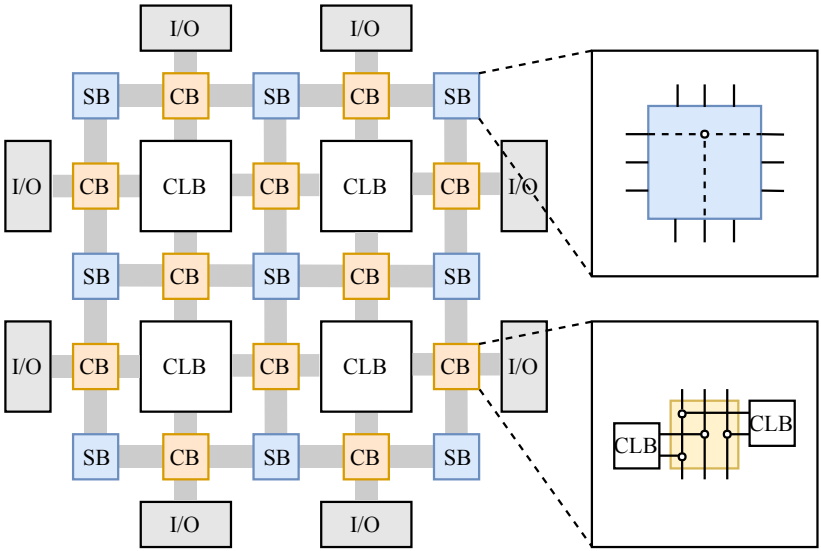
Source: based on (BEZERRA; LETTNIN, 2014; FAROOQ; MARRAKCHI; MEHREZ, 2012).

Island-style organizes wiring segments that are connected through switches and connection blocks, however, as a drawback, this scheme uses between 80-90% of the total area of the chip, and logical blocks occupies only 10-20% (BETZ; ROSE; MARQUARDT, 1999). Figure 4 presents an architecture overview of the Island-style scheme, where the switches blocks are referred as 'SB', and the connection blocks 'CB' (FAROOQ; MARRAKCHI; MEHREZ, 2012).

Hierarchical routing architecture remains a tree-like structure, where the logic blocks are positioned as leaves in a tree, and there are several levels of interconnections. In the example presented in Figure 5, the architecture has four wire levels. This scheme uses two different kinds of switches (FAROOQ; MARRAKCHI; MEHREZ, 2012):

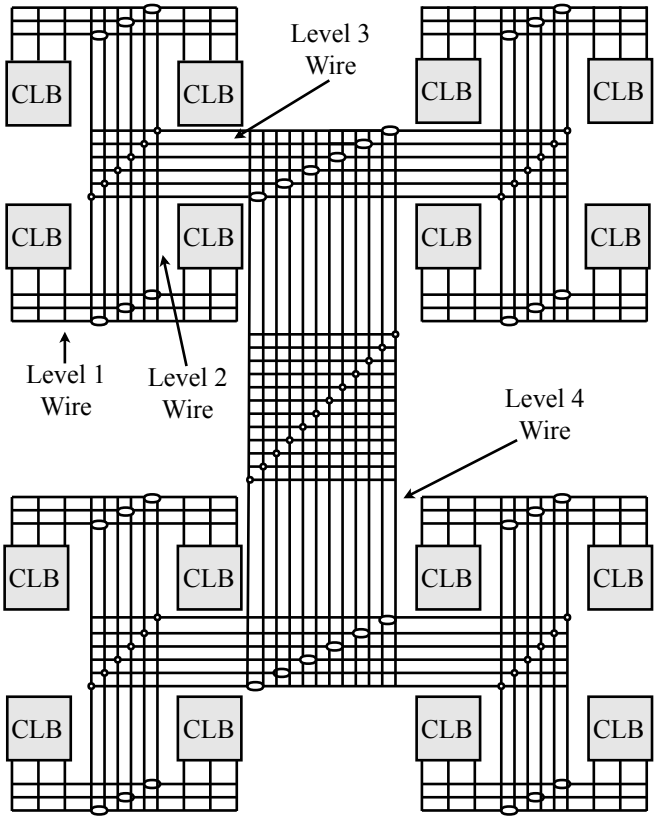
- Non-compressing: The number of tracks in the upper wire level is equal to the sum of the number of tracks in the lower wire level.
- Compressing: the number of tracks in the upper wire level is equal to the number of tracks in the lower wire level.

Figure 4 – Overview of the Island-style interconnection scheme.



Source: based on (FAROOQ; MARRAKCHI; MEHREZ, 2012).

Figure 5 – Hierarchical routing architecture.



Source: based on (FAROOQ; MARRAKCHI; MEHREZ, 2012).

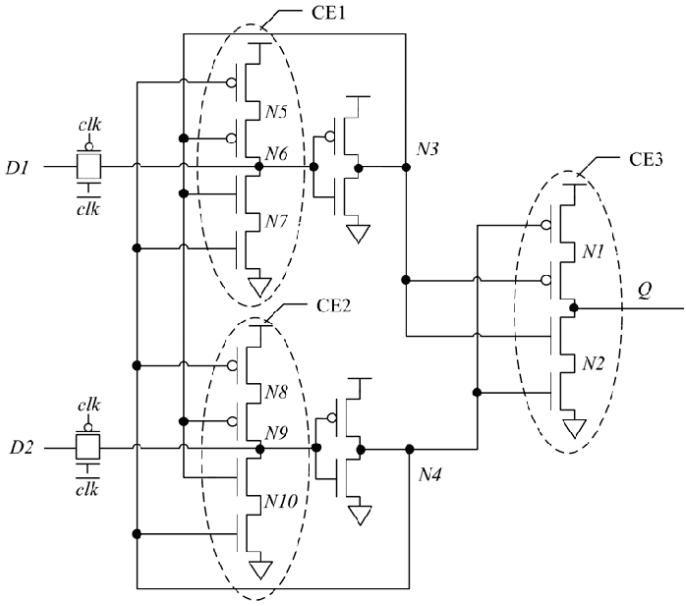
2.2.2 FPGA Radiation Hardened Architecture

SRAM-based FPGAs are based on CMOS technology and they are susceptible to SEE (YUE et al., 2017). Several techniques can be introduced in these devices to increase their reliability and avoid the radiation effects. Besides the techniques used in the application upper level, Radiation-Hardened By Design (RHBD) devices mitigate these effects at hardware level (BATTEZZATI et al., 2009).

The work from Huang e Liang (2008) proposes an RHBD latch for sub-micron technologies that is immune to SEUs and SETs. According to the authors, the use of a dual interlocked scheme with internal feedback lines eliminate vulnerable nodes. Figure 6 present the architecture of the proposed latch.

Besides the technique for latch architectures presented by Huang e Liang (2008), we can found several different ways to face radiation-induced effects. Examples are presented in the works from Chen e Orailoglu (2007), Naseer e Draper (2006).

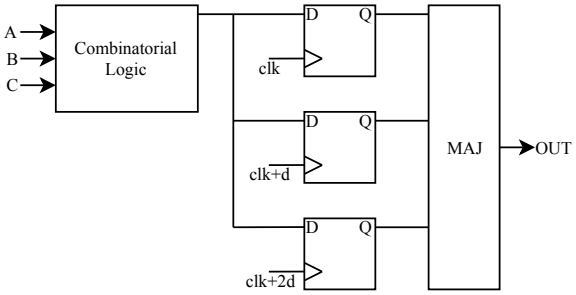
Figure 6 – A Radiation-Hardened By Design latch.



Source: based on (HUANG; LIANG, 2008).

Redundancy techniques can be divided into Full-Time Redundancy (that mitigates SET) and Full Hardware Redundancy (that mitigates SEU). The Full-Time Redundancy gets the output of combinatorial logic in three different moments that are defined by a small delay, and a majority voter (MAJ) is responsible for defining the correct output value (ZHANG; LIE, 2011; CALIN; NICOLAIDIS; VELAZCO, 1996). Figure 7 presents Full-Time Redundancy scheme.

Figure 7 – Full Time Redundancy scheme.



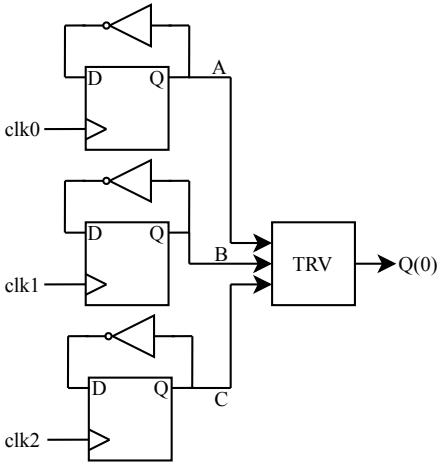
Source: adapted from (ZHANG; LIE, 2011).

According to Zhang e Lie (2011), Full Hardware Redundancy can be achieved by a Triple Modular Redundancy TMR scheme, where the logic is triplicated and a majority voter defines the correct output value. To further improve reliability, the scheme presented in Figure 8 implement the use of a triplicated voter and Flip Flops with feedback (ZHANG; LIE, 2011).

The application of EDACs is an efficient solution to protect memories against SEUs, with coding techniques applied to high capacity memory and periodically 'scrubbing' of the entire array (ZHANG; LIE, 2011; WANG et al., 1999; CALIN; NICOLAIDIS; VELAZCO, 1996).

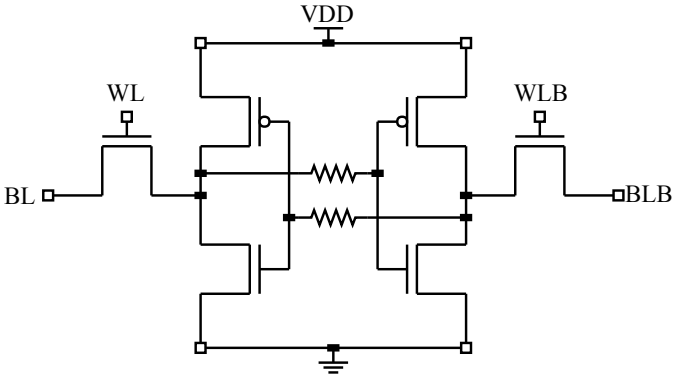
At cell level different hardening techniques can be applied leading, for example, to the Hardened Memory Cells, Canaris Hardened Cells, DICE Hardened Memory Cell, and NASA Hardened Memory Cell (I and II). Focusing on commonly used techniques, the Hardened Memory Cells use two resistors between the inverters of an SRAM cell, and when a particle strikes one of the nodes these resistors will delay the spike propagation and prevent the bit-flip (ZHANG; LIE, 2011). Figure 9 presents this structure.

Figure 8 – Full Hardware Redundancy, a TMR One-Bit Counter.



Source: adapted from (ZHANG; LIE, 2011).

Figure 9 – Resistivity hardened CMOS SRAM cell design.

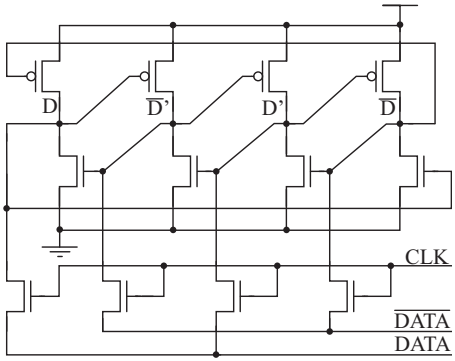


Source: adapted from (ZHANG; LIE, 2011).

According to Danilov et al. (2018), the DICE structure applies the idea that a charged particle has to achieve two or more sensitive nodes to generated a SEU. This principle was described by Calin, Nicolaidis e Velazco (1996), in which the authors refer that the SEU occurrence probability can be reduced if the sensitive node pairs are spaced

on cell's layout.

Figure 10 – DICE Cell.



Source: (DANILOV et al., 2018).

2.2.3 BRAVE FPGA

This work leads to the use of the re-programmable European Radiation-Hardened FPGA developed by NanoXplore that is a fabless semiconductor company headquartered in France. This FPGA targets space applications and it is the result of a project between the NanoXplore (design, verify and FPGA tools) and the ST Microelectronics (foundry, IP provider and packing evaluation and qualification), involving the AirBus D&S and Thales Alenia as key alpha customers.

The NG-MEDIUM is the first FPGA of the Big Re-programmable Array for Versatile Environments (BRAVE) FPGA family and is radiation hardening by design in configuration memories and registers.

Register files and DPRAM (Dual-Port RAM) use EDAC to be able to detect and correct SEUs and detect MBUs. The configuration memory, user registers, SRAM and Flip-Flops use the DICE (Dual Interlocked Storage Cell) scheme that are more resilient to radiation than the conventional 6 transistor scheme. The clock tree applies DMR (Dual Modular Redundancy) strategy in the clock buffer and the distributed matrix system. Others logic cells apply TMR.

Besides this techniques, an embedded Configuration Memory Integrity Check (CMIC) scheme performs an automatic verification and repair in the configuration memory, this process is done at run time, and ECC protects the reference memory. As a result of the implemented

techniques, the BRAVE FPGA has SEL immunity up to a LET of 60 MeV .cm^2/mg, and can endure a TID up to 100 krads.

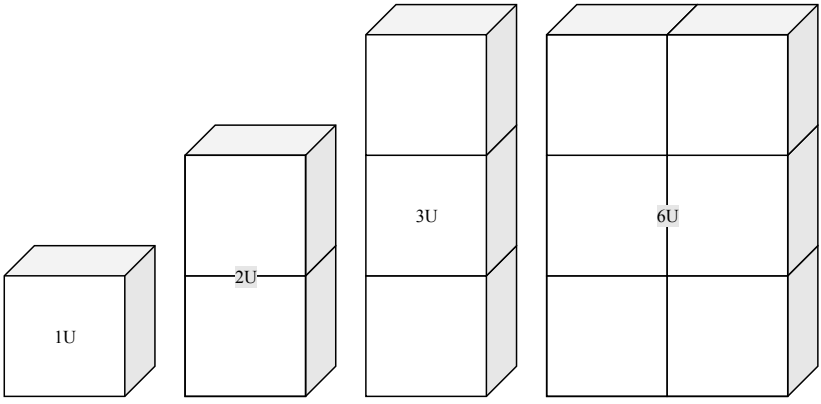
2.3 SMALL SATELLITES - CUBESATS

With the mission named OPAL (Orbiting Picosatelitte Automated Launcher), the launch of a small satellite developed by a student group at Stanford University created a new concept in the architecture of these devices. The mission was based on a microsatellite of 25 kg responsible for launching 6 picosatellites in the space (ENGBERG; OTA; SUCHMAN, 1995).

With the acquired experiences of the OPAL mission and the necessity to have picosatellites that have the requirements to a default launcher, the California Polytechnic State University and Stanford University started a new partnership project to create a standard picosatellite, the result of this project was the CubeSat[2] (HEIDT et al., 2000; PUIG-SUARI; TURNER; AHLGREN, 2001).

The CubeSat specification defines that a standard 1U is a cube of nearly 10x10x10 cm^3 with mass up to 1.33 kg and this unit could be combined forming a large spacecraft. CubeSats are classified according to their mass and volume and Figure 11 and Table 1 presents a concept of this classification (JOHNSTON, 2010; NASA CUBESAT LAUNCH INITIATIVE, 2017; THE CUBESAT PROGRAM, 2014).

Figure 11 – CubeSat classification according their volume.



Source: based on (JOHNSTON, 2010; NASA CUBESAT LAUNCH INITIATIVE, 2017; THE CUBESAT PROGRAM, 2014).

Table 1 – Satelites classification according their mass.

Classification	Mass (kg)
Pico	< 1
Nano	$1 \sim 10$
Micro	$10 \sim 100$
Mini	> 100

Source: adapted from (JOHNSTON, 2010).

According to Arnold, Nuzzaci e Gordon-Ross (2012) CubeSats usually are placed in Low Earth Orbit (LEO), which reach until 2.000 km from the surface. Orbits are classified related with their altitude, then, above LEO is Medium Earth Orbit (MEO) extended outward to 35.786 km, and right after is the High Earth Orbit (HEO) for altitudes higher the MEO limit.

Also, orbit patterns are related to their inclination (satellite orbit angle with respect to the equator) and the kind of orbit is directly related with the CubeSat power generation, since, for example, a CubeSat that orbits in a Sun-Synchronous orbit received a near-constant sun illumination and the power system is capable of providing a better power supply to the subsystems. Besides that, the size and weight restrictions of a CubeSat limits the use of batteries and external solar panels and thus the power available by the system (ARNOLD; NUZZACI; GORDON-ROSS, 2012).

2.4 COMMUNICATION PROTOCOLS FOR SMALL SATELLITES

2.4.1 AX.25 Protocol

The AX.25 protocol was developed by the radio amateur community since there was a need to define a protocol capable to accept and send data in a reliable way over a communication link between two terminals (BEECH; NIELSEN; TAYLOR, 1998). According to studies carried out in Muri e Mcnair (2012) and Klofas, Anderson e Leveque (2008) the AX.25 protocol has been chosen as principal choice for the small satellites' scopes.

The frame implemented in the AX.25 protocol is divided into three types: Information frame; Supervisory frame; and Unnumbered frame. The frames are composed of fields that identify the start and end of the frame ("Flag" field), the address of the source and destination

("Address" field), the frame type ("Control" field), the protocol identifier ("PID" field), the user data ("Info" field) and the frame check sequence that is calculated accordingly with ISO 3309 (HDLC) Recommendations and it is used to check if the received data is corrupted during the transmission ("FSC" field) (BEECH; NIELSEN; TAYLOR, 1998). Figures 12 and 13 present the two different constructions of the frame.

Figure 12 – Unnumbered and Supervisory frame construction.

First Bit Sent				
Flag	Address	Control	FCS	Flag
0111 1110	112/560 Bits	8 Bits	16 Bits	0111 1110

Source: (BEECH; NIELSEN; TAYLOR, 1998).

Figure 13 – Infomation frame construction.

First Bit Sent						
Flag	Address	Control	PID	Info.	FCS	Flag
0111 1110	112/560 Bits	8 Bits	8 Bits	n*8 Bits	16 Bits	0111 1110

Source: (BEECH; NIELSEN; TAYLOR, 1998).

This protocol was not conceived to embed telemetry and telecommands for space applications. Futher limitations are related to the fact that this protocol does not implement a Forward Error Correction (FEC) scheme, and it is unable of correcting errors in the received data (ZEIGER; SCHMIDT; SCHILLING, 2006).

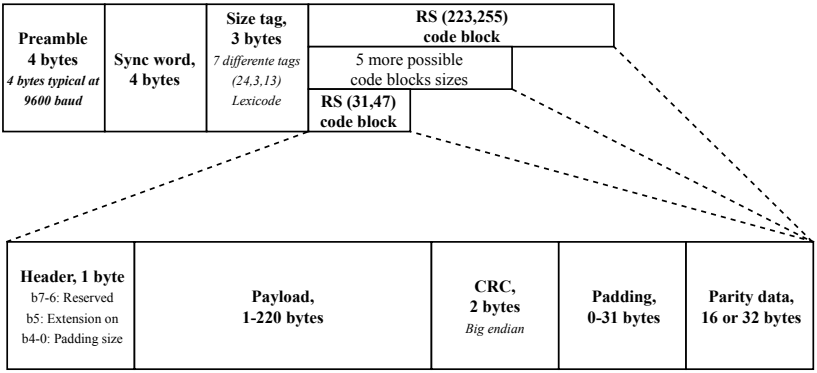
2.4.2 NGHam Protocol

Next Generation Ham (NGHam) is a protocol that is based on the idea of the AX.25 protocol, however, improving the reliability using a FEC on the link layer. The FEC implements a Reed-Solomon (RS) algorithm with 16 or 32 check symbols (SKAGMO, 2014; KLEINSCHRODT et al., 2017; BIRKELAND; LØFALDLI, 2016).

Figure 14 presents the structure of a NGHam frame.

Preamble field is used to assure a timing requirement for proper interpretation since it is sending during the ramping up to the radio power amplifier and the receiver Automatic Control Gain (ACG) is

Figure 14 – Structure of a NGHam frame.



