

6-2023

DESIGN MODULAR COMMAND AND DATA HANDLING SUBSYSTEM HARDWARE ARCHITECTURES

Abdullah Alsalmani

Follow this and additional works at: https://scholarworks.uaeu.ac.ae/all_theses



Part of the [Astrophysics and Astronomy Commons](#), and the [Atomic, Molecular and Optical Physics Commons](#)

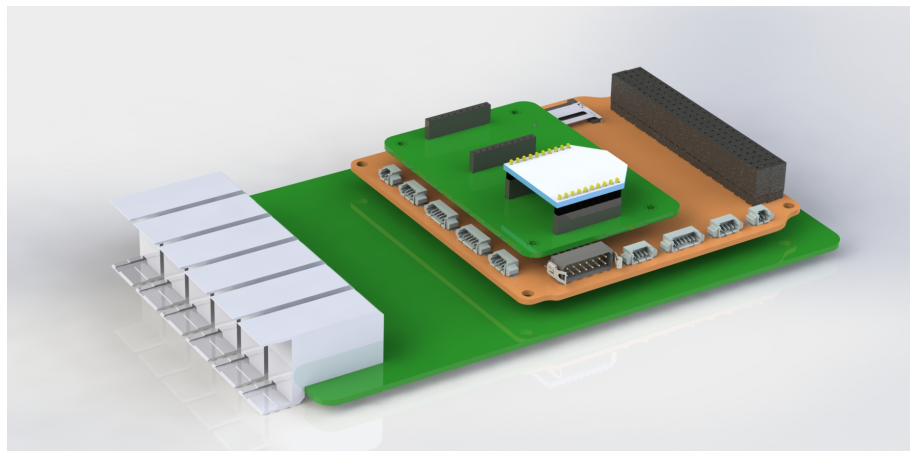
MASTER THESIS NO. 2023: 61

College of Science

Department of Physics

**DESIGN MODULAR COMMAND AND DATA HANDLING SUBSYSTEM
HARDWARE ARCHITECTURES**

Abdullah Alsalmani



June 2023

United Arab Emirates University
College of Science
Department of Physics

**DESIGN MODULAR COMMAND AND DATA HANDLING
SUBSYSTEM HARDWARE ARCHITECTURES**

Abdullah Alsalmami

This thesis is submitted in partial fulfillment of the requirements for the
degree of Master of Science in Space Science

June 2023

United Arab Emirates University Master Thesis

2023: 61

Cover: Image regarding the proposed modular architecture in this thesis.

(Photo: By Abdullah Alsalmani)

© 2023 Abdullah Alsalmani, Al Ain, UAE

All Rights Reserved

Print: University Print Service, UAEU 2023

Declaration of Original Work

I, Abdullah Alsalmani, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this thesis entitled "*Design Modular Command and Data Handling Subsystem Hardware Architectures*", hereby, solemnly declare that this thesis is my own original research work that has been done and prepared by me under the supervision of Dr. Abdul-Halim Jallad, in the College of Science at the UAEU. This work has not previously been presented or published, or formed the basis for the award of any academic degree, diploma or a similar title at this or any other university. Any materials borrowed from other sources (whether published or unpublished) and relied upon or included in my thesis have been properly cited and acknowledged in accordance with appropriate academic conventions. I further declare that there is no potential conflict of interest with respect to the research, data collection, authorship, presentation and/or publication of this thesis.

Student's Signature:  _____

Date: 11.06.2023

Approval of the Master Thesis

This Master Thesis is approved by the following Examining Committee Members:

1) Advisor (Committee Chair): Abdul-Halim Jallad

Title: Assistant Professor

Department of Electrical and Communication Engineering

College of Engineering

Signature -  — Date 12.06.2023

2) Member: Mohammad Naouss

Title: Professor

Department of Computer and Network Engineering

College of Information Technology

Signature -  — Date 12.06.2023

3) Member (External Examiner): Mohammed Ghazal

Title: Professor

Department of Electrical, Computer, and Biomedical Engineering

College of Engineering

Institution: Abu Dhabi University, UAE

Signature - *M Ghazal* — Date 12.06.2023

This Master Thesis is accepted by:

Dean of the College of Science: Professor Maamar Benkraouda

Signature Maamar Benkraouda

Date 18/08/2023

Dean of the College of Graduate Studies: Professor Ali Al-Marzouqi

Signature Ali Hassan

Date 18/08/2021

Abstract

Over the past few years, On-Board Computing Systems for satellites have been facing a limited level of modularity. Modularity is the ability to reuse and reconstruct the system from a set of predesigned units, with minimal additional engineering effort. CDHS hardware systems currently available have a limited ability to scale with mission needs. This thesis addresses the integration of smaller form factor CDHS modules used for nanosatellites with the larger counterparts that are used for larger missions. In particular, the thesis discusses the interfacing between Modular Computer Systems based on Open Standard commonly used in large spacecrafts and PC/104 used for nanosatellites. It also aims to create a set of layers that would represent a hardware library of COTS-like modules. At the beginning, a review of related and previous work has been done to identify the gaps in previous studies and understand more about Modular Computer Systems based on Open Standard commonly used in large spacecrafts, such as cPCI Serial Space and SpaceVPX. Next, the design requirements have been set to achieve this thesis objectives, which included conducting a prestudy of system alternatives before creating a modular CDHS hardware architecture which was later tested. After, the hardware suitable for this architecture based on the specified requirements was chosen and the PCB was designed based on global standards. Later, several functional tests and communication tests were conducted to assess the practicality of the proposed architecture. Finally, thermal vacuum testing was done on one of the architecture's layers to test its ability to withstand the space environment, with the aim to perform the vibration testing of the full modular architecture in the future. The aim of this thesis has been achieved after going through several tests, comparing between interfaces, and understanding the process of interfacing between different levels of the CDHS. The findings of this study pave the way for future research in the field and offer valuable insights that could contribute to the development of modular architectures for other satellite subsystems.

Keywords: Command and Data Handling Systems, Modular Architectures, Satellites, Space.

Title and Abstract (in Arabic)

اتصميم النظام الفرعي لمعالجة الأوامر و البيانات المرن.

الملخص

خلال السنوات القليلة الماضية، واجهت أنظمة الحوسبة الداخلية المستخدمة في كمبيوترات القمر الصناعي مستوى محدود من المرونة. المرونة هي القدرة على إعادة استخدام النظام وإعادة تركيبه من مجموعة من الوحدات المصممة مسبقًا مع جهد هندسي إضافي بسيط. تواجه أنظمة الكمبيوتر الداخلية المتاحة حاليًا لوحات المعالجة المركزية والذواكر والتخزين (CDHS) التابعة لكمبيوتر القمر الصناعي محدودية في القدرة على التوسع لتلبية احتياجات المهمة. يتناول هذا الأطروحة تكامل وحدات النظام الفرعي لمعالجة الأوامر و البيانات للقمر الصناعي ذات حجم أصغر والتي تستخدم في الأقمار الصناعية الصغيرة مع نظرائها ذات الحجم الأكبر المستخدمة في المهام الأكبر. تتناول هذه الأطروحة بشكل خاص التواصل بين أنظمة الحاسب المعيارية المتعددة التي تعتمد على المعايير المفتوحة والتي تستخدم بشكل شائع في المركبات الفضائية الكبيرة، وبين نظام الاتصال طرفي - ١٠٤ (PC/104) المستخدم في الأقمار الصناعية الصغيرة. كما تهدف الأطروحة إلى إنشاء مجموعة من الطبقات التي تمثل مكتبة الأجهزة المعدة مسبقًا، المتاحة بالأسواق. وفي البداية، تم إجراء مراجعة للأعمال ذات الصلة والأبحاث السابقة لتحديد الثغرات في الدراسات السابقة وفهم المزيد حول أنظمة الكمبيوتر المعيارية المتعددة التي تستخدم عادة في المركبات الفضائية الكبيرة، مثل سي بي سي اي الخاص بالمشاريع الفضائية (cPCI Serial Space) و في بي اكس الخاص بالمشاريع الفضائية (SpaceVPX). بعد ذلك، تم تعيين متطلبات التصميم لتحقيق أهداف هذه الأطروحة، والتي تضمنت إجراء دراسة مسبقة لبدايل النظام قبل إنشاء بنية أجهزة كمبيوتر القمر الصناعي المعيارية

التي تم اختبارها لاحقًا. بعد ذلك، تم اختيار الأجهزة المناسبة لهذه البنية بناءً على المتطلبات المحددة وتم تصميم لوحة الكترونية مطبوعة وفقًا للمعايير العالمية. في وقت لاحق، تم إجراء العديد من الاختبارات الوظيفية واختبارات الاتصال للوصول إلى التطبيق العملي للبنية المقترحة. أخيرًا، تم إجراء اختبار الفراغ الحراري على إحدى طبقات العمارة لاختبار قدرتها على تحمل البيئة الفضائية، بهدف إجراء اختبار الاهتزاز للبنية المعيارية الكاملة في المستقبل. تم تحقيق الهدف من هذه الرسالة بعد إجراء العديد من الاختبارات، والمقارنة بين الواجهات، وفهم عملية التفاعل بين المستويات المختلفة لكمبيوتر القمر الصناعي. هذا سيفتح الأبواب لمزيد من البحث في هذا المجال والدروس المستفادة يمكن أن تساعد في إنشاء معماريات معيارية لبقية أنظمة الأقمار الصناعية الفرعية.