Source: (SKAGMO, 2014).

stabilizing. Also, a Sync Word is used for packet synchronization. The Size tag field identifies one of the seven possibilities of code block sizes. In the code block, the Header field is used to identify the padding size. Furthermore, the CRC field and the Parity data are introduced for error detection and correction implementations (SKAGMO, 2014).

2.4.3 CCSDS Recommendations

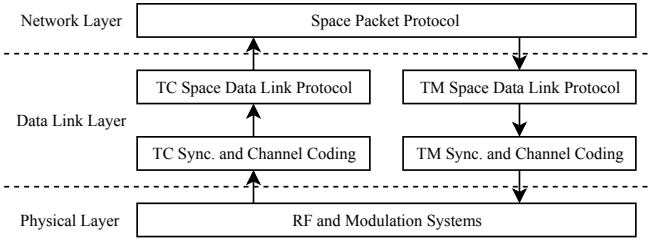
Historically, in the 1980s, CCSDS proposed an international standard for sending telemetry data using a variable-length data unit called Source Packet. This standard defined the continuous transmission of Source Packets generated from the instruments and subsystems of a spacecraft through Transfer Frames. Following this concept, another international standard was developed targeting the transmission of commands to a spacecraft with a data unit named TC Packet. In this case, the transmission is sporadic and uses a variable length Transfer Frame. Since the necessity to meet the requirements of the Advanced Orbiting Systems (e.g., the International Space Station) a third standard known as AOS was produced. The main concern in this standard is to add services for online transmission data (e.g., audio and video data) (CCSDS, 2014).

The above mentioned standards were restructured with the objective to define the protocols in a more structured way, generating: *Space Packet Protocol*; *TM, TC, and AOS Space Data Link Protocols*;

and *TM and TC Synchronization and Channel Coding*.

The CCSDS defines several protocols, but this section only describes the recommendations that are within the scope of the work. These recommendations are distributed following a stack of protocols that are associated with layers. Figure 15 presents the protocols with their layer and their relationships.

Figure 15 – CCSDS protocols defined by layers.

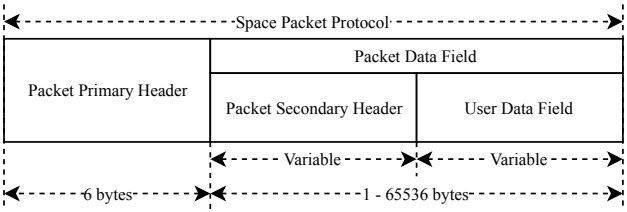


Source: Adapted from (CCSDS, 2014).

2.4.3.1 Space Packet Protocol

The Space Packet Protocol defines a way of transferring application data over a network in a ground-to-space or space-to-ground link. This protocol is structured following the Figure 16 (CCSDS, 2003).

Figure 16 – Space Packet Protocol structure.

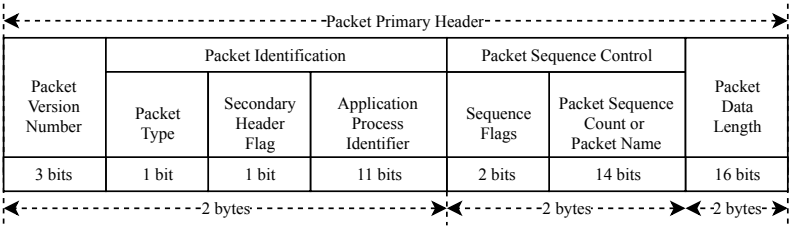


Source: Adapted from (CCSDS, 2003).

The Packet Primary Header is mandatory and consists of four fields, contiguously positioned. This header starts by defining which is the version of the packet that is related to the recommendation (e.g., CCSDS 133.0-B-1 defines version 1). The Packet Identification field specifies if the packet is telemetry or telecommand, and indicates the

presence or absence of the Packet Secondary Header; and the identification of the application process that has to receive or is sending this data. Packet Sequence Control defines if the packet is segmented and provides the sequential binary count of each Space Packet generated by the application process (mandatory for telemetry packets) or the packet name (that could be used in for telecommand packets. Packet Data Length contains a 16-bit count that represents the total number of bytes present in the Packet Data Field (CCSDS, 2003). Figure 17 presents its format.

Figure 17 – Space Packet Protocol primary header structure.



Source: Adapted from (CCSDS, 2003).

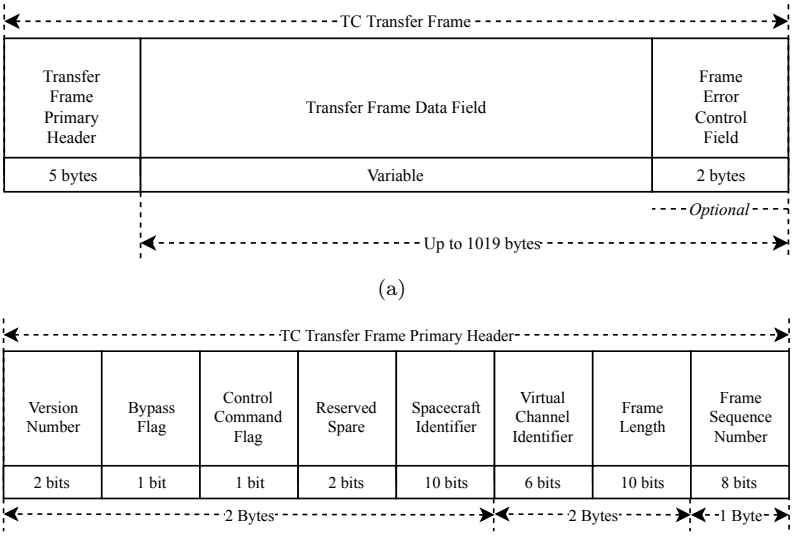
2.4.3.2 TM and TC Space Data Link Protocols

Following the purpose to specify a link protocol for telecommand and telemetry to be used over ground-to-space or space-to-space communications, the CCSDS defines the CCSDS 132.0-B-2 TM Space Data Link Protocol (CCSDS, 2015a) and CCSDS 232.0-B-3 TC Space Data Link Protocol (CCSDS, 2015b), that defines the services, units and procedures of these protocols.

CCSDS (2015b) defines a protocol data unit named "TC Transfer Frame" that is structured as presented in Figure 18, while CCSDS (2015a) defines the "TM Transfer Frame", as shown in Figure 19. The TC and TM Space Data Link Protocol have as principal functions the segmentation and blocking of service data units and transmission control of service data units. This recommendation also provides an automatic retransmission mechanism procedure to ensure reliability on the received telecommands defined as Communications Operation Procedure (COP) that is specified in (CCSDS, 2010).

The COP specification implements the ARQ-GBN[1]. This mechanism to manage the retransmission, this procedure works as a closed-loop. This loop is composed of the Frame Acceptance and Reporting Mechanism (FARM) (which is used in the receiver) and the Frame Operational Procedure (FOP) (which is applied in the sender). The FOP sends TC Frames to FARM, and the FARM provides a status report through the Communications Link Control Word (CLCW). This procedure uses sequence numbers 'N' within the frames, and variables 'V' that store these values. The Figure 20 presents this loop.

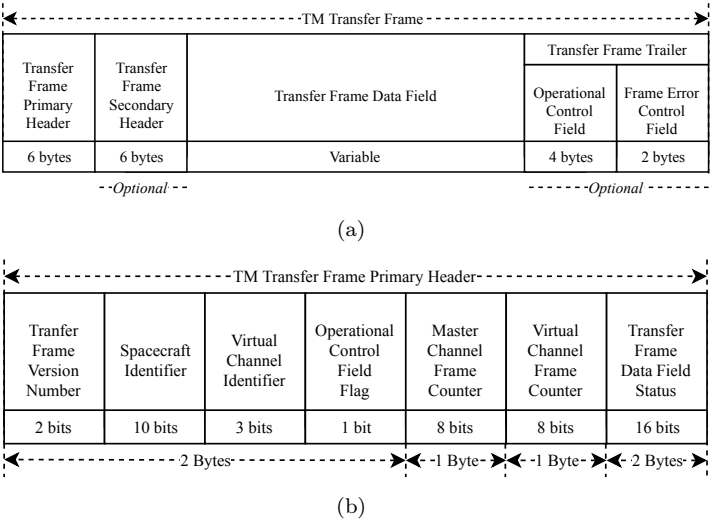
Figure 18 – TC Transfer Frame: (a) structural components; (b) primary header structure.



Source: Adapted from (CCSDS, 2015b).

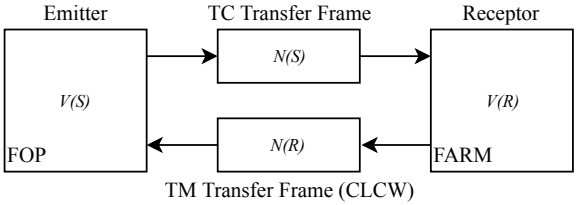
[1]ARQ-GBN (Automatic Repeat Query - Go-Back-N) is a retransmission mechanism used as a method of error control, and more information can be reached in the reference (Shu Lin; COSTELLO; MILLER, 1984)

Figure 19 – TM Transfer Frame: (a) structural components; (b) primary header structure.



Source: Adapted from (CCSDS, 2015b).

Figure 20 – COP representation constituting of variables, frame and report values.



Source: Adapted from (CCSDS, 2010)

In Figure 20, N(S) represents transmitted frame sequence number, N(R) is the next frame sequence number that has to be received, V(S) stores the sequence number, and V(R) stores the frame sequence number that has to be received. To validate the transmission the receptor has to satisfy: $N(S) = V(R)$.

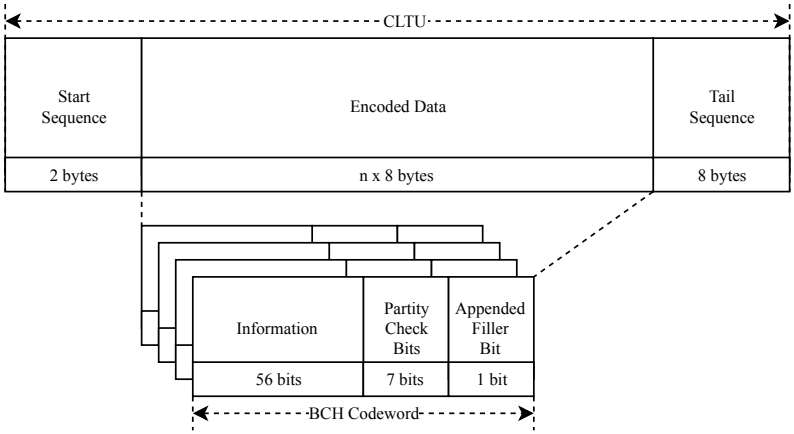
2.4.3.3 TM and TC Synchronization and Channel Coding

The recommendations CCSDS (2017a) and CCSDS (2017b) describes the synchronization and channel coding for telemetry and telecommand respectively and provide four functions for data transfer over a space link: error-control coding; synchronization; pseudo-randomizing (optional); and repeated transmissions (optional).

In the scope of the telecommand flow, the CCSDS recommendation 231.0-B-3 (CCSDS, 2017b) defines two error-control coding schemes, the first is the Bose–Chaudhuri–Hocquenghem (BCH) and the second is the Low-Density Parity-Check (LDPC), also, specifies these methods using a data unit called Communications Link Transmission Unit (CLTU) (CCSDS, 2017b).

The CLTU structure is composed of a start sequence, followed by the encoded data and a tail sequence. The encoded data consists of Transfer Frames according to CCSDS (2015b), and the tail sequence defines the end of the packet that is constructed to be a non-correctable pattern of the coding scheme. Figures 21 and 22 present the CLTU components when using, respectively, BCH and LDPC schemes.

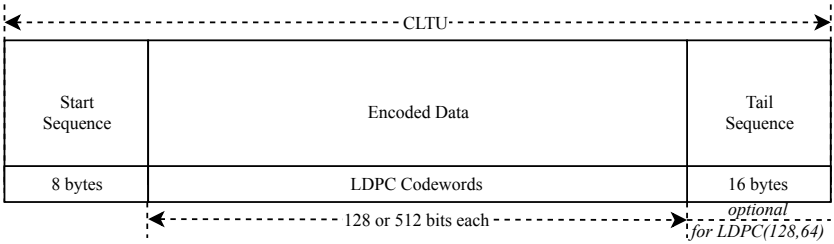
Figure 21 – CLTU data unit with the structure of a BCH codeword.



Source: Adapted from (CCSDS, 2017b)

CCSDS (2017a) defines the use of four different coding schemes: Convolutional; Reed-Solomon; Turbo Coding; and LDPC. These algorithms methods are applied in the data unit called Channel Access Data

Figure 22 – CLTU data unit with the structure of a LDPC codeword.



Source: Adapted from (CCSDS, 2017b)

Unit (CADU). The Concatenated coding consists of a combination of a Reed-Solomon (outer code) with the Convolutional (inner code). Table 2 presents the CADU contents related to the coding scheme, and the containing data may or may not be randomized.

Table 2 – CADU with different coding schemes.

Applied Coding Scheme	**CADU**
Convolutional Coding	ASM and the Transfer Frame
Reed-Solomon Coding	ASM and Reed-Solomon codeblock
Concatenated Coding	ASM and Reed-Solomon codeblock
Turbo Coding	ASM and Turbo codeword
LDPC Coding	ASM and LDPC codeword

Source: Based on (CCSDS, 2017a).

2.5 PACKET UTILIZATION STANDARD

The Packet Utilization Standard (PUS) (ECSS-E-ST-70-41C) defines the application-level interface using the protocol established in the CCSDS Space Packet Protocol (CCSDS 133.0-B-1) targeting the utilization of telecommand and telemetry packets for remote monitoring and control of the subsystems (ECSS, 2016). For this, the standard defines a set of services with capabilities and structured data unit, which defines the content data of the Secondary Header and User Data fields presented in Figure 17.

The Secondary Header has two different formats, one is related to a telecommand, and the other is related to telemetry. Figure 23(a) presents the telecommand structure and the Figure 23(b) presents the

telemetry one. When this secondary header is presented in the Space Packet, these fields shall follow, without a gap, the Packet Primary Header(ECSS, 2016).

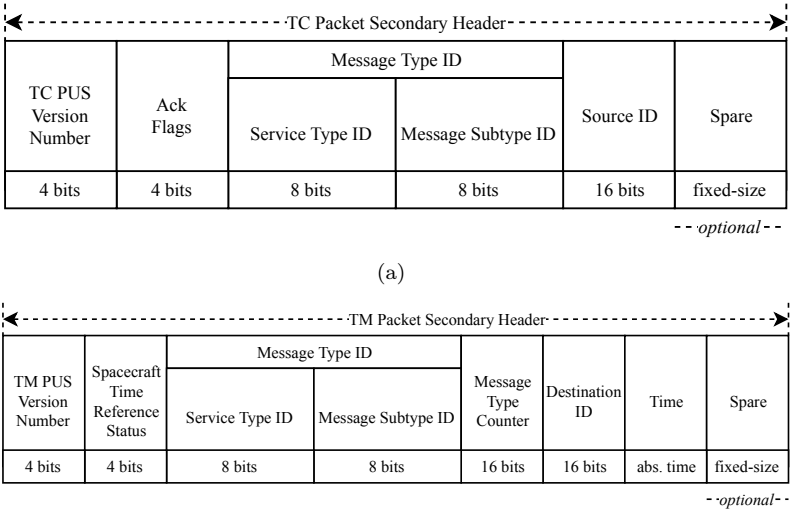
The fields "PUS Version Number", "Message Type ID", and "Spare", are presented in both structures. The fields inform the applied ECSS version, identifies the type of the message that will define the structure of the user data field, and, when needed, the usage of the Spare field aiming at constraining the length to an integral number of words (ECSS, 2016).

Following the telecommand structure, "Ack Flags" are used to signalize which kind of report is required by the telecommand that is related with the service applied. These reports are defined by the Service 1 - Request Verification and possibilities four different types of report. "Source ID" field corresponds to the application process that hosts the "Service Type" (ECSS, 2016).

Related to telemetry structure, "Spacecraft Time Reference Status" defines a reference time report using CUC (CCSDS Unsegmented) or CDS (CCSDS Day Segmented) format, or, if it is not supported, has to be set as '0000'. A "Message Type Counter" related with the "Message Type" has to be maintained. "Time" reports the absolute time according to their reference, and the use of the "Spare" field is optional (ECSS, 2016).

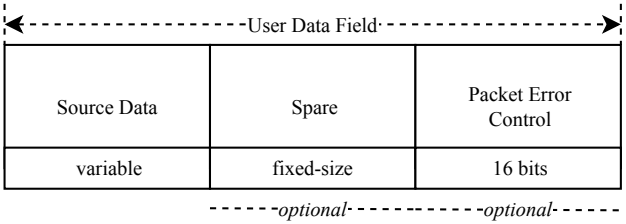
User data field structure is composed of a variable field of source data that could be complemented by a spare field. Also, the "Packet Error Control" is optional and applies a CRC standard 16-bits checksum algorithm known as CRC-16-CCITT that used the polynomial generator $G(x) = x^{16} + x^{12} + x^5 + 1$ (ECSS, 2016). Figure 24 presents its structure.

Figure 23 – Space Packet Protocol secondary header structure, (a) telecommand and (b) telemetry.



Source: Adapted from (CCSDS, 2003) and (ECSS, 2016).

Figure 24 – Space Packet Protocol user data structure.

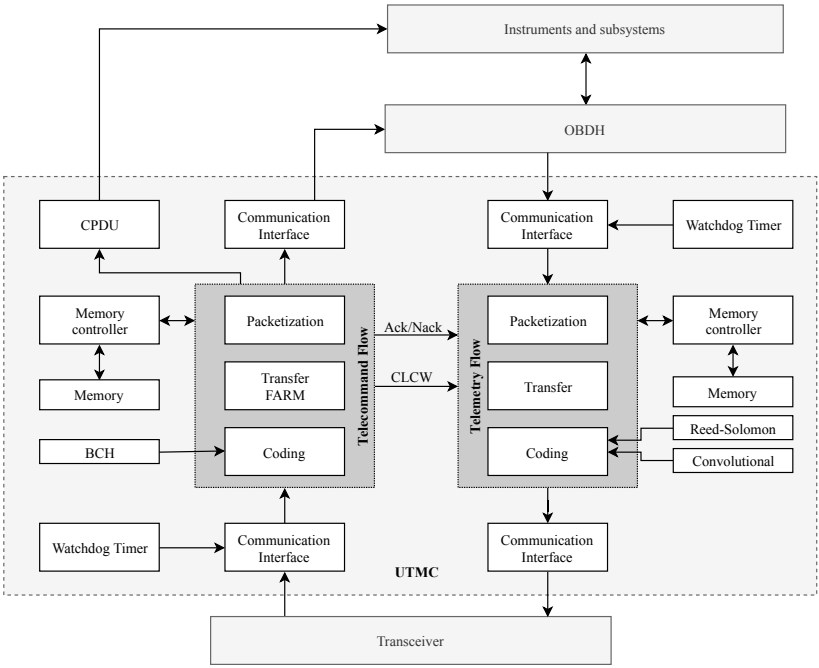


Source: Adapted from (ECSS, 2016).

2.6 UNITY OF TELEMETRY AND TELECOMMAND

The GSE group developed in previous works a UTMC to be applied in the on-board computer of a Brazilian satellite. The main function of this implementation is to handle the TM and TC flow according to CCSDS recommendations and use the Data Link Layer presented in Section 2.4.3. The UTMC block is divided into two segments dedicated to telemetry data processing and telecommands handling. Figure 25 presents the UTMC architecture.

Figure 25 – UTMC top level diagram.



Source: adapted from (BEZERRA et al., 2010; BEZERRA; AZEVEDO; SILVA, 2011).

In the TC flow, when a TC is delivered to UTMC from the communication interface, the CLTU is decoding using BCH algorithms and performing the error detection and correction. When a Direct TC (DTC) is detected, the UTMC delivers it to a CPDU layer that interprets and generates a strictly controlled configurable time pulse

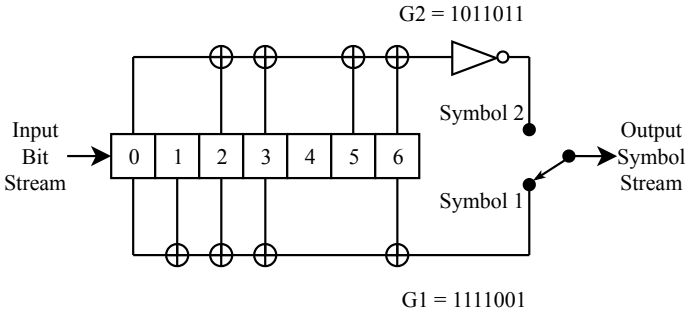
in one of the outputs available. Also, when a Routed TC (RTC) is detected, the UTMC delivers the Space Packet data to the OBDH system (BEZERRA et al., 2010; BEZERRA; AZEVEDO; SILVA, 2011).

The BCH decoder is implemented following the CCSDS recommendations which describes a BCH(63, 56) code using a generator matrix *G(56, 63)* and a parity matrix *H(63, 7)*. The generator matrix G is composed by two submatrices, the identity matrix $I_1(56, 56)$ and the matrix $G_1(56, 7)$ that is composed by the syndrome values generated from the polynomial $g(x) = x^7 + x^6 + x^2 + 1$. The $H^T(63, 7)$ is formed by G_1 and I_1 matrices. For the decoding process, the identification of the syndrome value is given by the multiplication of the received message (BCH codeword) by the parity matrix H^T. A syndrome value equal zero means a message received without errors, otherwise, the syndrome value is searched between the columns 57 and 63 of the matrix G, and if this value is founded, a XOR operation between the first 56 bits of the codeword and the 56 bits of the G corresponding line is performed, resulting in the original value of the message (BEZERRA et al., 2010; BEZERRA; AZEVEDO; SILVA, 2011).

Following the TM flow, the data generated by the sensors are processed in the OBDH, which creates a telemetry packet and sends it to the UTMC, then a Telemetry Transfer Frame (TMTF) is generated by the Transfer Layer. The TMTF can be coded with either of Reed-Solomon (RS), Convolutional, or the combination of Reed-Solomon + Convolutional algorithms and results in the CADU that is dispatched by communication interface (BEZERRA et al., 2010; BEZERRA; AZEVEDO; SILVA, 2011).

The implementation of the RS(255, 223) code is based on a Linear Feedback Shift Register (LFSR) that is used for mapping symbols in terms of its basic elements. However, to reduce the area consumption for the add and multiply operations realized by the LFSR, the UTMC applies two pre-defined vectors holding log and anti-log tables for circuit optimization. The convolutional encoding generates two encoded bits for each received bit and performs this action as the last step before to send the data to the UTMC output and follow the mechanism presented in Figure 26.

Figure 26 – Convolutional Encoder.



Source: adapted from (BEZERRA et al., 2010).

3 RELATED WORKS

This chapter presents the related work in the context of communication protocols. In addition, there is a discussion around the system controller, considering its internal resources and the PUS Services usage. A comparison among the related works, highlighting important features of each work, is summarized in a table at the end of the chapter.

Selčan, Kirbiš and Kramberger (2015) present in their work an FPGA-based Communication Stack for nanosatellites using the CCSDS recommendations. The work implements its communication protocols, the TC flow performs the BCH code, and the TM flow implements a concatenated code, which combines Reed-Solomon encoding with the Convolution encoding. Following the CCSDS recommendations, the authors developed a FARM with some modifications targeting the reduction of resource use. However, they do not give a comparison between this modified FARM and the proposed in the CCSDS, in terms of used resources. In order to interface the communications between the systems that apply the Communication Stack, the architecture uses a CAN bus with the CAN 2.0 B protocol. This Communication Stack was designed for the TRISAT mission (TRISAT, 2019).

In Suresh et al. (2015), the authors describe the development of a Command and Data Management System (CDMS) to the nanosatellite IITMSAT. The authors developed the system using a 32-bits ARM microcontroller based on the mbed-RTOS system to schedule the system tasks. As principal function, the controller has as to manage with the generated telemetry data coming from the payload and sensors by I^2C and SPI interfaces. The system also uses an adapted version of the CCSDS protocol, which, notwithstanding the lack of information about the EDAC in the TC flow, the authors describe the use of a convolutional code with a constraint length of 5 in the TM channel coding (MEVADA et al., 2015). This work also implements the following PUS services: Telecommand Verification; Memory Management; Payload Management; Function Management; Onboard Operations Scheduling; Large Data Transfer; and On-Board Storage and Retrieval.

Ivanov et al. (2014) describes the CubETH project, that was developed by the Swiss's Polytechnical School. The system was implemented by using a protocol following the ECSS recommendations (based on earlier recommendations of the CCSDS). In the telecommand flow, the BCH coding scheme was used, as well as, the Reed-Solomon

and Convolutional codes in the telemetry flow. The system implementation is made in an MCU (Microcontroller Unit) MSP340. This work also describes a CDMS that acts as the On-Board Computer that has the primary function to get the telemetry information and scheduling the received commands.

The European Space Agency (ESA) is developing a nanosatellite in a CubeSat format which is named OPS-SAT (European Space Agency, 2019). This project proposes the use of CCSDS protocols, the implementations of the communication stack and the coding schemes based on an IP core embedded into a radiation hardened reconfigurable FPGA. Also, the project follows the PUS standard services (EVANS; MERRI, 2014).

In this context, this work proposes the implementation of a communication module into a Rad-Hard FPGA following the CCSDS recommendations. The objective is the usage in nanosatellites missions. This module can be configured to use the BCH scheme into the telecommand flux, as well as the RS, Convolutional, and a combination between RS and Convolutional in the telemetry flux. Thus, the work presents a range of possible EDACs that could be chosen depending on the mission's requirements. Following the CCSDS recommendations, the Communications Operations Procedure (COP) that defines the transmissions frames reliability, has a pertinent part defined as the Frame Acceptance and Reporting Mechanism (FARM), this retransmission mode is used in the implementation.

This work utilizes UART communication interface between the module and others systems (e.g., radio transceiver), it is possible to modify this interface to SSI or SPI, but the second one is limited in the telemetry flow. Also, the communication module applies the Command Pulse Distribution Unit (CPDU), this service is defined in PUS Standard (ECSS-E-ST-70-41C) and enables direct access from the ground to onboard equipment.

Table 3 synthesizes the principal's points of the presented works and aims to facilitate the comparison between them. Fields marked with a dash (-) refer information that was not founded, and fields marked with "N.I." refer to not implemented.

Concerning the presented works, hardware implementation using FPGA, as mentioned in Section 2.2, can potentially achieve higher performance than a microcontroller solution since its remove the overhead of reading/interpreting/executing each instruction that is applied in microcontroller architectures. Besides, the use of a radiation-hardened component increases the reliability of the system once it is subjected

to a radiation environment.

The present work uses the CCSDS protocol, including its coding layer. However, we can highlight that this work, besides applying the protocol structure, uses the retransmission mechanism performed by the FARM and implements the structure proposed by the ECSS PUS recommendation in the network layer. The CPDU service is used for direct ground access to subsystems.

Table 3 – Characterization of the systems presented in the related works.

	(Selčan; Kirbiš; Kramberger, 2015)	**(Suresh et al., 2015)**	**(Ivanov et al., 2014)**	**(Evans; Merri, 2014)**	**This Work**
Implementation	HW	HW	SW	SW	HW
Target Device	Rad-Hard FPGA	FPGA	MCU	MCU	Rad-Hard FPGA
Protocol	CCSDS	CCSDS	CCSDS	CCSDS (custom)	CCSDS
TM EDAC	RS, Convolutional	Convolutional	RS, Convolutional	-	RS, Convolutional
TC EDAC	BCH	-	BCH	-	BCH
Retransmission	FARM (custom)	N.I.	N.I.	-	FARM
Com. Interface	CAN	UART	-	SPI	UART
PUS Services	N.I.	Some Services	N.I.	Implemented	CPDU

Source: Made by the author, based in: (SELČAN; KIRBIŠ; KRAMBERGER, 2015),(SURESH et al., 2015),(IVANOV et al., 2014),(EVANS; MERRI, 2014).

4 PROPOSED COMMUNICATION MODULE

4.1 MATERIALS AND METHODS

4.1.1 Hardware Description Language

This work uses VHDL (VHSIC Hardware Description Language) to model digital systems at several levels of abstraction ranging from the algorithm level to the gate level. This language is an integration of: sequential; combinatorial; concurrent; net-list; timing constraints; and waveform generation language (BHASKER, 1999; BEZERRA; LETTNIN, 2014).

Especific CAD (Computer Aided Design) tools use the VHDL language to converter the description that models the system into a stream of bits used to programming the FPGA. The flow to generate the final bit stream to program the FPGA could be roughly divided into five steps: logic synthesis; technology mapping; mapping; placement; and routing (FAROOQ; MARRAKCHI; MEHREZ, 2012).

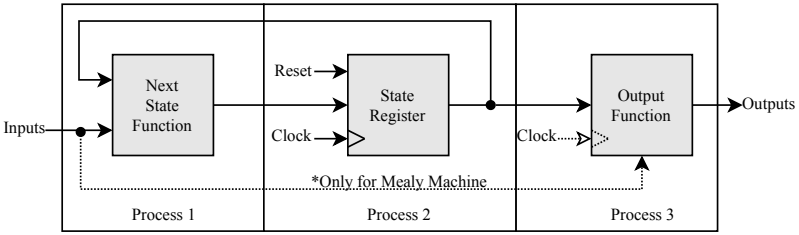
4.1.2 Finite State Machine Coding

The hardware description uses a set of finite state machine (FSM) to control the circuit. The construction model is based on the *XTS User Guide* defined by Xilinx (XILINX; INC, 2002), following the standard used in the implementation of the UTMC.

The HDL Coding Technique of FSM implementations is the three process scheme. This model constitutes of three blocks: a "Next State Function" that applies the logic to define the next state based on the actual one and is described by the process 1; a "State Register" which implements a register that stores the current state as a function of the next state and is described by the process 2; and the "Output Function" that is responsible for output assignments and is described by the process 3 (XILINX; INC, 2002; BEZERRA; LETTNIN, 2014). The output function can be registered, and, in this case of a Mealy Machine, the generated outputs are related to inputs.

Figure 27 presents the diagram of this scheme relating the blocks with their process, and a VHDL example description is presented at Appendix A - FSM VHDL Coding Example.

Figure 27 – Finite state machine with three processes diagram.



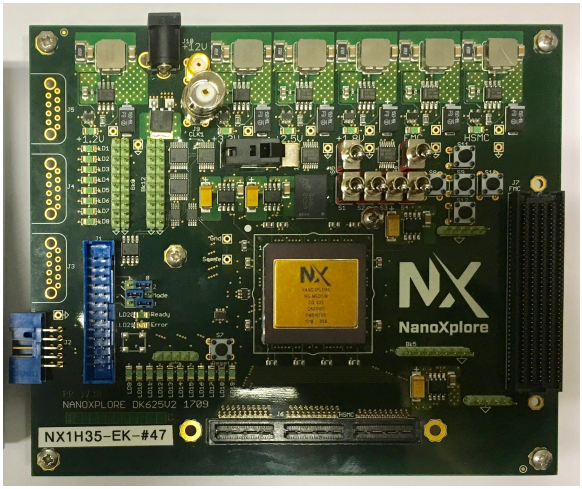
Source: adapted from (XILINX; INC, 2002).

4.1.3 Development Kit

For the hardware implementation, a development kit board (DevKit), provided by the NanoXplore company, was used.

The DevKit provides means for testing the implementation on BRAVE NG-Medium CLGA625 FPGA, and afford access to an onboard crystal oscillator of 25 MHz and also externals SMA clocks inputs, a 128Mx16 DDR2 memory chip, interfaces such as switches, pushbuttons, LEDs and connectors. Figure 28 presents the development kit.

Figure 28 – NanoXplore Brave development kit.



Source: from author.

4.1.4 Cortex

During the development work, it was possible to have access to Cortex equipment that is a Command Ranging & Telemetry Unit which is a reference for earth observation missions since it has as main features Telemetry data processing and provides CCSDS decoders. This equipment can work with a 70 MHz intermediate frequency, a Doppler estimation and spread spectrum processing for TM and TC.

During the short period of testing, the telemetry flow test was performed with the objective of verifying the stability of the data generated by the implementation. The test setup was composed of the Brave DevKit running the UTMC implementation in a continuous transmission mode. The telemetry output is connected with the Cortex that checks data consistency and performs a de-codification accordingly the used EDAC. Moreover, through a TCP/IP interface, a computer stores the logs from the Cortex. Figure 29 presents the equipment.

Figure 29 – Cortex setup.

Source: (Censin Technology, 2019)

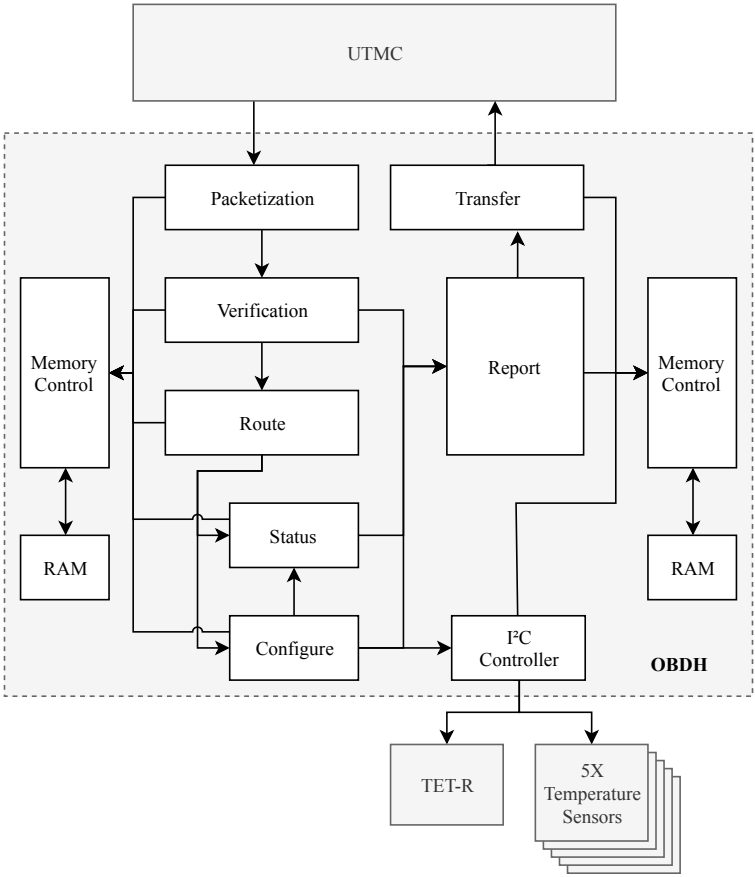
4.2 ON-BOARD DATA HANDLING

The On-Board Data Handling (OBDH) has two main objectives: validation and distribution of received telecommands; and generation and packetization of the generated data. Then, the architecture provides a collection of entities with different functionality being able to

identify and decode a received telecommand using CRC algorithms, generate reports by request, and packet the generated data (e.g., I^2C sensors).

The OBDH handles the Network Layer defined by the Section 2.4.3, by using the Space Packet Protocol based on the ECSS PUS definition. The following subsections present the developed architecture and its functionality. To represent the developed OBDH architecture, Figure 30 presents a top-level diagram with the essential connections between the blocks that constitute this system.

Figure 30 – OBDH top level diagram.



Source: from author.

The internal communication between the blocks is based on the use of two embedded memories, one for the telecommand flow and the second for the telemetry flow. The use of these embedded memories allows the exchange of internal information with a smaller number of interconnections and makes the inclusion of new blocks flexible to this architecture since there is no need for significant changes in the existing blocks to have access to the data.

The basic functionality of the blocks are presented as following:

- Packetization Layer: it receives the telecommand data from UTMC and stores the information in the telecommand embedded memory;
- Verification Layer: it verifies the structure of the received telecommand and verifies the data integrity with a CRC algorithm;
- Route Layer: when a telecommand passes through the verification, this layer defines the data destination and distributes the command;
- Configure Layer: it changes under request the value of an internal register (e.g., register that defines the interval time between I^2C acquisitions);
- Status Layer: generates by request status reports with information of the internal registers values and reports how many telecommand packets were successfully received and how many telemetry packets were generated;
- Report Layer: it generates, under telecommand request, acknowledgments identifying the success or failure in the reception of a new telecommand; this report informs with an error code in the frame whether a failure occurred;
- I^2C Controller: it operates with the I^2C interface to acquire data from a set of sensors; this data is stored in the telemetry embedded memory;
- Memory Control: it controls the memory access of the blocks; this layer works with a static priority;
- Transfer Layer: when the amount of generated data is able to fill a complete telemetry packet (accordingly the CCSDS frame structure) or when a request is done, this layer reads the data from the telemetry memory and transfer using the communication interface.

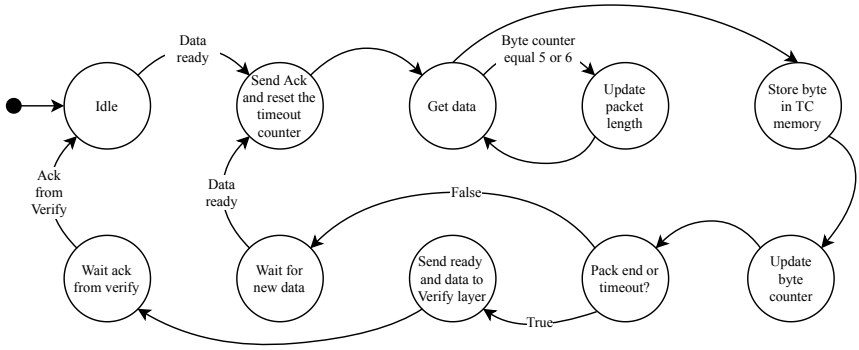
The following subsections present the operation of the blocks that make up the OBDH. In the graphical representations of the implemented state machines, the transitions are made unconditionally. However, the transitions are conditioned to the occurrence of changes in input signals, which are not explicitly shown in the FSMs, in order to simplify the graphic representation.

4.2.1 Packetization Layer

Packetization Layer has the function to store the received telecommand in TC memory. There are two different mechanisms to define the end of a new packet. The first one identifies the information about the packet length, this can be done checking the 5^{th} and 6^{th} byte of the Space Packet Protocol. Then, at each received data, this layer verifies if the packet is finished.

The second mechanism is the utilization of a timeout counter since the information about the packet length can be corrupted. During the receiving process, if the final of a packet is not achieved and the implementation did not receive a new byte during a time window, the process identifies a timeout and this layer informs the Verification Layer that a timeout occur and the packet is discarded. Figure 31 presents a simplified FSM of this procedure.

Figure 31 – Packetization layer finite state machine.



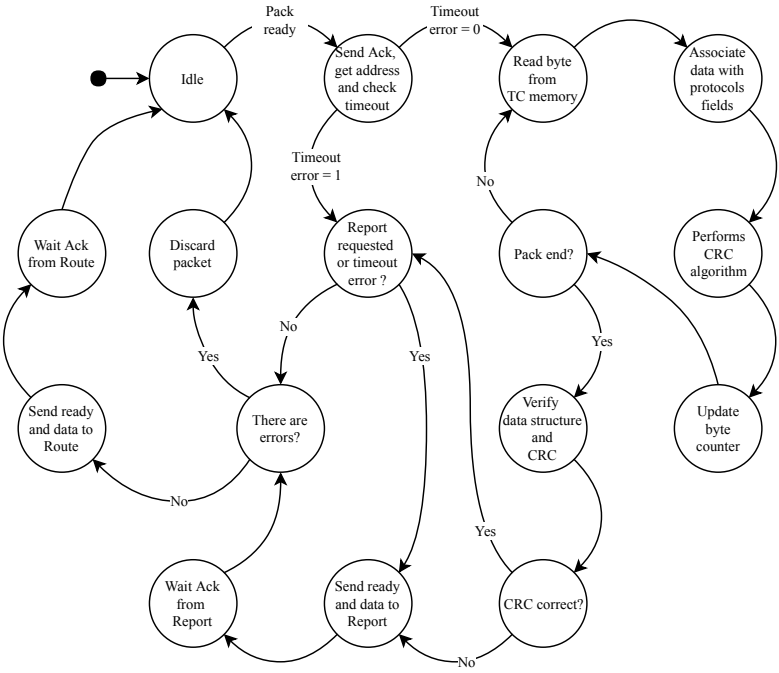
Source: from author.