مفاهيم البحث الرئيسية: كمبيوتر القمر الصناعي، الأنظمة المرنة، الأقمار الصناعية، الفضاء.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Abdul-Halim Jallad for his continuous support during my MSc study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research, complete and write this thesis. I could not have imagined having better mentor for my MSc study.

I would also like to thank all the faculty members of College of Science at UAEU for their support and cooperation during my study. My thanks should go to my colleagues at NSSTC and friends, especially Teana Alniwairi, Mohamed Alamim, Mohammed Almehezi, and AbuBaker Alzubaidi for their support, discussions, and suggestions.

This work was generously supported by the UAE Space Agency (UAESA), National Space Science and Technology Center (NSSTC), and UAE University. I am grateful to the UAE University that granted me a scholarship, under the Arab Space Pioneers' Programme, to pursue my MSc study.

I take this opportunity to express my sincere love and warmest gratitude to my family: my grand parents (Mr. Hussien Al Hemairy and Mrs. Adeeba Jameel) and my mother (Mrs. Raghad Hussein). I would have never accomplished this work without their continuous support and encouragements.

Dedication

To my beloved grand parents and my great mother for their unwavering support, encouragement, and love.

Table of Contents

Title	i
Declaration of Original Work	iii
Approval of the Master Thesis	iv
Abstract	vi
Title and Abstract (in Arabic)	viii
Acknowledgments	x
Dedication	xi
Table of Contents	xii
List of Abbreviations	xix
Chapter 1: Introduction	1
1.1 Related Work	4
1.2 A Review on Modular Command and Data Handling Systems	7
1.2.1 Space Plug-and-Play Avionics (SPA)	8
1.2.2 Modular Architecture for Robust Computing (MARC)	9
1.2.3 Integrated Modular Avionics (IMA)	10
1.2.4 AraMiS	11
1.2.5 PiCPoT	12
1.2.6 BRAC Onnesha	13
1.2.7 iBOSS	14
1.2.8 SNAP-1	14
1.2.9 UWE-3	15
1.3 Overall Comparison between the Surveyed Modular Architectures	16
1.4 Modularity Standards	19
1.4.1 Data Interfaces to Support Modularity	19
1.4.2 Low Speed Communication Protocols	19
1.4.3 High Speed Communication Protocols	20
1.4.4 Connectors	22
1.5 Standard Architectures	24
Chapter 2: Design	28
2.1 Design Requirements	28

2.1.1	PCIe Interface	28
2.1.2	Physical Size and Characteristics	29
2.1.3	Power Consumption	31
2.1.4	Heat Dissipation	31
2.2	Prestudy of System Alternatives	31
2.2.1	System Sketch	31
2.2.2	Potential Solutions	32
2.3	Selection of Hardware	36
2.3.1	FPGA Selection	37
2.3.2	Microcontroller Selection	38
2.4	Ethernet	40
2.5	System Architecture	43
2.6	PCB Design	46
2.7	Layout	47
Chapter 3:	Experimental Testing and Results	50
3.1	Communication Tests	50
3.1.1	UART Communication	50
3.1.2	I2C Bus Communication	52
3.1.3	SPI Communication	55
3.1.4	CAN Bus Communication	57
3.1.5	Ethernet Communication	59
3.1.6	PCIe Communication	63
3.1.7	Bridge Performance	73
3.2	Functional Tests	74
3.2.1	Functional Test of the PC/104 Computing Layer of the CDHS Architecture	74
3.2.2	Functional Test of Layers 1 and 2 Mounted on Layer 3	79
3.2.3	Functional Test of the Fully Integrated System	79
3.3	Thermal Vacuum Test of the PC/104 Computing Layer of the CDHS architecture	84
3.3.1	Test Plan	84
3.3.2	Test Setup	85