4.2.2 Verification Layer

The Verification Layer starts its procedure when the Package process is complete. The first step is to check if a timeout error has occurred in the Packetization layer, and, when true, a report is generated with a timeout error, and the packet is discarded. If no timeout error occurs, the systems begin to obtain the data and verify if the fields that made up the structure of a space packet are consistent and starts the CRC algorithm.

When all data are verified, the FSM can follow different flows. If the CRC and data are consistent, this layer will report a ready signal to the Route Layer, and if there is a request, a report will be generated. If the CRC is incorrect, a report is generated, and the package is discarded. If a fielding error has occurred, the report is generated if requested, and the packet is discarded. Figure 32 presents the FSM of this layer.

Figure 32 – Verification layer finite state machine.

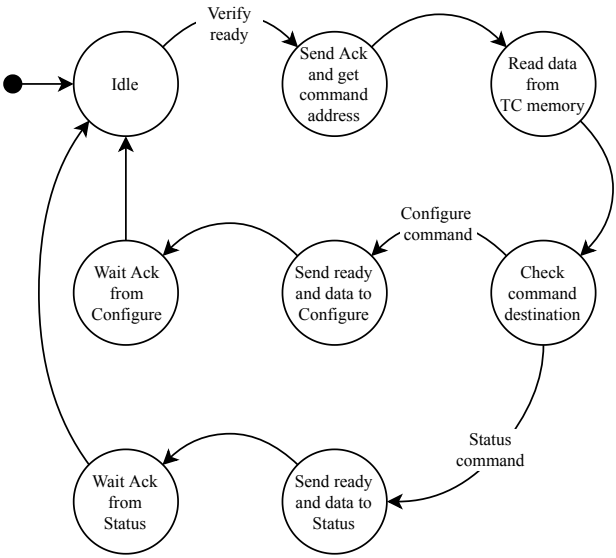


Source: from author.

4.2.3 Route Layer

Since the Verification layer sends a ready signal, the Routing layer gets the telecommand data and identifies the destination. Next, the process routes a ready signal to its destination and wait for an acknowledgment. Figure 33 shows the FSM applied in this process.

Figure 33 – Route layer finite state machine.

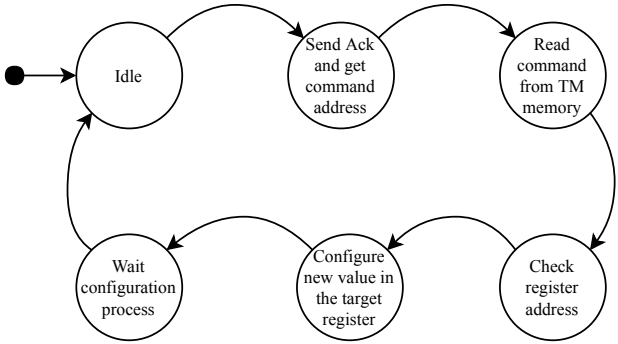


Source: from author.

4.2.4 Configure Layer

Configuration layer is capable of reconfiguring the value of an internal register, for example, the register that defines the acquisition period of the I^2C Controller, the procedure is simple, and this entity is able of modifying the register when requested. Figure 34 presents an FSM to show this procedure.

Figure 34 – Configuration layer finite state machine.

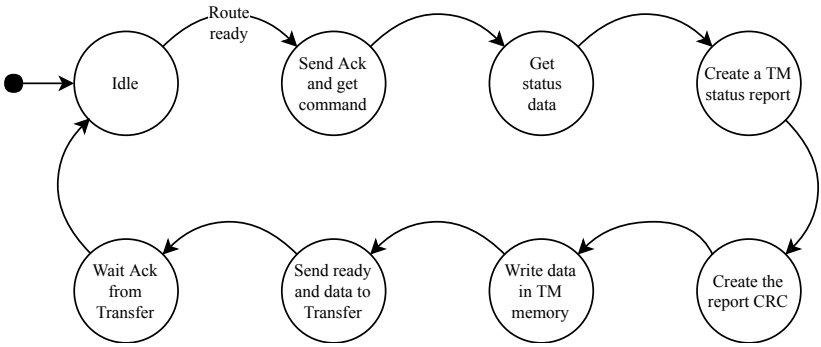


Source: from author.

4.2.5 Status Layer

The status layer is able of creating a packet with information about the internal configurable registers and informing how many successfully packets are received and sent. The report follows the structure defined in the ECSS PUS with an error control field that is filled by a 16 bits CRC. Figure 35 presents the FSM that defines the process.

Figure 35 – Status layer finite state machine.

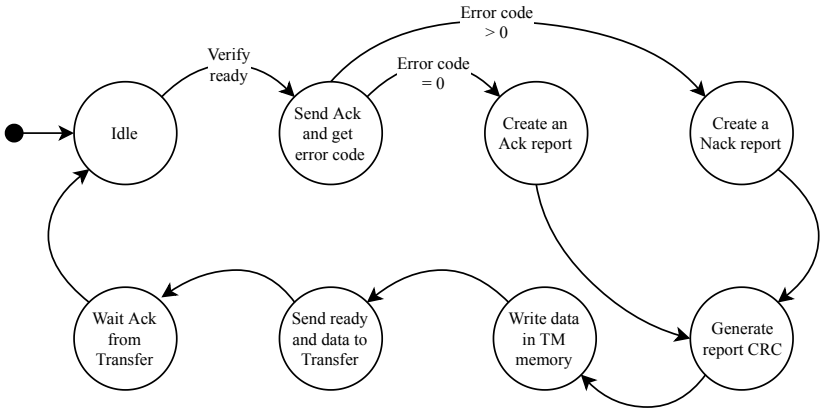


Source: from author.

4.2.6 Report Layer

The Report Layer creates an ACK or NACK report according to the error code received from the Verification layer. As the previous blocks, the Report follows the ECSS PUS structure and maintains a different counter for ACK and NACK packets. The Report layer process is presented in Figure 36.

Figure 36 – Report layer finite state machine.

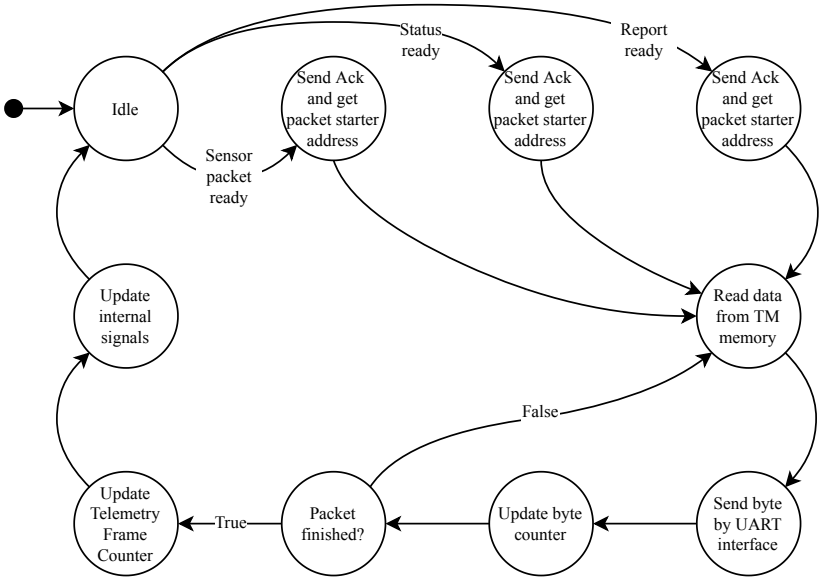


Source: from author.

4.2.7 Transfer Layer

The transfer layer regularly checks the amount of data generated by the sensors and requests from the Report and Status layers. Then, when the sensors data achieve a size to fill a full telemetry packet, or a request event occurs, this layer read the data from the TM memory and transfers it through the communication interface. The FSM applied in Transfer layer is showed in Figure 37.

Figure 37 – Transfer layer finite state machine.



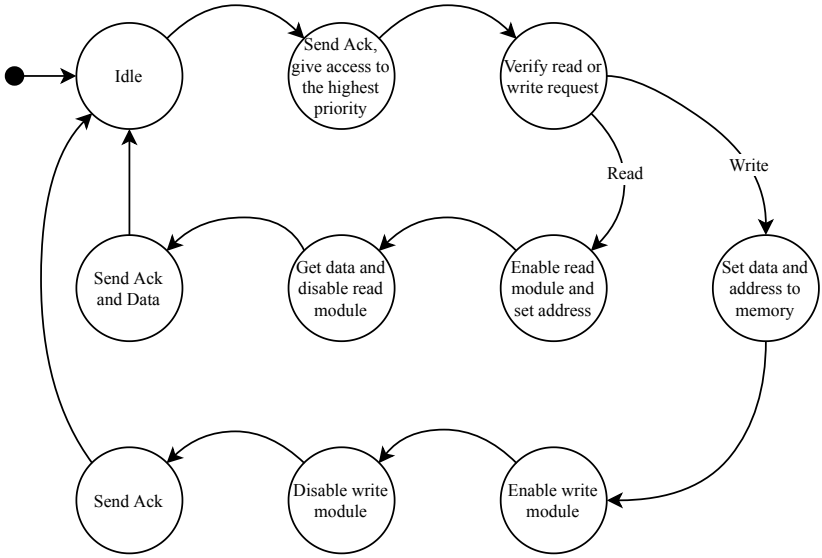
Source: from author.

4.2.8 Memory Control

Memory Controller acts as an interface between the blocks that access the embedded memory. When the process receives request signals the access is given to the block with highest priority. As the procedures to receive, verify, route and act from a telecommand are made in stages, the request signals are needed at different times, and there is no need of a dynamic priority algorithm. However, in the TM flow, the I^2C Controller accesses the memory regularly and to allow Report and Status blocks to access memory, which requires sporadic access. This layer has the lowest priority in the system.

In addition to the priority access mechanism, this layer applies the write and read commands according to the address (read procedure) and data (read and write procedure) for all blocks connected to it. Figure 38 presents a simplified state machine that explicates the process sequence.

Figure 38 – Memory control finite state machine.



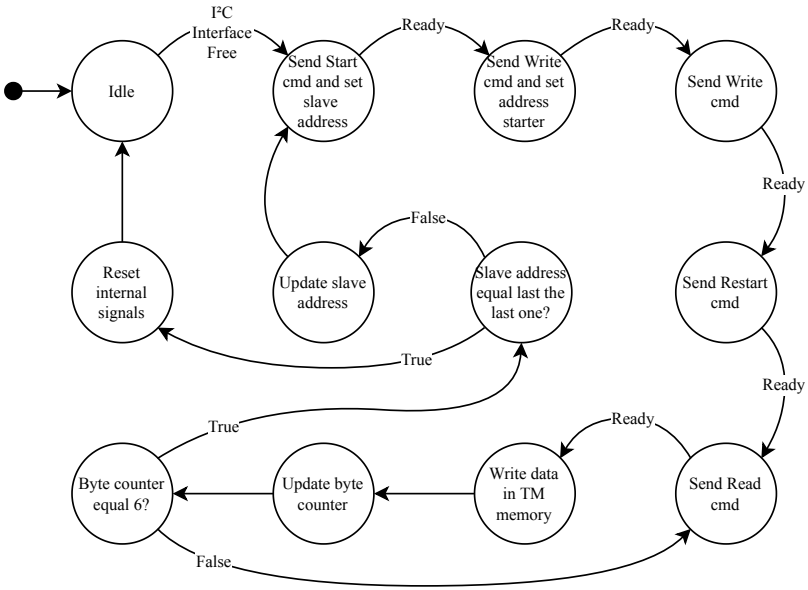
Source: from author.

4.2.9 I²C Controller

The I²C Controller interacts with a communication interface and controls the flux to receive data from all the sensors connected in the bus. The layer stores the acquired data in the embedded memory according to an interval time defined by an internal register (the same that can be modified by the Configuration layer).

Figure 39 presents the flow of this process using a FSM. This layer can be easily modified to achieve different procedures in the reading protocol of the devices. The actual architecture works with five temperature sensors and a total-elapsed-time recorder (TET-R).

Figure 39 – I²C controller finite state machine.



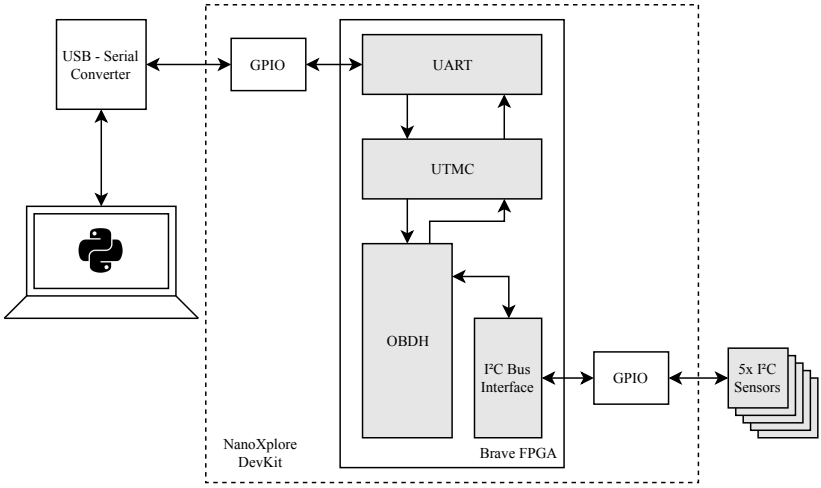
Source: from author.

4.3 TEST SETUP

Besides the use of the Cortex equipment, a test setup was used to check the correct flow of telecommands and telemetry using the FPGA implementation. The proposed setup uses a USB-RS232 converter, the NanoXplore development kit and python scripts are responsible for the data flow control between a host computer and the DUT (Device Under Test).

In this test context, valid and invalids CLTUs are generated by a python script according to the test needs. Then the data are sent by the USB Serial converter using a UART interface with a transmission speed of 115200 bit/s. The received CLTU is transmitted to the UTMC and from there different results can be achieved. Since the telecommand can have different destinations, when a data is expected as an answer, the Python script is able to get the data from the UTMC and plot according to the CCSDS packets structures. Figure 40 presents the proposed setup.

Figure 40 – Test setup

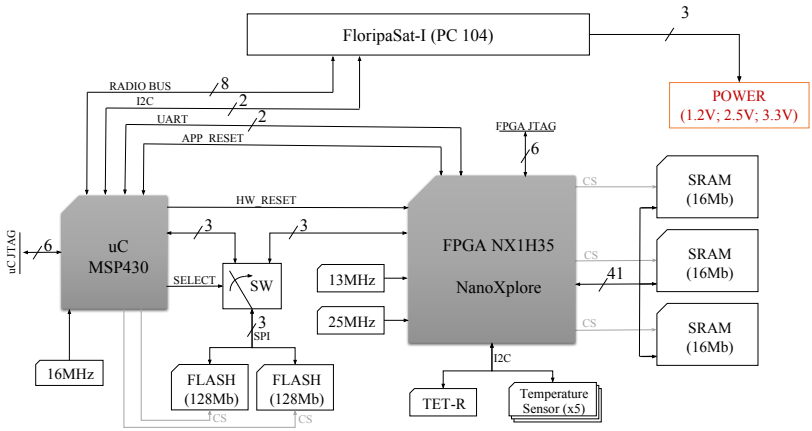


Source: from author.

4.4 PAYLOAD-X

In the context of this research, a payload was developed for the FloripaSat mission, with the purpose of in-orbit validation of the communication module and a hardware configuration update mechanism. This project is named Payload-X and Figure 41 shows the project interconnection diagram.

Figure 41 – Payload-X interconnection diagram.



Source: from (LUZA et al., 2018; RIGO et al., 2019).

The printed circuit board (PCB) where the architecture is implemented was designed by Cézar A. Rigo as a result of his master project (RIGO, 2019), the design follows the European Space Agency (ESA) space product standards. It has a layered structure that mitigates the effects of radiation and electromagnetic interference on the components signals. All signal layers lie between power and ground planes, avoiding tracks on the outer layers. Moreover, all components were selected to tolerate wide temperature variation, and some of then can tolerate radiation – as the microcontroller (MSP430FR6989) with ferromagnetic program memory and the rad-hard FPGA (NX1H35S-BG625PR) (LUZA et al., 2018; RIGO et al., 2019).

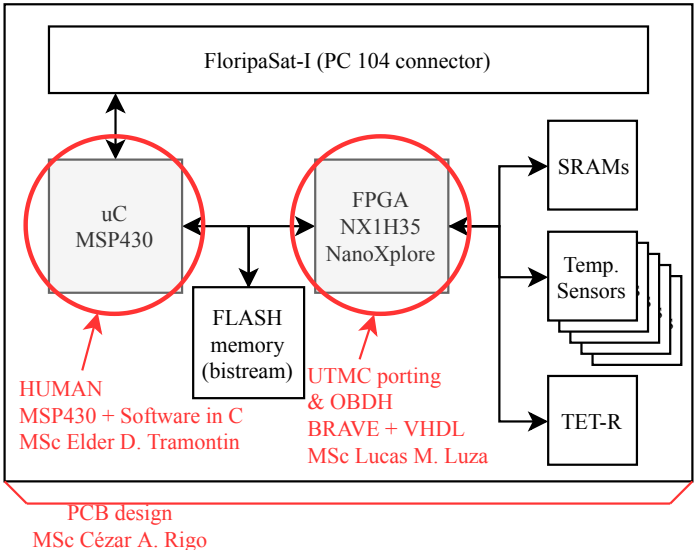
A central feature of the architecture is its capability to change the hardware configuration of the FPGA through a remote up-link of its bit stream. The proposed mechanism is called HUMAN (Housekeeper and Update MANager) and was proposed in the master project

from Elder D. Tramontin (TRAMONTIN, 2018). The MCU is responsible for updating the configuration bistream stored in a non-volatile flash memory. An alternative bitstream is also stored in the memory, for applying a fail-safe technique. There are three stored bitstreams copies, and a voting scheme is used to ensure data integrity, since the flash memory is susceptible to SEE. The MCU module is also responsible for the housekeeping and for the update management of the radiation hardened FPGA. The architecture also includes five temperature sensors, a TET-R and three SRAM devices. The data collected by these devices will be used for radiation monitoring (LUZA et al., 2018; RIGO et al., 2019).

In the present work, it has been developed the application stored in the FPGA, including the proposed OBDH, and the the UTMC porting to the BRAVE FGPA. The communication module handles TC and TM data, and it is an interface between the radio transceiver and the OBDH. The OBDH is based on the CCSDS and ECSS recommendations and performs the validation of the received telecommand, and the packeting of the telemetry that is acquired by the available sensors.

Figure 42 presents these works contribution overview.

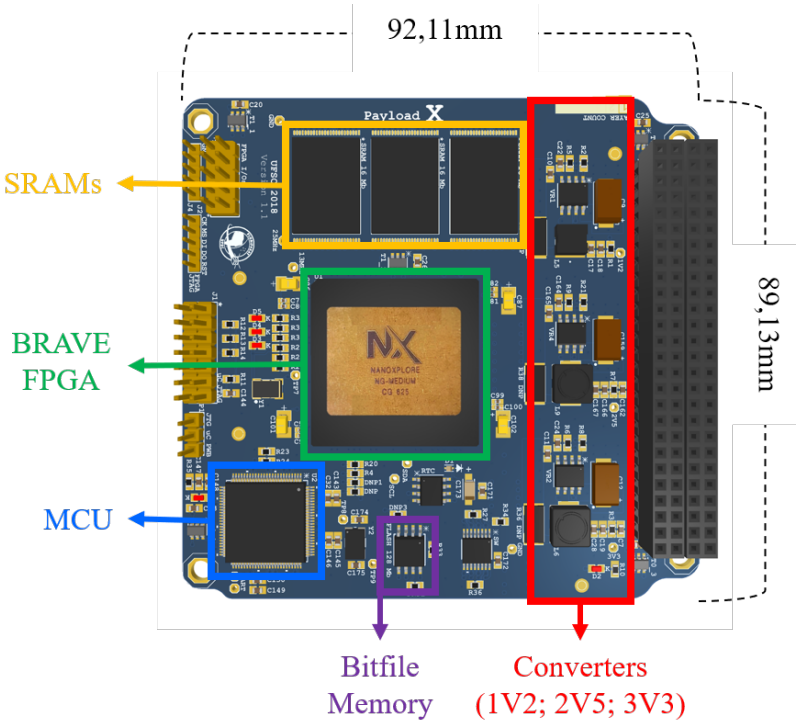
Figure 42 – Contribution for the Payload-X project.



Source: from author.

Figure 43 presents the model of the board.

Figure 43 – Payload-X board.



Source: from (LUZA et al., 2018; RIGO et al., 2019).

4.5 FLORIPASAT-I INTEGRATION

The payload architecture presented in the previous section is under integration in the FloripaSat mission of the Federal University of Santa Catarina. The FloripaSat-I satellite consists of a 1U CubeSat with five functional modules: EPS; OBDH; TT&C; battery and interface board, besides the payload modules where the Payload-X is integrated (SLONGO et al., 2016).

Payload-X integration with FloripaSat-I is based in two operation modes. The first mode is called Nominal and it uses the I^2C interface available to communicate with the FloripaSat-I OBDH. It is

treated as a safe operating mode. The second mode, called Advanced, uses the SPI interface and GPIO to control the CubeSat radio during a window of 4 hours or while the energy level is at higher status, and it is used as a risky operation mode once it takes control of CubeSat's main communication channel. The following subsections will define both operation modes and the data flow related to case scenarios.

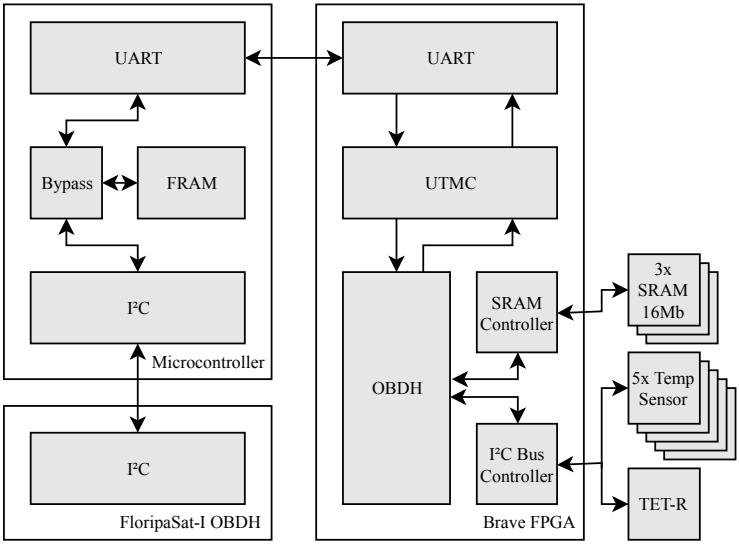
4.5.1 Nominal Operation Mode

The Nominal operation mode was designed to use the interface communication already available in FloripaSat-I's architecture since it has an I^2C channel to communicate with the payload. The UTMC does not have an I^2C interface, so the microcontroller is used to convert this I^2C to a UART communication interface. The microcontroller also uses its FRAM memory to store telemetry received from the FPGA implementation, then, when the FloripaSat-I OBDH requires data from the Payload-X, the microcontroller sends the last received telemetry to be stored by the FloripaSat-I. Figure 44 shows the blocks diagram of Nominal mode.

4.5.2 Advanced Operation Mode

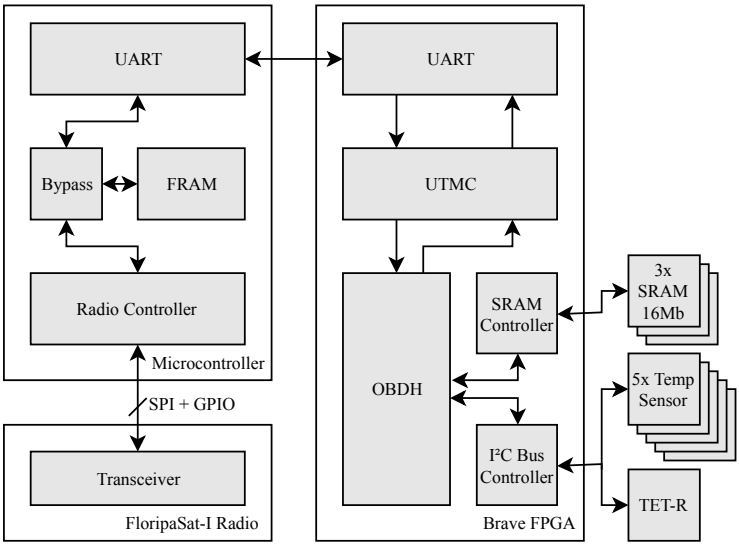
The Advanced mode was designed to make use of the CCSDS and ECSS protocols directly, bypassing the NGHam protocol used by the FloripaSat-I communication module. A telecommand starts this mode, and when the FloripaSat-I recognizes this telecommand, it allows the Payload-X to assume full control of the radio transceiver for a 4 hour time window. Figure 45 presents the diagram of Advanced mode.

Figure 44 – Nominal operation mode top level diagram.



Source: from author.

Figure 45 – Advanced operation mode top level diagram.



Source: from author.

4.5.3 Data Flow

For each case, there is a diagram showing the respective data flow, and there is a step-by-step description of the performed activities. The acronym FS will be used to referer to FloripaSat-I, and the acronym PX will be used for Payload-X.

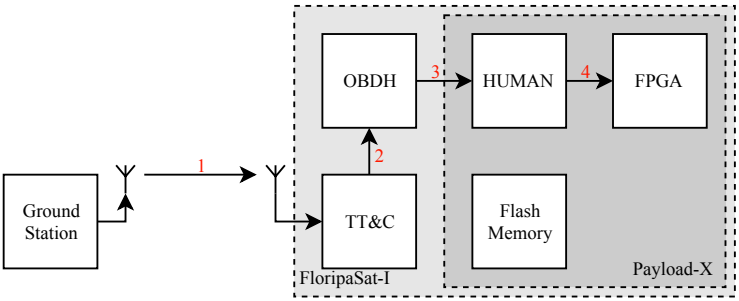
4.5.3.1 CCSDS Telecommand

For the validation of the CCSDS communication, it is necessary to send some data or telecommand over CCSDS protocol. Until reaching the UTMC (CCSDS decoder), the packet should pass through other devices. The Figure 46 shows the data flow of the CCSDS telecommand case. The activities performed in each step are described in Table 4.

4.5.3.2 CCSDS Telemetry

As well as the CCSDS Telemetry, it is necessary to receive some data over CCSDS protocol to validate the CCSDS communication. Some events could generate telemetry: a telecommand or a measurement from Payload OBDH. The Figure 47 shows the data flow of the CCSDS telemetry case. The activities performed in each step are described in the Table 5.

Figure 46 – Use case CCSDS telecommand diagram.



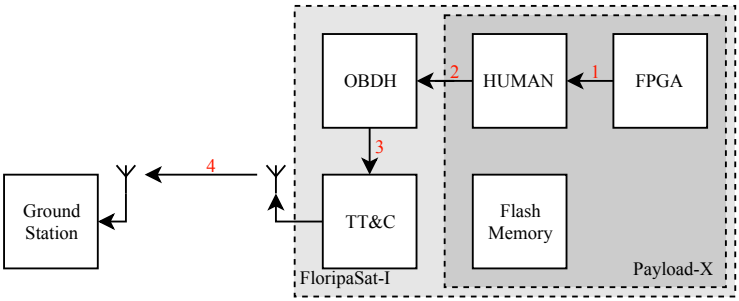
Source: from author.

Table 4 – CCSDS Telecommand step-by-step.

Step	Description	Source	Destination
1	Ground station sends data @ UFH to FloripaSat-I	Ground-station	FS-TT&C
2	FS-OBDH read data from radio	FS-TT&C	FS-OBDH
	FS-OBDH decodes NGHam packet	FS-OBDH	FS-OBDH
	FS-OBDH interprets the TC	FS-OBDH	FS-OBDH
3	FS-OBDH request to send TC data to PX	FS-OBDH	PX-HUMAN
	FS-HUMAN stores data on the FRAM memory	PX-HUMAN	PX-HUMAN
4	PX-HUMAN sends TC to PX-UTMC	PX-HUMAN	PX-UTMC
	PX-UTMC decodes CLTU frame	PX-UTMC	PX-UTMC
	PX-UTMC decodes TC frame	PX-UTMC	PX-OBDH
	PX-OBDH decodes TC Packet	PX-OBDH	PX-OBDH
	PX-OBDH extracts usefull data	PX-OBDH	PX-OBDH

Source: from author.

Figure 47 – Use case CCSDS telemetry diagram.



Source: from author.

Table 5 – CCSDS Telemetry step-by-step.

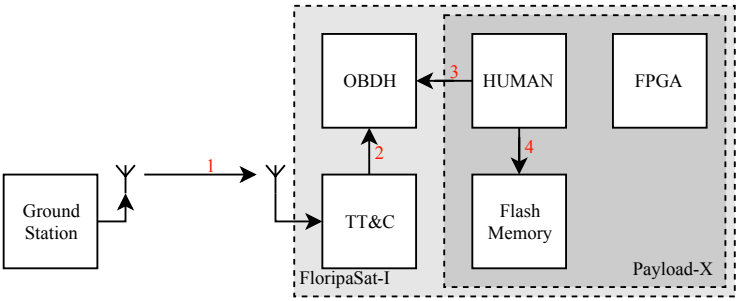
Step	Description	Source	Destination
1	PX-OBDH generated a TM packet	PX-OBDH	PX-UTMC
	PX-UTMC sends a CCSDS TM packet to PX-HUMAN	PX-UTMC	PX-HUMAN
	PX-HUMAN stores the data in FRAM	PX-HUMAN	PX-HUMAN
2	When received a request, PX-HUMAN sends the data to FS-OBDH	PX-HUMAN	FS-OBDH
3	FS-OBDH decodes each packet in NGHam protocol and send to FS-TT&C	FS-OBDH	FS-TT&C
4	FS-TT&C send data @ UHF to Ground station	FS-TT&C	Ground-station

Source: from author.

4.5.3.3 Bitstream Upload

When a new bitstream needs to be uploaded to Payload X, it is split into many segments and sent through the radio. From the ground station until data reaching the target in the Payload X board, some control bytes are added to the segments. Figure 48 shows the data flow of a bitstream segment upload.

Figure 48 – Use case bitstream upload diagram.



Source: from author.

Table 6 – Bitstream upload step-by-step.

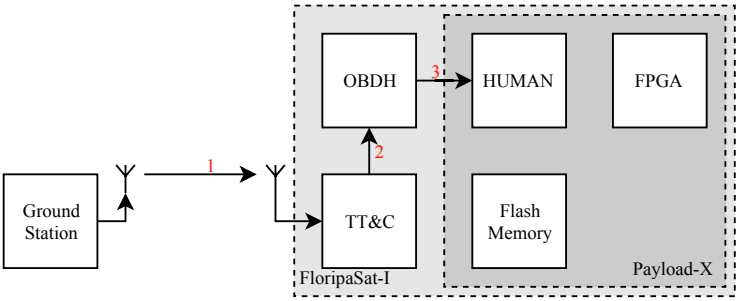
Step	Description	Source	Destination
1	Ground station sends data @ UHF to FloripaSat-I	Ground-station	FS-TT&C
2	FS-OBDH read data from radio	FS-TT&C	FS-OBDH
	FS-OBDH decodes NGHam packet	FS-OBDH	FS-OBDH
	FS-OBDH interprets the telecommand	FS-OBDH	FS-OBDH
3	FS-OBDH request to send bitstream data to Payload-X	FS-OBDH	PX-HUMAN
	PX-HUMAN stores data on the FRAM memory	PX-HUMAN	PX-HUMAN
	PX-HUMAN check message integrity using CRC	PX-HUMAN	PX-HUMAN
4	PX-HUMAN stores data at 3 positions on the flash memory	PX-HUMAN	Flash-memory

Source: from author.

4.5.3.4 Bitstream Status Request

Since the bitstream upload has no acknowledgment, to know if the packets reached correctly the target, it is necessary to send a “Status Request" telecommand. The Figure 49 shows the data flow of a bitstream status request.

Figure 49 – Use case bitstream status request diagram.



Source: from author.

Table 7 – Bitstream status request step-by-step.

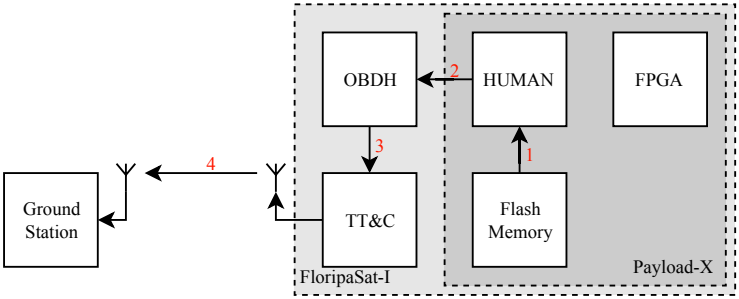
Step	Description	Source	Destination
1	Ground station sends data @ UHF to FloripaSat-I	Ground-station	FS-TT&C
2	FS-OBDH read data from radio	FS-TT&C	FS-OBDH
	FS-OBDH decodes NGHam packet	FS-OBDH	FS-OBDH
	FS-OBDH interprets the telecommand	FS-OBDH	FS-OBDH
3	FS-OBDH request bitstream status data from PX-HUMAN	FS-OBDH	PX-HUMAN

Source: from author.

4.5.3.5 Bitstream Status Reply

Once the status request arrives at the HUMAN, it is necessary to answer. Since bitstream segment status is too large to fit into 1 data package, it needs to be split into many packets and, then, send over the radio. Figure 50 shows the data flow of a bitstream status reply.

Figure 50 – Use case bitstream status reply diagram.



Source: from author.

Table 8 – Bitstream status reply step-by-step.

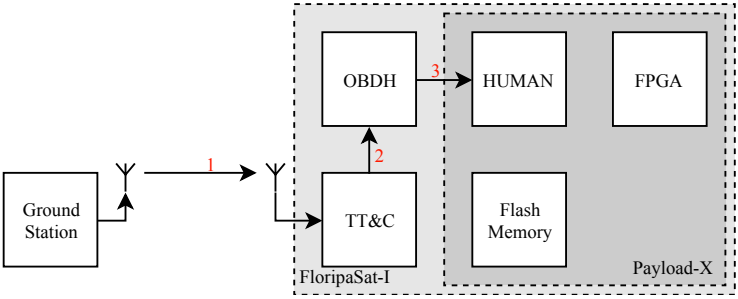
Step	Description	Source	Destination
1	Ground station sends data @ UHF to FloripaSat-I	Ground-station	FS-TT&C
2	FS-OBDH read data from radio	FS-TT&C	FS-OBDH
	FS-OBDH decodes NGHam packet	FS-OBDH	FS-OBDH
	FS-OBDH interprets the telecommand	FS-OBDH	FS-OBDH
3	FS-OBDH requests from Payload-X to swap version	FS-OBDH	PX-HUMAN
	PX-HUMAN change the flag that indicates the current version in use	FS-HUMAN	FS-HUMAN

Source: from author.

4.5.3.6 Bitstream Swap Version

There are always two bit stream versions on-board the Payload-X: the currently used version and the alternative version (fallback). To swap from the latter to the former, it is necessary to send a "Bitstream Swap Version" telecommand. Figure 51 shows the data flow of a bitstream swap version.

Figure 51 – Use case bitstream swap version diagram.



Source: from author.

Table 9 – Bitstream swap version step-by-step.

Step	Description	Source	Destination
1	PX-HUMAN reads the frame status from flash memory	PX-HUMAN	Flash-memory
2	PX-HUMAN sends status data to FS-OBDH	PX-HUMAN	FS-OBDH
	FS-OBDH encodes NGHam packet	FS-OBDH	FS-OBDH
3	FS-OBDH writes data on radio	FS-OBDH	FS-TT&C
4	FloripaSat-I sends data @ UHF to Ground station	FS-TT&C	Ground-station

Source: from author.

5 RESULTS

5.1 UTMC PORTING TO BRAVE FPGA

The implementation of the UTMC in the Brave FPGA required modifications over the original VHDL description since the NanoXplore synthesis tool infers logical blocks according to the resources available in the FPGA and the result of the RTL is different from those presented by tools such as Quartus, Vivado, and Libero (which was the tool used in the original UTMC design).

The two encountered problems were the inference of latches in state machine implementations and low operating frequency.

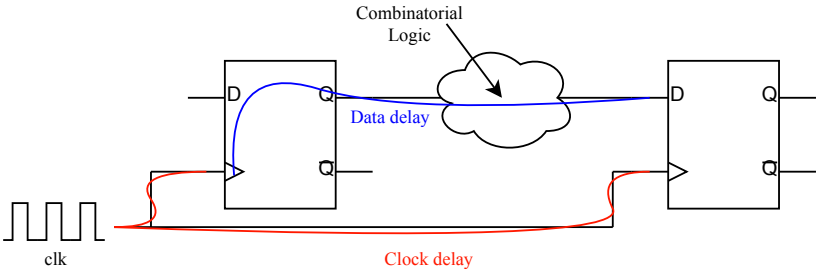
The solution for the first problem is directly connected with the way the synthesis tool works. The described process has to cover all the possibilities, then, the “case others” process statement needed a function. Using this need, if the process achieves the "others =>" statement it means that an error occurred in the transition from the current state to the new state, thereby this implementation uses this process to generate an error signal to force to reset in the FPGA.

Also, some processes need to be modified to avoid latch inference. An example of this modification is presented next.

```
-- Process with latch
process : process (input)
    begin
        if (input = '1') then
            state <= s1;
        end if;
    end process process;

-- Process without latch
process : process (input)
    begin
        if (input = '1') then
            state <= s1;
        else
            state <= s2;
        end if;
    end process process;
```

The approach related to the second issue is associated with the way the codification process has been implemented. The BCH implementation used a "for" statement to do the cyclic interactions, then, it generated a huge combinational logic between registers and created a long critical path. Figure 52 presents the problematic. The "Data delay" must be lower than the "Clock delay", then, with a large combinational logic, the maximum operation frequency is 8.5 MHz.

Figure 52 – Data and clock delay between a source and destination register.



Source: adapted from (NANOXPLORE, 2018b)

Then, the approach used to avoid this issue was to add pipeline, thus the "for" procedure was split in several stages reducing the combinational logic between the registers. Although, with this approach, the number of cycles were increased to finalize the operation, it was possible to achieve a four times higher operating frequency. This modification is necessary since the resources made available by the BRAVE FPGA are different from those of the original application, since in the original application the operating frequency was higher than required.