3.3.3 Test Results	87
3.3.4 Use Cases	89
Chapter 4: Conclusion	97
References	99
List of Publications	106

List of Tables

Table 1.1:	Overview of the existing surveys	7
Table 1.2:	Detailed Comparison Between SPA, MARC, and IMA .	12
Table 1.3:	Overall Comparison between the Surveyed Modular Architectures	18
Table 1.4:	Comparison Between Low-Speed Communication Interfaces	20
Table 1.5:	High Speed Communication Protocols	22
Table 1.6:	Comparison between space qualified connectors	23
Table 1.7:	Backplane Standards: CPCI Serial Space vs SpaceVPX .	27
Table 2.1:	Comparison between flight-proven FPGAs	37
Table 2.2:	Comparison between flight-proven microcontrollers . . .	39
Table 2.3:	Comparison between flight-proven STM32 microcontrollers	40
Table 2.4:	Ethernet Cables Categories	41
Table 2.5:	JLC2313 layer stackup	47

List of Figures

Figure 1.1: Command and Data Handling System Architecture of a Satellite	3
Figure 1.2: RS-422 software with additional specifications provided by SPA (Left) vs Typical RS-422 software (Right)	8
Figure 1.3: MARC SpW High Flexibility Cluster (SpW-HFC)	10
Figure 1.4: Integrated Modular Avionics Architecture	11
Figure 1.5: CPCI Serial Space slot profiles	25
Figure 1.6: SpaceVPX slot profiles	26
Figure 2.1: A cPCI-based On-board Computer Architecture	29
Figure 2.2: The physical size of the four layers	30
Figure 2.3: The system's high-level sketch	32
Figure 2.4: The solution's system sketch when using a FPGA	33
Figure 2.5: The solution's system sketch when using a CPU with native PCIe capabilities	35
Figure 2.6: The solution's system sketch when using a CPU without native PCIe capabilities	36
Figure 2.7: A block diagram that represents data transfer using Ethernet	42
Figure 2.8: Description of the system's layers	44
Figure 2.9: The system architecture of the modular CDHS	44
Figure 2.10: The system architecture of the modular CDHS (using BergStak connectors)	45
Figure 2.11: The system architecture of the modular CDHS (using BergStak connectors)	45
Figure 2.12: The system architecture of the modular CDHS (using PCIe connectors)	46
Figure 2.13: The system architecture of the modular CDHS (using PCIe connectors)	46

Figure 2.14:PCIe Layout	48
Figure 2.15:Representation of the individual modules of the system's layers	49
Figure 3.1: UART communication setup between two PC/104 computing units	51
Figure 3.2: UART communication test results	52
Figure 3.3: I2C bus communication setup between an PC/104 computing unit and an Arduino Mega board	54
Figure 3.4: I2C bus communication results	54
Figure 3.5: SPI communication setup between a flash memory and an MCU	56
Figure 3.6: SPI communication results	57
Figure 3.7: CAN bus communication setup between two PC/104 computing unit boards	59
Figure 3.8: CAN bus communication results	59
Figure 3.9: The IPV4 Settings	60
Figure 3.10:The Command Prompt (cmd) view	61
Figure 3.11:TCP server port number and IP address added to the http_test.c file	62
Figure 3.12:Running the script that writes to the PC	62
Figure 3.13:Writing data manually	63
Figure 3.14:The PCIe VHDL block design overview	64
Figure 3.15:Basic PCIe IP block on Vivado design suite	66
Figure 3.16:Customized PCIe block to supply link speed of 5 GT/s	67
Figure 3.17:Initial PCIe communication	69
Figure 3.18:Writing strings using the PCIe link	69
Figure 3.19:Simulation done after adding the led indicators and the counter - link is not ready	70
Figure 3.20:Data is being sent and received using PCIe	71
Figure 3.21:Flash programming is successful	72

Figure 3.22: The FPGA connected to the PC through the PCIe enabled M.2 slot	73
Figure 3.23: Temperature and Voltage Values of the FPGA	74
Figure 3.24: PC/104 computing unit full functional test results - part 1	75
Figure 3.25: PC/104 computing unit full functional test results - part 2	76
Figure 3.26: PC/104 computing unit full functional test results - part 3	77
Figure 3.27: Power Consumption of the PC/104 computing unit	78
Figure 3.28: Zigbee Communication Test Setup	79
Figure 3.29: Setup used for the cPCI interfacing	80
Figure 3.30: Actual setup used for the cPCI interfacing	80
Figure 3.31: The microcontroller and cPCI backplane (represented by a PC) data transfer test setup	81
Figure 3.32: The image sent during the test	82
Figure 3.33: The C header file	82
Figure 3.34: The changed IP address	83
Figure 3.35: Data received (verified using PuTTY)	83
Figure 3.36: Temperature Profile	84
Figure 3.37: Board's setup in the STVAC	86
Figure 3.38: Thermocouple on the door	86
Figure 3.39: Thermocouple on the base plate	86
Figure 3.40: Thermocouple on the cylindrical shroud	86
Figure 3.41: Test Pressure Chart	87
Figure 3.42: Test Shroud and Base Plate Temperature Chart	88
Figure 3.43: Thermocouple Temperature Chart	89
Figure 3.44: Scatter plot of the modular architectures: Performance vs Modularity	96