5.2 USE CASES

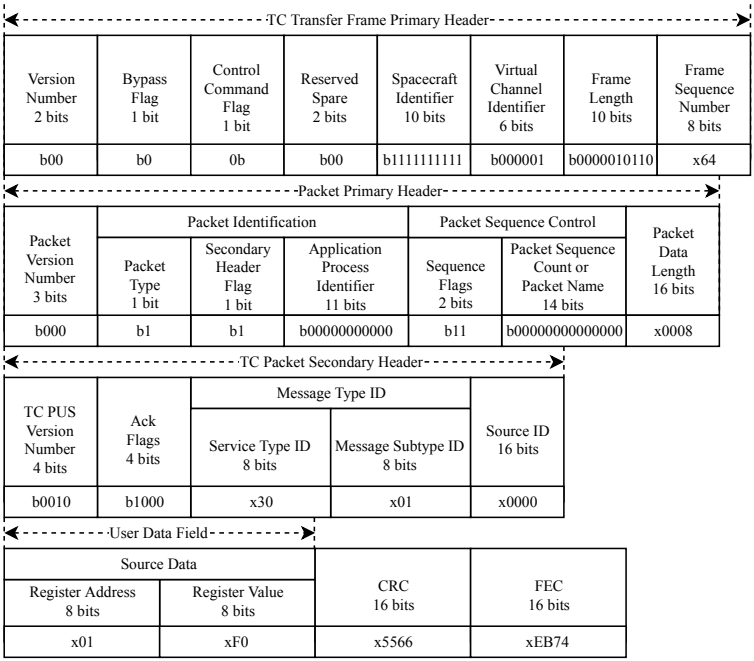
This section shows the results of the use cases of the system; each subsection presents its telecommand structure and also the process using the python script. Moreover, a bit flip is randomly inserted in all the constructed CLTUs, as a fault injection mechanism. The fault injection system is not detailed in the manuscript since it is out of the scope of this work.

5.2.1 OBDH Register Configuration Flow

The operation consists of sending a specific command that defines the address of the registers and the configuration values. Constructed from the data presented in the Figure 53, the command generates the CLTU, which includes all the CCSDS layers used in this implementation. The generated CLTU is sent from a python script that operates as serial USB converter, following the setup test presented in Section 4.3.

This command should generate a system response since a return is requested using the "Ack Flags" field. It is interesting to highlight that the "Ack Flags" field of the "TC Packet Secondary Header" is set as "*b1000*" meaning that a verification report has to be generated.

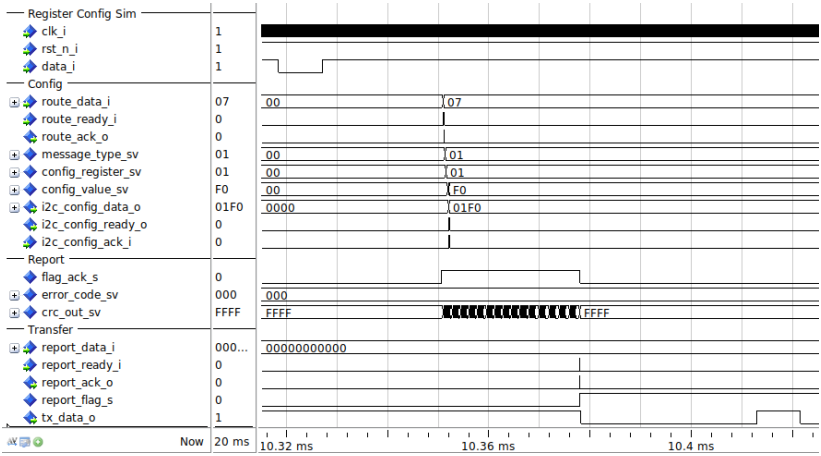
Figure 53 – OBDH register configuration telecommand structure.



Source: from author.

A previous simulation is done to verify the behavior of the system, the main signals related with this procedure are presented in Figure 54.

Figure 54 – Results captured from the VHDL simulation of the OBDH register configuration flow (ModelSim).



Source: from author.

The Configure Layer starts the configuration process when the signal "*route_ready_i*" changes from '0' to '1'. The pointer address presented in the vector signal "*route_data_i*" indicates the data location in the embedded memory, and from these data the process identify the destination "*config_register_sv*" and the new value "*config_value_sv*". Then, the destination receives the new value and returns an acknowledged to the layer showing a successful configuration.

As a result of this simulation, the Report Layer creates a report packet following the space packet structure. A positive acknowledged showed by the signal "*flag_ack_s*" (that is is a high state) that embeds the *"error_code_sv"* information (which in case of a positive acknowledged, is defined as "000h") and a 16-bit CRC "*crc_out_sv*" is generated.

The signal "*report_ready_i*" indicates that the Report Layer procedure is finished and the Transfer Layer starts to send the data through the UART interface for the UTMC.

Figures 55, 56 and 57 present the telecommand data and the received telemetry by using the test setup. Unlock and Set commands are needed to handle the FARM implementation. In Figure 56 the "Service Type ID" identifies that it is a verification report, and the "Message Type ID" indicates that is a ACK message. The "User Data" presents the Space Packet primary header of the source telecommand packet.

Figure 55 – OBDH register configuration flow: telecommand to be sent.

```
lucas@cubos-licenses:~/Projects/payload-x/scripts$ python3 test_procedure.py

__________Payload-X Test Procedure__________

=> Choose the use case:
  1 - Destination: OBDH Config | Expected: Ack
  2 - Destination: OBDH Status | Expected: Status Report
  3 - Destination: UTMC CPDU   | Expected: Ack
  4 - Destination: UTMC CPDU   | Expected: NAck (code error  x08)
  5 - Destination: OBDH        | Expected: NAck (code error x080)
Commando: 1

=> Telecommand (TT&C -> Payload-X):

__Unlock Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF04070000FA52
 BCH Code Block (8 bytes): xB20000000000008A
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__Set Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF0409008200EE
 BCH Code Block (8 bytes): x64CD230000000014
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__Configure Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x03FF041664C018BE
 BCH Code Block (8 bytes): x00C00000082830A0
 BCH Code Block (8 bytes): x01000001F05566BA
 BCH Code Block (8 bytes): xEB7400000000002A
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579
```

Source: from author.

Figure 56 – OBDH register configuration flow: expected telemetry (first part).

```
=> Telemetry (Payload-X -> TT&C):

ASM: x1acffc1d

Transfer Frame Primary Header: x3ff300001800
  .Version Number (2 bits): b00
  .Virtual Channel ID (3 bits): b001
  .Operational Control Field Flag (1 bit): b1
  .Master Channel Frame Counter (1 byte): x00
  .Virtual Channel Frame Counter (1 byte): x00
  .Transfer Frame Data Field Status (2 bytes): x1800

Space Packet Primary Header: x0800c0000010
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Application Process ID (11 bits): b00000000000
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): xc000
  .Packet Lenght (2 bytes): x0010

Space Packet Secondary Header: x200101000f000000000000
  .TM PUS Version Number (4 bits): b0010
  .Spacecraft Time (4 bits): b0000
  .Service Type ID (1 byte): x01
  .Message Type ID (1 byte): x01
  .Message Type Counter (2 byte): x000f
  .Destination ID (2 bytes): x0000
  .Time (1 byte): x00
  .Spare (3 byte): x000000

User Data : x1800c000

CRC (2 bytes): xeed6
```

Source: from author.

Figure 57 – OBDH register configuration flow: expected telemetry (second part).

```
Idle Packet: x07ffc000043155..55
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Secondary Header Flag (1 bit): b0
  .Application Process ID (11 bits): b11111111111
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): xc000
  .Packet Lenght (2 bytes): x0431
  .Idle Data: x55..55

CLCW: x01040465
  .Type (1 bit): b0
  .Version Number (2 bits): b00
  .Status (3 bits): b000
  .COP Use (2 bits): b01
  .Virtual Channel ID (6 bits): b000001
  .Reserved (2 bits): b00
  .Flags (5 bits): b00000
  .FARM-B Counter (2 bits): b10
  .Reserved (1 bit): b0
  .Informed Value (1 byte): b65

FECW (2 bytes): x177e

__________Test Procedure Finished__________
```

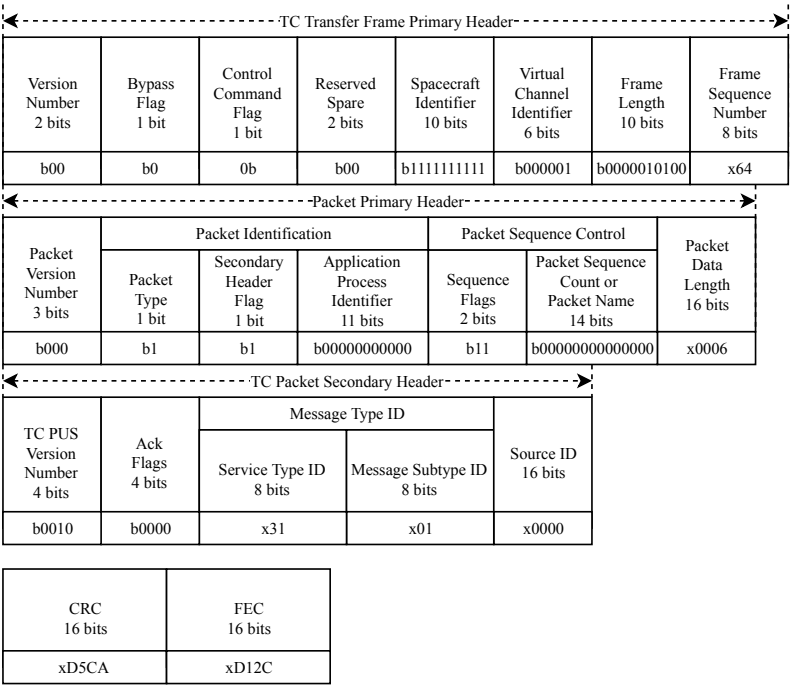
Source: from author.

5.2.2 OBDH Status Request Flow

The OBDH system has the capability to transmit a status telemetry frame under request. Following the same test procedure, the interface communication sends a request telecommand to the communication module, and the python script checks the received telemetry consistency.

Figure 58 presents the structure and data of the request telecommand, it is interesting to highlight that the "Ack Flags" field of the "TC Packet Secondary Header" is set to "*b0000*", mean that a verification report should not to be generated since the answer is the status telemetry.

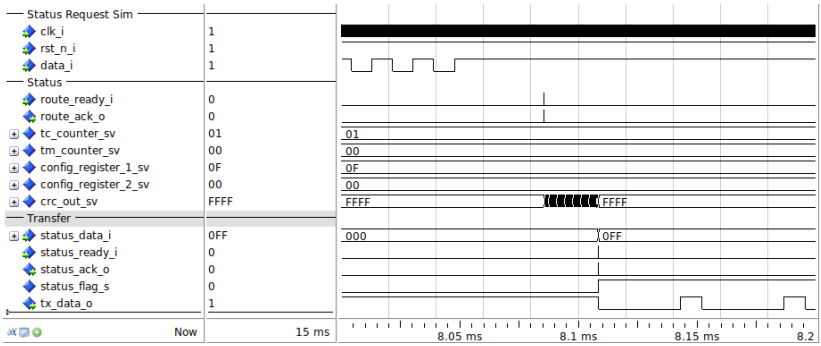
Figure 58 – OBDH status request telecommand structure.



Source: from author.

A simulation of this flow is presented in Figure 59.

Figure 59 – Results captured from the VHDL simulation of the OBDH status request flow (ModelSim).



Source: from author.

When the Status Layer receives a request signal *route_ready_i* from the Route Layer, a status packet is generated within the data get from "*tc_counter_sv*", "*tm_counter_sv*", "*config_register_1_sv*", and "*config_register_2_sv*" and the 16-bit CRC generated. When the status process is finished, the "*status_data_i*" is set high and the Transfer Layer starts to sent the packet to the UTMC using the UART interface.

Figures 60, 61 and 62 present the telecommand and the telemetry using the test setup.

Figure 60 – OBDH request status flow: telecommand to be sent.

```
lucas@cubos-licenses:~/Projects/payload-x/scripts$ python3 test_procedure.py

__________Payload-X Test Procedure__________

=> Choose the use case:
  1 - Destination: OBDH Config | Expected: Ack
  2 - Destination: OBDH Status | Expected: Status Report
  3 - Destination: UTMC CPDU   | Expected: Ack
  4 - Destination: UTMC CPDU   | Expected: NAck (code error  x08)
  5 - Destination: OBDH        | Expected: NAck (code error x080)
Commando: 2

=> Telecommand (TT&C -> Payload-X):

__Unlock Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF04070000FA52
 BCH Code Block (8 bytes): xB20000000000008A
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__Set Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF0409008200EE
 BCH Code Block (8 bytes): x64CD230000000014
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__Status Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x03FF041464C0182A
 BCH Code Block (8 bytes): x00C00000062031DA
 BCH Code Block (8 bytes): x01000074D5CAD12C
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579
```

Source: from author.

In Figure 61, the value ‘31’ in the “Service Type ID” identifies that a message is from the Status unit and the “Message Type ID” indicates that the message is a Status Report. The “User Data” field is composed of four bytes, the first byte gives the telemetry counter state, the second byte gives the telecommand counter state, the third byte is the value of the register 1 and the last byte is the value of the internal register 2.

Figure 61 – OBDH request status flow: expected telemetry (first part).

```
=> Telemetry (Payload-X -> TT&C):

ASM: x1acffc1d

Transfer Frame Primary Header: x3ff30e0e1800
  .Version Number (2 bits): b00
  .Virtual Channel ID (3 bits): b001
  .Operational Control Field Flag (1 bit): b1
  .Master Channel Frame Counter (1 byte): x0e
  .Virtual Channel Frame Counter (1 byte): x0e
  .Transfer Frame Data Field Status (2 bytes): x1800

Space Packet Primary Header: x0800c00e000e
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Application Process ID (11 bits): b00000000000
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): xc00e
  .Packet Lenght (2 bytes): x000e

Space Packet Secondary Header: x203102000c00000000
  .TM PUS Version Number (4 bits): b0010
  .Spacecraft Time (4 bits): b0000
  .Service Type ID (1 byte): x31
  .Message Type ID (1 byte): x02
  .Message Type Counter (2 byte): x000c
  .Destination ID (2 bytes): x0000
  .Time (1 byte): x00
  .Spare (1 byte): x00

User Data : x00010AFF
```

Source: from author.

Figure 62 – OBDH request status flow: expected telemetry (second part).

```
 Idle Packet: x07ffc000043355..55
   .Packet Version Number (3 bits): b000
   .Packet Type (1 bit): b0
   .Secondary Header Flag (1 bit): b0
   .Application Process ID (11 bits): b11111111111
   .Sequence Flag (2 bits): b11
   .Sequence Count (14 bits): xc000
   .Packet Lenght (2 bytes): x0433
   .Idle Data: x55..55

 CLCW: x01040465
   .Type (1 bit): b0
   .Version Number (2 bits): b00
   .Status (3 bits): b000
   .COP Use (2 bits): b01
   .Virtual Channel ID (6 bits): b000001
   .Reserved (2 bits): b00
   .Flags (5 bits): b00000
   .FARM-B Counter (2 bits): b10
   .Reserved (1 bit): b0
   .Informed Value (1 byte): b65

 FECW (2 bytes): x7d4d

__________Test Procedure Finished__________
```

Source: from author.

5.2.3 CPDU Command Flow

The Command Pulse Distribution Unit is a simple onboard unit that is capable of providing direct access from the ground to equipment. The flow starts with a telecommand that embeds information about the pulse duration and the target output. A direct telecommand to the CPDU is created within a CLTU; then this data is transferred by the communication interface to the FPGA. The FPGA receives this command and generates an ACK (acknowledged) or NACK (not acknowledged) within a CADU, that is dispatched by the TM flow.

This flow was simulated with ModelSim, and the resulting waveform is presented in Figure 63. For the TC flow, the "*start_ seq_ ok_ s*" signal is active when the start sequence of the CLTU is identified and the signal "*stop_ seq_ ok_ s*" is activated by a stop sequence. In the specific example, the CLTU is composed of several blocks that are coded following the BCH(63,56) algorithm, then the signals "*status_ s*" and "*aff_ s*" are defined according to the situations presents in Table 10. Also, the signal "*sindrome_ s*" is responsible for the BCH algebraic decoding method known as syndrome decoding (MOON, 2005). For example, in the flow presented in Figure 63 two errors were found and corrected.

In addition, it is possible to identify the pulse of 13 ms in the signal "*pulses_ o(0)*". The "*ready_ i*" stands for the CPDU acknowledge value, "*done_ o*" is set to '0' when the layer is busy, "*cpdu_ size_ i*" represents 13×2^n ms pulse duration and "*cpdu_ addr_ i*" is the physical address of the output pin.

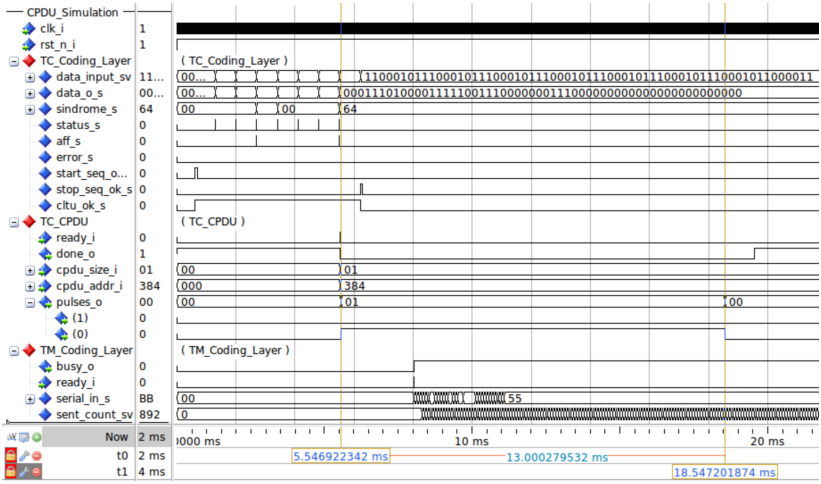
Table 10 – BCH cases according the *status* and *aff* signals.

Status	**Aff**	**Result**
0	X	An error has been found and can not be corrected
1	0	No error was found
1	1	An error was found and corrected

Source: adapted from (BEZERRA et al., 2010).

This test is performed by using the proposed test setup, where a CPDU command is sent through the serial communication interface, and the expected data is checked. The structure of the telecommand and the received ack/nack are respectively presented in the Figure 64, 65, 66 and 67.

Figure 63 – Results captured from the VHDL simulation of the CPDU procedure (ModelSim).



Source: from author.

Figure 64 – CPDU telecommand structure.

TC Transfer Frame Primary Header

Version Number 2 bits	Bypass Flag 1 bit	Control Command Flag 1 bit	Reserved Spare 2 bits	Spacecraft Identifier 10 bits	Virtual Channel Identifier 6 bits	Frame Length 10 bits	Frame Sequence Number 8 bits
b00	b0	0b	b00	b1111111111	b000000	b0000010001	xC8

Packet Primary Header

Packet Version Number 3 bits	Packet Identification			Packet Sequence Control		Packet Data Length 16 bits
	Packet Type 1 bit	Secondary Header Flag 1 bit	Application Process Identifier 11 bits	Sequence Flags 2 bits	Packet Sequence Count or Packet Name 14 bits	
b000	b1	b0	b00000000001	b11	b00000000000000	x0003

User Data Field

Source Data			CRC 16 bits	FEC 16 bits
CPDU Output Number 12 bits	Reserved 1 bit	Register Value 3 bits		
b000000000000	b0	b000	x1D0F	x9C07

Source: from author.

Figure 65 – CPDU command flow: telecommand to be sent.

```
lucas@cubos-licenses:~/Projects/payload-x/scripts$ python3 test_procedure.py

__________Payload-X Test Procedure__________

=> Choose the use case:
  1 - Destination: OBDH Config | Expected: Ack
  2 - Destination: OBDH Status | Expected: Status Report
  3 - Destination: UTMC CPDU   | Expected: Ack
  4 - Destination: UTMC CPDU   | Expected: NAck (code error  x08)
  5 - Destination: OBDH        | Expected: NAck (code error x080)
Commando: 3

=> Telecommand (TT&C -> Payload-X):

__Unlock Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF0007000030AA
 BCH Code Block (8 bytes): x4300000000000078
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__Set Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF000900820052
 BCH Code Block (8 bytes): xC8BFE4000000004A
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__CPDU Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x03FF0011C8C0108C
 BCH Code Block (8 bytes): x01C000000300002E
 BCH Code Block (8 bytes): x1D0F9C070000006C
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579
```

Source: from author.

In Figure 66, the value “01” in the “*Service Type ID*” identifies that a confirmation message and the value ‘01’ in the “*Message Type ID*” indicates that the message is ACK report.

Figure 66 – CPDU command flow: expected telemetry (first part).

```
=> Telemetry (Payload-X -> TT&C):

ASM: x1acffc1d

Transfer Frame Primary Header: x3ff300001800
  .Version Number (2 bits): b00
  .Virtual Channel ID (3 bits): b001
  .Operational Control Field Flag (1 bit): b1
  .Master Channel Frame Counter (1 byte): x00
  .Virtual Channel Frame Counter (1 byte): x00
  .Transfer Frame Data Field Status (2 bytes): x1800

Space Packet Primary Header: x0801c000000f
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Application Process ID (11 bits): b00000000001
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): xc000
  .Packet Lenght (2 bytes): x000f

Space Packet Secondary Header: x2001010000000000
  .TM PUS Version Number (4 bits): b0010
  .Spacecraft Time (4 bits): b0000
  .Service Type ID (1 byte): x01
  .Message Type ID (1 byte): x01
  .Message Type Counter (2 byte): x0000
  .Destination ID (2 bytes): x0000
  .Time (1 byte): x00

User Data : x1001c000

CRC (2 bytes): x0242
```

Source: from author.

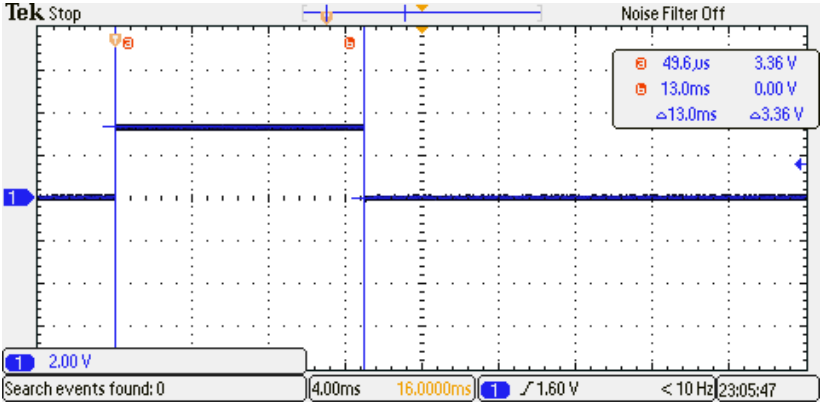
Figure 67 – CPDU command flow: expected telemetry (second part).

```
Idle Packet: x07ffc000043455..55
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Secondary Header Flag (1 bit): b0
  .Application Process ID (11 bits): b11111111111
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): x0004
  .Packet Lenght (2 bytes): x3455
  .Idle Data: x55..55

CLCW: x010004c9
  .Type (1 bit): b0
  .Version Number (2 bits): b00
  .Status (3 bits): b000
  .COP Use (2 bits): b01
  .Virtual Channel ID (6 bits): b000000
  .Reserved (2 bits): b00
  .Flags (5 bits): b00000
  .FARM-B Counter (2 bits): b10
  .Reserved (1 bit): b0
  .Informed Value (1 byte): bc9

FECW (2 bytes): xeb62

__________Test Procedure Finished__________
```

Source: from author.

As a result of this command, the CPDU output signal was acquired using the Tektronix MSO-2024B oscilloscope and Figure 68 presents the result in which it is possible to identify a pulse of 13 ms (since the time scale is of 4 ms) and two cursors identify the rising and the falling edge.

Figure 68 – Output 13ms pulse from a CPDU command.



Source: acquired with the Tektronix MSO-2024B.

5.2.4 Negative-acknowledgement Error Codes

The communication module applies a verification process when a direct telecommand (CPDU command) is received by the UTMC, and when the OBDH received a routed telecommand from the UTMC.

This process consists of a data consistency check related to the protocol structure. As a result of this verification, a report packet is generated containing the source command header, and when an error is detected, the packet consists in the annex of an error code with the source command header. Table 11 presents the UTMC possible values of the identification error field.

However, the OBDH only generates the confirmation packet under request or if a CRC error occurs, since it is not possible to define if this error was in the “Ack Flag” field. The packet follow the same pattern of those generated by the UTMC system and the possible values of the "Error Code" field are presented in Table 12.

Table 11 – Direct telecommand NACK error codes.

Error Code	Description
x00	Illegal APID
x01	Incomplete or invalid length packet
x02	Incorrect CRC
x03	Illegal packet type
x05	Illegal or inconsistent application data
x08	Invalid version number
x09	Invalid data field header flag

Source: from (BEZERRA et al., 2010).

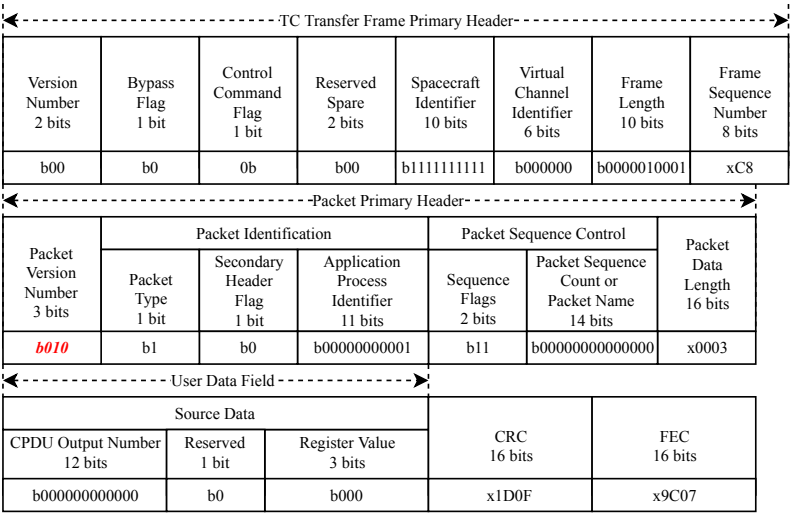
Table 12 – OBDH Verification NACK error codes.

Error Code	Description
x001	Incorrect CRC
x002	Invalid version number
x004	Illegal packet type
x008	Illegal or inconsistent application data
x010	Invalid sequence flag
x020	Invalid PUS version number
x040	Invalid ACK flag
x080	Illegal service type
x100	Illegal message subtype

Source: from author.

To test this verification functionality, a CPDU command with a known forced error in the "Version Number" field is sent to the communication module. The expected answer should contain in the "Error Code" field the '08'. Figure 69 presents the CPDU telecommand structure, where the inserted error is written in red in the "Packet Version Number" field. Figures 70, 71 and 72 presents the received report packet.

Figure 69 – CPDU telecommand structure with inseted error.



Source: from author.

Figure 70 – CPDU with error command flow: telecommand to be sent.

```
lucas@cubos-licenses:~/Projects/payload-x/scripts$ python3 test_procedure.py

__________Payload-X Test Procedure__________

=> Choose the use case:
  1 - Destination: OBDH Config | Expected: Ack
  2 - Destination: OBDH Status | Expected: Status Report
  3 - Destination: UTMC CPDU   | Expected: Ack
  4 - Destination: UTMC CPDU   | Expected: NAck (code error  x08)
  5 - Destination: OBDH        | Expected: NAck (code error x080)
Commando: 4

=> Telecommand (TT&C -> Payload-X):

__Unlock Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF0007000030AA
 BCH Code Block (8 bytes): x4300000000000078
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__Set Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF000900820052
 BCH Code Block (8 bytes): xC8BFE4000000004A
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__CPDU Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x03FF0011C8C0501E
 BCH Code Block (8 bytes): x01C000000300002E
 BCH Code Block (8 bytes): x1D0FEE1D000000FE
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579
```

Source: from author.

In Figure 71 the value ‘01’ in the “Service Type ID” identifies that a confirmation message and the value ‘02’ in the “Message Type ID” indicates that the message is a NACK report. The last byte of the “User Data” field is the error code from the CPDU unit.

Figure 71 – CPDU with error command flow: expected telemetry (first part).

```
=> Telemetry (Payload-X -> TT&C):

ASM: x1acffc1d

Transfer Frame Primary Header: x3ff300001800
  .Version Number (2 bits): b00
  .Virtual Channel ID (3 bits): b001
  .Operational Control Field Flag (1 bit): b1
  .Master Channel Frame Counter (1 byte): x00
  .Virtual Channel Frame Counter (1 byte): x00
  .Transfer Frame Data Field Status (2 bytes): x1800

Space Packet Primary Header: x0801c0000010
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Application Process ID (11 bits): b00000000001
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): xc000
  .Packet Lenght (2 bytes): x0010

Space Packet Secondary Header: x2001020000000000
  .TM PUS Version Number (4 bits): b0010
  .Spacecraft Time (4 bits): b0000
  .Service Type ID (1 byte): x01
  .Message Type ID (1 byte): x02
  .Message Type Counter (2 byte): x0000
  .Destination ID (2 bytes): x0000
  .Time (1 byte): x00

User Data : x5001c00008

CRC (2 bytes): xd742
```

Source: from author.

Figure 72 – CPDU with error command flow: expected telemetry (second part).

```
Idle Packet: x07ffc000043355..55
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Secondary Header Flag (1 bit): b0
  .Application Process ID (11 bits): b11111111111
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): xc000
  .Packet Lenght (2 bytes): x0433
  .Idle Data: x55..55

CLCW: x010004c9
  .Type (1 bit): b0
  .Version Number (2 bits): b00
  .Status (3 bits): b000
  .COP Use (2 bits): b01
  .Virtual Channel ID (6 bits): b000000
  .Reserved (2 bits): b00
  .Flags (5 bits): b00000
  .FARM-B Counter (2 bits): b10
  .Reserved (1 bit): b0
  .Informed Value (1 byte): bc9

FECW (2 bytes): xa033

__________Test Procedure Finished__________
```

Source: from author.

The second case, a telecomand using a wrong destination for the OBDH system is sent using the test setup. Figure 73 presents the structure of the telecommand. Figures 74, 75 and 76 presents the received report packet.

Figure 73 – OBDH command structure with wrong destination.

TC Transfer Frame Primary Header

Version Number 2 bits	Bypass Flag 1 bit	Control Command Flag 1 bit	Reserved Spare 2 bits	Spacecraft Identifier 10 bits	Virtual Channel Identifier 6 bits	Frame Length 10 bits	Frame Sequence Number 8 bits
b00	b0	0b	b00	b1111111111	b000001	b0000010100	x64

Packet Primary Header

Packet Version Number 3 bits	Packet Identification			Packet Sequence Control		Packet Data Length 16 bits
	Packet Type 1 bit	Secondary Header Flag 1 bit	Application Process Identifier 11 bits	Sequence Flags 2 bits	Packet Sequence Count or Packet Name 14 bits	
b000	b1	b1	b00000000000	b11	b00000000000000	x0006

TC Packet Secondary Header

TC PUS Version Number 4 bits	Ack Flags 4 bits	Message Type ID		Source ID 16 bits
		Service Type ID 8 bits	Message Subtype ID 8 bits	
b0010	b1000	x06	x07	x0000

CRC 16 bits	FEC 16 bits
xB99C	xCAD1

Source: from author.

Figure 74 – Command flow of a OBDH command with wrong destination: telecommand to be sent.

```
lucas@cubos-licenses:~/Projects/payload-x/scripts$ python3 test_procedure.py

__________Payload-X Test Procedure__________

=> Choose the use case:
  1 - Destination: OBDH Config | Expected: Ack
  2 - Destination: OBDH Status | Expected: Status Report
  3 - Destination: UTMC CPDU   | Expected: Ack
  4 - Destination: UTMC CPDU   | Expected: NAck (code error  x08)
  5 - Destination: OBDH        | Expected: NAck (code error x080)
Commando: 5

=> Telecommand (TT&C -> Payload-X):

__Unlock Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF04070000FA52
 BCH Code Block (8 bytes): xB20000000000008A
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__Set Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x33FF0409008200EE
 BCH Code Block (8 bytes): x64CD230000000014
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579

__OBDH Command__

Generated CLTU:
 Start Sequence (2 bytes): xEB90
 BCH Code Block (8 bytes): x03FF041464C0182A
 BCH Code Block (8 bytes): x00C000000628062A
 BCH Code Block (8 bytes): x070000B99CCAD18C
 Stop Sequence  (8 bytes): xC5C5C5C5C5C5C579
```

Source: from author.

In Figure 75 the value ‘01’ in the “Service Type ID” indicates a confirmation message, while the value ‘02’ in the “Message Type ID” indicates that the message is a NACK report. The 2 first bytes in the “User Data” are the error code and the last four bytes indicates the Space Packet header of the source telecommand.

Figure 75 – Command flow of a OBDH command with wrong destination: expected telemetry (first part).

```
=> Telemetry (Payload-X -> TT&C):

ASM: x1acffc1d

Transfer Frame Primary Header: x3ff304041800
  .Version Number (2 bits): b00
  .Virtual Channel ID (3 bits): b001
  .Operational Control Field Flag (1 bit): b1
  .Master Channel Frame Counter (1 byte): x04
  .Virtual Channel Frame Counter (1 byte): x04
  .Transfer Frame Data Field Status (2 bytes): x1800

Space Packet Primary Header: x0800c0040010
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Application Process ID (11 bits): b00000000000
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): xc004
  .Packet Lenght (2 bytes): x0010

Space Packet Secondary Header: x2001020004000000000080
  .TM PUS Version Number (4 bits): b0010
  .Spacecraft Time (4 bits): b0000
  .Service Type ID (1 byte): x01
  .Message Type ID (1 byte): x02
  .Message Type Counter (2 byte): x0004
  .Destination ID (2 bytes): x0000
  .Time (1 byte): x00
  .Spare (1 byte): x00

User Data : x00801800c000

CRC (2 bytes): xd9d7
```

Source: from author.

Figure 76 – Command flow of a OBDH command with wrong destination: expected telemetry (second part).

```
Idle Packet: x07ffc000043155..55
  .Packet Version Number (3 bits): b000
  .Packet Type (1 bit): b0
  .Secondary Header Flag (1 bit): b0
  .Application Process ID (11 bits): b11111111111
  .Sequence Flag (2 bits): b11
  .Sequence Count (14 bits): xc000
  .Packet Lenght (2 bytes): x0431
  .Idle Data: x55..55

CLCW: x01040465
  .Type (1 bit): b0
  .Version Number (2 bits): b00
  .Status (3 bits): b000
  .COP Use (2 bits): b01
  .Virtual Channel ID (6 bits): b000001
  .Reserved (2 bits): b00
  .Flags (5 bits): b00000
  .FARM-B Counter (2 bits): b10
  .Reserved (1 bit): b0
  .Informed Value (1 byte): b65

FECW (2 bytes): xbc9c

__________Test Procedure Finished__________
```

Source: from author.

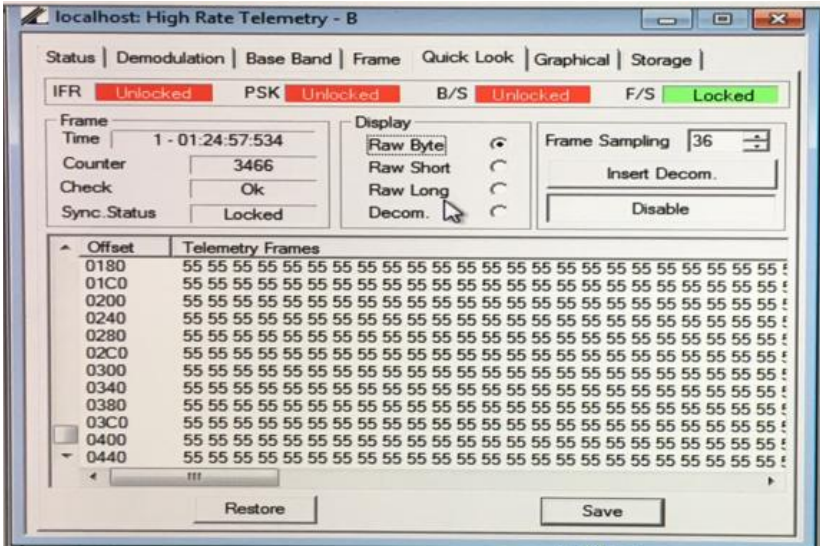
5.3 CORTEX RESULTS

During the development of this work, it was possible to have access to Cortex equipment. However, the available time was limited, and the OBDH system was not ready at that moment. Therefore the test focused on a continuous transmission in the telemetry flow using the Synchronous Serial Interface (SSI) as communication interface, with a data rate of 500 kbps. The Cortex equipment is capable of verifying the consistency of the data and performs the decoding using the Reed-Solomon algorithm.

Considering the information presented above, the embedded application in the development kit was only the UTMC, and it was configured for a continuous transmission mode of idle packets. Two different approaches were tested. For the first one, the telemetry frame was uncoded and, in the second one, the Reed-Solomon algorithm was applied.

Fig. 77 shows a received idle packet consisting of several 55_{16} (55 in hexadecimal) bytes using the embedded software provided by the equipment. The counter is at 3466 showing the compatibility of the packets sent by the UTMC, since wrong data would reset the counter.

Figure 77 – Packet received in CORTEX.



Source: from Cortex equipment.

The logs of the performed tests were saved using the TCP/IP interface provided by the equipment. To represent the acquired results, Table 13 presents the report for un-coded telemetry, while Table 14 presents the telemetry by using Reed-Solomon coding. In both cases, the test was performed several times, and it was stopped to start another one always when the Sequence Counter was higher than 100,000 frames. This means that the implementation was able to send the correct pattern without errors, at least more than 100,000 times.

Table 13 – Cortex report of the un-coded telemetry frame.

Cortex Report - Un-coded Telemetry Frame	
Telemetry Frame	1acffc1d3ff395951ffe55....5501002000 a8ab
Cortex Local Time	0:31:37.000431
Sequence Counter	103,828
Frame Check and RS Status	[2] CRC verification ON
Frame Check Result	[0] Frame Check OK
Frame Sync Status	[1] Frame sync. ON, LOCK
Bit Slip Status	[0] No bit flip
TM Delay	0
Frame Length (bytes)	1119
Sync Word Length (bytes)	32

Source: based on the results from Cortex.

Table 14 – Cortex report of the Reed-Solomon telemetry frame.

Cortex Report	
Telemetry Frame	1acffc1d3ff367671ffe55....5501002000 040d7be5a4f264d43788b929e1bfe233 cff131b49a01 79c67973749099038426 12a8560dc34e43291232ae8f32c20996 ca287ca3a0c032716319a53ad4a72500 2cfdcace0af024ff265e3f8fefa5591d084 12c4375c5cb65caa276e86c7ef9287369 4b927c10f5199cdf986b101440e085772 edeb2204eba81368c04d388a93db9f87f 3afd18f3f9ff4d284866ed415391409c5d ab2f1e7f9fedc89ef4253b272305
Corte Local Time	1:05:58.000104
Sequence Counter	105,257
Frame Check and RS Status	[3] RS decoder; CRC verification ON
Frame Check Result	[0] Frame Check OK
Frame Sync Status	[1] Frame sync. ON, LOCK
Bit Slip Status	[0] No bit flip
TM Delay	0
Frame Lenght (bytes)	1279
Sync Word Lenght (bytes)	32

Source: based on the results from Cortex.

5.4 SYNTHESIS RESULTS

Taking into account the features available in the Brave FPGA, Table 15 presents the results obtained from the NanoXplore synthesis tool. The reference version for the presented results uses BCH coding in the telecommand stream, and the telemetry is generated without codes for error detection and correction.

Moreover, a Static Time Analysis was performed to check if the implementation is capable of running at required frequency. Table 16 presents the results. The required frequency for this design is 25 MHz, and the implementation could run using a maximum frequency of 34.301 MHz. It is important to underline that there is no negative slack.

Table 15 – Synthesis result: resource usage.

Instance	**Count**	**Total Available**	**Usage (%)**
4-LUT	7861	34272	23
DFF	3599	32256	12
XLUT	265	2016	14
Carry	2077	8064	26
Register File Block	0	168	0
Clock Switch	0	336	0
DSP Block	0	112	0
Memory Block	5	56	9
WFG	1	32	4
PLL	0	4	0

Source: based on the results from NanoXplore Synthesis tool.

Table 16 – Static Time Analisys results.

Static Time Analisys Report			
Source	***Target***	***Hold/Removal Summary***	
		Slack	*Min. Data Arrive Time*
Input	Clock (falling)	-	1.095ns
Input	Clock (rising)	-	3.022ns
Clock (falling)	Clock (falling)	1.248ns	1.248ns
Clock (falling)	Clock (rising)	21.250ns	1.250ns
Clock (rising)	Clock (falling)	21.250ns	1.250ns
Clock (rising)	Clock (rising)	1.243ns	1.243ns
Clock (rising)	Output	-	7.923ns
Source	***Target***	***Setup/Recovery Summary***	
		Slack	*Min. Data Arrive Time*
Input	Clock (falling)	-	6.643ns
Input	Clock (rising)	-	10.097ns
Clock (falling)	Clock (falling)	28.753ns	11.247ns
Clock (falling)	Clock (rising)	5.423ns	14.577ns
Clock (rising)	Clock (falling)	6.320ns	13.680ns
Clock (rising)	Clock (rising)	23.302ns	16.698ns
Clock (rising)	Output	-	15.896ns
Frequency		*Required*	*Maximum*
		25.000 MHz	34.301 MHz

Source: based on the results from NanoXplore Synthesis tool.

6 CONCLUSION

This work presented the development of a communication module for nanosatellites based on a hardware application. The main focus of this work was the construction of an architecture based on the CCSDS recommendations. The implemented architecture is composed of a Unit of Telemetry and Telecommands (UTMC) and an OBDH (On-Board Data Handling). The UTMC works in the scope of the Data Link Layer described in the CCSDS 130.0-G-3 - Overview of Space Communications Protocols, since this unit handles the data structure presented by the recommendations that embrace this layer and performs the data coding and decoding. The developed OBDH performs the Network Layer and uses the Space Packet Protocol. The OBDH was implemented to be able to validate the received telecommands and generate status and scientific data to be sent within the telemetry framework.

The proposed architecture represents a trade off between meeting the specific demands of the case study and being generic enough to be applied in other scenarios. The implementation was first simulated through the tool ModelSim and next it was physically tested with the help of the NanoXplore development board.

It is worth noting that after the implementation of the physical synthesis, it was verified that the FPGA could support not only the current implementation, but also additional control modules. Thereby, aiming at the development of a complete control system for a CubeSat mission within an FPGA, a future expansion of the system will be achieved possible thank to more capable FPGAs that the company NanoXplore plans to produce.

Since the presented results are satisfactory for the proposed application, it is possible to conclude that the work achieved the intended objectives. It is important to note that the work has continuity in the scope of the FloripaSat-I mission since it is embedded in the developed architecture for Payload-X, which is the result of team work within the research group.

6.1 ACADEMIC PRODUCTION

In the framework of this project, two research articles were submitted and accepted as regular papers for oral presentation in international conference of the domain.

1. LUZA, L. M.; RIGO, C. A.; TRAMONTIN, E. D.; MARTINS, V.; MARTINEZ, S. V; SLONGO, L. K.; SEMAN, L. O.; DILILLO, L.; BEZERRA, E. A. Enabling deep-space CubeSat missions through state-of-the-art radiation-hardned technologies. In: *III IAA Latin American CubeSat Workshop (LACW-IAA 2018)*, Ubatuba, São Paulo, 2018.

2. RIGO, C. A.; LUZA, L. M.; TRAMONTIN, E. D.; MARTINS, V.; MARTINEZ, S. V; SLONGO, L. K.; SEMAN, L. O.; DILILLO, L.; VARGAS, F. L.; BEZERRA, E. A. A Fault-Tolerant Reconfigurable Platform for Communication Modules of Satellites *The 20th IEEE Latin-American Test Symposium (LATS 2019)*, Santiago, Chile, 2019.

6.2 FUTURE WORKS

During the development of this work the following possibilities of future work have been identified:

- Validation of the system using the Payload-X architecture. A development kit was used to test the proposed communication module, since the access to a Payload-X board was yet not possible until the end of this work. Thus, there is still a need to test the final implementation with all the components interconnected.

- Implementation of the SRAM-based controller monitor. Once the memory controller will be avaiable by a partner of FloripaSat project, the integration with the OBDH described in this work must be performed;

- PUS Services implementation. The proposed work used the ECSS PUS frame structure to define custom services using the OBDH system. For future works, it would be interesting the addition of the default services proposed by the recommendation since it increases the possibility to reuse the architecture;

- Addition of an on-chip bus. The actual work can use two different protocols to do the communication between the UTMC and OBDH, the Synchronous Serial Interface (SSI) and Universal Asynchronous Receiver/Transmitter (UART). However, an on-chip interconnection bus as Advanced Microcontroller Bus Archi-

tecture (AMBA) could provide a higher frequency communication between the entities.

REFERENCES

ARNOLD, S. S.; NUZZACI, R.; GORDON-ROSS, A. Energy budgeting for CubeSats with an integrated FPGA. In: *2012 IEEE Aerospace Conference*. [S.l.]: IEEE, 2012. p. 1–14.

BATTEZZATI, N. et al. Application-oriented SEU sensitiveness analysis of Atmel rad-hard FPGAs. In: *2009 15th IEEE International On-Line Testing Symposium*. Sesimbra, PT: IEEE, 2009. p. 89–94.

BEECH, W. A.; NIELSEN, D. E.; TAYLOR, J. *AX.25 Link Access Protocol for Amateur Packet Radio*. 1998. 1–133 p. Tucson Amateur Packet Radio Corporation. Rev. 2.2.

BETZ, V.; ROSE, J.; MARQUARDT, A. *Architecture and CAD for deep-submicron FPGAs*. Massachusetts, US: Kluwer Academic Publishers, 1999. 247 p.

BEZERRA, E.; GOUGH, M. A guide to migrating from microprocessor to FPGA coping with the support tool limitations. *Microprocessors and Microsystems*, Elsevier, v. 23, n. 10, p. 561–572, mar 2000.

BEZERRA, E. A. et al. An adaptive communications module for on-board computers of satellites. In: *2010 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2010*. Anaheim, USA: IEEE, 2010. p. 317–324.

BEZERRA, E. A.; AZEVEDO, L. R.; SILVA, E. M. da. Dependability Issues of INPE's ESA/CCSDS Telecommand/Telemetry Subsystem. In: *2011 Fifth Latin-American Symposium on Dependable Computing Workshops*. [S.l.]: IEEE, 2011. p. 58–63.

BEZERRA, E. A.; LETTNIN, D. V. *Synthesizable VHDL Design for FPGAs*. 1st. ed. Cham: Springer International Publishing, 2014. 157 p.

BHASKER, J. *A VHDL primer*. 3rd. ed. New Jersey, US: Prentice Hall PTR, 1999. 373 p.

BIRKELAND, R.; LØFALDLI, A. Implementation of a Software Defined Radio Prototype Ground Station for CubeSats Coastal and Arctic Maritime Operations and Surveillance View project Sky-fi View project Implementation of a Software Defined Radio Prototype

Ground Station for CubeSats. In: *The 4S Symposium*. Malta, IT: [s.n.], 2016.

CALIN, T.; NICOLAIDIS, M.; VELAZCO, R. Upset hardened memory design for submicron CMOS technology. *IEEE Transactions on Nuclear Science*, v. 43, n. 6, p. 2874–2878, 1996.

CCSDS. *CCSDS 133.0-B-1 - Space Packet Protocol*. [S.l.], 2003. <Avaiable in: https://public.ccsds.org/Pubs/133x0b1c2.pdf>.

CCSDS. *CCSDS 232.1-B-2 - Communications Operation Procedure-1*. [S.l.], 2010. <Avaiable in: https://public.ccsds.org/Pubs/232x1b2e1.pdf>.

CCSDS. *CCSDS 130.0-G-3 - Overview of Space Communications Protocols*. [S.l.], 2014. <Avaible in: https://public.ccsds.org/Pubs/130x0g3.pdf>.

CCSDS. *CCSDS 132.0-B-2 - TM Space Data Link Protocol*. [S.l.], 2015. <Avaiable in: https://public.ccsds.org/Pubs/132x0b2.pdf>.

CCSDS. *CCSDS 232.0-B-3 - TC Space Data Link Protocol*. [S.l.], 2015. <Avaiable in: https://public.ccsds.org/Pubs/232x0b3.pdf>.

CCSDS. *CCSDS 131.0-B-3 - TM Synchronizations and Channel Coding*. [S.l.], 2017. <Avaiable in: https://public.ccsds.org/Pubs/131x0b3e1.pdf>.

CCSDS. *CCSDS 231.0-B-3 - TC Synchronizations and Channel Coding*. [S.l.], 2017. <Avaiable in: https://public.ccsds.org/Pubs/231x0b3.pdf>.

CCSDS. *CCSDS.org - The Consultative Committee for Space Data Systems (CCSDS)*. 2019. Available in: https://public.ccsds.org/default.aspx. Access in: January 13, 2019.

Censin Technology. *Censin Technology*. 2019. Available in: http://censintechnology.com/Cortex-CRT/. Access in: January 13, 2019.

CHEN, M.; ORAILOGLU, A. Improving Circuit Robustness with Cost-Effective Soft-Error-Tolerant Sequential Elements. In: *16th Asian Test Symposium (ATS 2007)*. [S.l.]: IEEE, 2007. p. 307–312.

COMPTON, K.; HAUCK, S. Reconfigurable Computing: A Survey of Systems and Software. *ACM Computing Surveys*, v. 34, p. 171–210, 2002.

COOK, K. L. B. The ITAR and you - what you need to know about the International Traffic in Arms Regulations. In: *2010 IEEE Aerospace Conference*. [S.l.]: IEEE, 2010. p. 1–12.

DANILOV, I. et al. On board electronic devices safety provided by DICE-based Muller C-elements. *Acta Astronautica*, v. 150, p. 28–32, sep 2018.

DOWD, M. *How Rad Hard Do You Need? The Changing Approach To Space Parts Selection?* San Diego, US: [s.n.], 2003. White Paper, Maxwell Technologies.

DYER, C. Radiation effects on spacecraft & aircraft. In: *Solspa 2001, Proceedings of the Second Solar Cycle and Space Weather Euroconference*. Vico Equense, IT: [s.n.], 2002. v. 477, p. 505–512.

ECSS. *ECSS-E-ST-70-41C – Telemetry and telecommand packet utilization (15 April 2016)*. [S.l.], 2016. <Avaible in: https://ecss.nl/standard/ecss-e-st-70-41c-space-engineering-telemetry-and-telecommand-packet-utilization-15-april-2016/>.

ENGBERG, B.; OTA, J.; SUCHMAN, J. The opal satellite project: Continuing the next generation of small satellite development. In: *Proceedings of the 9th Annual AIAA/US Conference on Small Satellites*. [S.l.: s.n.], 1995. p. 19–22.

European Space Agency. *OPS-SAT*. 2019. Avaiable in: https://www.esa.int/Our_Activities/Operations/OPS-SAT. Access in: January 21, 2019.

EVANS, D.; MERRI, M. OPS-SAT: A ESA nanosatellite for accelerating innovation in satellite control. In: *SpaceOps 2014 Conference*. Reston, Virginia: American Institute of Aeronautics and Astronautics, 2014. ISBN 978-1-62410-221-9.

FAROOQ, U.; MARRAKCHI, Z.; MEHREZ, H. *Tree-based Heterogeneous FPGA Architectures*. 1. ed. New York, NY: Springer-Verlag New York, 2012. 188 p.

FLEETWOOD, D. M.; WINOKUR, P. S.; DODD, P. E. An overview of radiation effects on electronics in the space telecommunications environment. *Microelectronics Reliability*, v. 40, p. 171–26, 2000.

GARCIA, P. et al. An Overview of Reconfigurable Hardware in Embedded Systems. *EURASIP Journal on Embedded Systems*, v. 2006, n. 1, p. 56320, 2006.

HEIDT, H. et al. Cubesat: A new generation of picosatellite for education and industry low-cost space experimentation. In: *14th Annual AIAA/USU Conference on Small Satellites*. Utah, US: [s.n.], 2000. p. 1–19.

HUANG, Z.; LIANG, H. A New Radiation Hardened by Design Latch for Ultra-Deep-Sub-Micron Technologies. In: *2008 14th IEEE International On-Line Testing Symposium*. [S.l.]: IEEE, 2008. p. 175–176.

IVANOV, A. et al. CubETH: low cost GNSS space experiment for precise orbit determination. In: *The 4S Symposium*. Majorca, ES: [s.n.], 2014. p. 13.

JOHNSTON, A. H. Space Radiation Effects and Reliability Considerations for Micro- and Optoelectronic Devices. *IEEE Transactions on Device and Materials Reliability*, v. 10, n. 4, p. 449–459, 2010.

KLEINSCHRODT, A. et al. Advances in Modulation and Communication Protocols for Small Satellite Ground Stations. In: *68th International Astronautical Congress*. Adelaide, AU: [s.n.], 2017.

KLOFAS, B.; ANDERSON, J.; LEVEQUE, K. A Survey of CubeSat Communication Systems. In: *Proceedings in 5th Annual CubeSat Developers Workshop*. San Luis Obispo, US: [s.n.], 2008.

LUZA, L. M. et al. Enabling deep-space CubeSat missions through state-of-the-art radiation-hardened technologies. In: *III IAA Latin American CubeSat Workshop (LACW-IAA 2018)*. Ubatuba, BR: [s.n.], 2018.

MEVADA, J. et al. Design and implementation of a robust downlink communication system for nanosatellites. In: *2015 International Conference on Space Science and Communication (IconSpace)*. IEEE, 2015. p. 164–169. ISBN 978-1-4799-1940-6. <http://ieeexplore.ieee.org/document/7283827/>.

MOON, T. K. *Error Correction Coding*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2005.

MURI, P.; MCNAIR, J. A Survey of Communication Sub-systems for Intersatellite Linked Systems and CubeSat Missions. *Journal of Communications*, v. 7, n. 4, p. 290—-308, 2012.

NANOXPLORE. NX1H35S Datasheet. *NanoXplore*, v. 1.6, n. March, p. 1–70, 2018.

NANOXPLORE. NxMap User Guide. *NanoXplore*, v. 2.9.1, n. March, p. 1–120, 2018.

NASA CUBESAT LAUNCH INITIATIVE. *CubeSat 101 : Basic Concepts and Processes for First-Time CubeSat Developers*. [S.l.], 2017.

NASEER, R.; DRAPER, J. DF-DICE: a scalable solution for soft error tolerant circuit design. In: *2006 IEEE International Symposium on Circuits and Systems*. [S.l.]: IEEE, 2006. p. 4.

NICOLAIDIS, M. (Ed.). *Soft Errors in Modern Electronic Systems*. Boston, US: Springer US, 2011. 316 p.

PETERSEN, E. *Single Event Effects in Aerospace*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2011. 520 p.

PUIG-SUARI, J.; TURNER, C.; AHLGREN, W. Development of the standard CubeSat deployer and a CubeSat class PicoSatellite. In: *2001 IEEE Aerospace Conference Proceedings*. Big Sky, US: [s.n.], 2001. p. 1/347–1/353.

RAZZAGHI, E. *Design and Qualification of On-Board Computer for Aalto-1 CubeSat*. Tese (Doutorado) — Luleå University of Technology, Espoo, FI, 2012.

RIGO, C. A. *Projeto de Placas de Circuito Impresso com FPGAs para uso em ambiente espacial*. Dissertação (Master in Electrical Engineering) — Universidade Federal de Santa Catarina, Florianópolis, 2019.

RIGO, C. A. et al. Fault-Tolerant Reconfigurable Platform for Communication Modules of Satellites. In: *The 20th IEEE Latin-American Test Symposium (LATS 2019)*. Santiago, CL: [s.n.], 2019.

SCHWANK, J. R. Basic mechanisms of radiation effects in the natural space environment. In: *Proc. 1994 NSREC Short Cours.* United States: [s.n.], 1994.

SELČAN, D.; KIRBIŠ, G.; KRAMBERGER, I. FPGA-Based CCSDS Compliant Miniaturized Satellite Communication Stack. *IFAC-PapersOnLine*, Elsevier, v. 48, n. 10, p. 28–33, jan 2015. ISSN 2405-8963.

Shu Lin; COSTELLO, D. J.; MILLER, M. J. Automatic-repeat-request error-control schemes. *IEEE Communications Magazine*, v. 22, n. 12, p. 5–17, 1984.

SKAGMO, J. P. *NGHam protocol*. 2014. Avaiable in: https://github.com/skagmo/ngham. Access in: January 17, 2019.

SLONGO, L. K. et al. The floripa-sat experience: mission progress and satellite‘s development. In: *II IAA Latin American CubeSat Workshop (LACW-IAA 2016)*. Brasília, BR: [s.n.], 2016.

SURESH, S. V. S. et al. Design of command and Data Management System for IITMSAT. In: *2015 International Conference on Space Science and Communication (IconSpace)*. IEEE, 2015. p. 11–16. ISBN 978-1-4799-1940-6. <http://ieeexplore.ieee.org/document/7283815/>.

THE CUBESAT PROGRAM. *CubeSat Design Specification*. 2014. California Polytechnic State University. Rev. 13.

TRAMONTIN, E. D. *Estratégia para atualização remota de sistemas computacionais embarcados em satélites: um estudo de caso com o nanossatélite FloripaSat-I*. Dissertação (Master in Electrical Engineering) — Universidade Federal de Santa Catarina, Florianópolis, 2018.

TRISAT. *Small Slovenian satellite with great potential*. 2019. Avaiable in: http://www.trisat.um.si/. Access in: January 20, 2019.

VELAZCO, R.; FOUILLAT, P.; REIS, R. (Ed.). *Radiation Effects on Embedded Systems*. Dordrecht, NL: Springer Netherlands, 2007. 269 p.

VILLA, P. et al. A complete cubesat mission: The floripasat experience. In: *I IAA Latin American CubeSat Workshop (LACW-IAA 2014)*. Brasília, BR: [s.n.], 2014.

WAIN, R. et al. *An overview of FPGAs and FPGA programming; Initial experiences at DaresburyNo Title*. 2006. Computational Science and Engineering Department, CCLRC Daresbury Laboratory.

WANG, J. et al. SRAM based re-programmable FPGA for space applications. v. 46, n. 6, p. 1728–1735, 1999.

XILINX; INC. *Xilinx XST User Guide - 10.1*. [S.l.], 2002. 600 p. <Avaiable in: https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/books/docs/xst/xst.pdf>.

YUE, G. et al. A Single Event Latch-up protection method for SRAM FPGA. In: *2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*. [S.l.]: IEEE, 2017. p. 332–336.

ZEIGER, F.; SCHMIDT, M.; SCHILLING, K. A Flexible Extension for Pico-Satellite Communication Based on Orbit Operation Results of UWE-1. In: *57th International Astronautical Congress*. Reston, Virigina: American Institute of Aeronautics and Astronautics, 2006.

ZHANG, S.; LIE, H. Synthetical analysis on space radiation tolerance techniques in ASICs and FPGAs. In: *2011 International Conference on System science, Engineering design and Manufacturing informatization*. [S.l.]: IEEE, 2011. p. 305–310.

APPENDIX A – FSM VHDL Coding Example

FSM With Three Processes VHDL Coding Example.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm_3 is
  port (clk, reset, input  : in std_logic;
        output             : out std_logic
  );
end entity;

architecture behavioral of fsm_3 is
    type  state_type is (s1,s2,s3,s4);
    signal state, next_state: state_type;
begin
    process1 : process (clk, reset)
        begin
            if (reset ='1') then
                state <= s1;
            elsif (clk='1' and clk'Event) then
                state <= next_state;
            end if;
    end process process1;

    process2 : process (state, input)
        begin
        case state is
            when s1 =>
                if (input = '1') then
                    next_state <= s2;
                else
                    next_state <= s3;
                end if;
            when s2 =>
                next_state <= s4;
            when s3 =>
                next_state <= s4;
            when s4 =>
                next_state <= s1;
        end case;
    end process process2;

    process3 : process (state)
        begin
        case state is
```

```
            when s1 =>
                output <= '1';
            when s2 =>
                output <= '1';
            when s3 =>
                output <= '0';
            when s4 =>
                output <= '0';
        end case;
    end process process3;

end behavioral;
```

Source: (XILINX; INC, 2002).