List of Abbreviations

API	Application Program Interface
ASIC	Application-Specific Integrated Circuit
C&DH	Command & Data Handling
CAN	Controller Area Network
CDHS	Command & Data Handling System
CMOS	Complementary Metal-Oxide-Semiconductor
COTS	Commercial Of the Shelf
CPU	Central Processing Unit
GPU	Graphical User Interface
GS	Ground Station
HK	House Keeping
HW	Hardware
I2C	Inter-Integrated Circuit
LVDS	Low Voltage Differential Signaling
MHz	Mega Hertz
OBC	On- Board Computer
RAM	Random Access Memory
RTOS	Real Time Operating system
SPI	Serial Peripheral Interface
TM&TC	Telemetry, Tracking, Command Monitoring
UART	Universal Asynchronous Receiver/Transmitter

Chapter 1: Introduction

According to industry experts, the current global space economy is capped at \$423.8 billion and is expected to reach \$1 trillion USD in annual revenue by 2040 [63]. Satellites can be categorized according to their sizes and respective mass. There are large satellites that have a mass more than 1,000 kg, medium-sized satellites having a mass of 500-1,000 kg, and small satellites. There are weights that determine the classification of small satellites, femto (less than 0.1 kg), pico (0.1-1 kg), nano (1-10 kg), micro (10-100 kg), and mini (100-500 kg) [5]. Geostationary satellites, or GEO satellites, are excellent for communication because they are always in the same place and have zero inclination above the equator. However, MEO satellites are between the LEO and GEO orbits, and they are used for navigation purposes. Furthermore, satellites placed in the elliptical orbits are used for government and military missions, while some of them can be used for commercial purposes. Under normal conditions, it takes between 5 and 15 years to determine the need for and install an average-sized or large satellite in the proper orbit. However, satellites in LEO orbit are usually small satellites, which includes nanosatellites that usually have low cost of making and are compact in size. The primary benefit of a nanosatellite, aside from its small size and low cost, is the rapidity with which each model is developed. It can take less than 8 months to identify a requirement and launch a nanosatellite into orbit. The development in small satellites is continuously emerging as there are also certain standards that are currently being developed in experimental format for picosatellites. This type of satellites can be used for earth observation, space observation, and telecommunications [38]. These reasons are continuously attracting more people to invest in this field, including not just companies, but students and researchers too. CubeSats have become popular as these cubic structured have dimensions of

10 cm³ per unit, mass of 1.33 kg per unit, relatively lower cost and power consumption than larger ones, and allowing the use of COTS (commercial off-the-shelf components) [36]. A satellite's Command and Data Handling System (CDHS) is a critical component of the satellite. The CDHS is considered as the brain of the satellite as it is responsible for controlling and coordinating the various subsystems and components on the satellite, as well as for processing and transmitting the data that the satellite collects. The CDHS consists of several elements as shown in Figure 1.1. Some of these units have their own CPUs, such as the Radhard platform controller, Instrument Control Unit (ICU), Mass Memory (provides a file system for payload data), Star Tracker, and GPS receiver. While other units without CPUs are the S-Band communication, X-Band Downlink, Antenna Control, and Remote Interface Unit that is used to connect all kind of sensors and actuators of the spacecraft. There are also different communication links that connect components with the CDHS and also that connect the CDHS with other subsystems within the satellite. These components are typically interconnected using a computer bus architecture, such as SpaceVPX or cPCI Serial, which allows for efficient communication and data transfer between the different subsystems. Additionally, software defined payload computers, which benefit from a modular architecture due to the growing demand for on-board computing and processing flexibility throughout the mission lifetime (e.g., change of protocols or algorithms), are useful to payload computers. The hardware configuration can be adjusted in accordance with the sensor and required processing power thanks to a modular architecture. In software or firmware, the exact mission requirements are implemented [22]. The CDHS has several types of communication protocols used for on-board interfacing between components and other protocols used for high speed board-to-board interfacing. The conventional on-board communication

protocols are I2C, UART, SPI, and CAN. However, the high speed board-to-board communication protocols are PCIe, Ethernet, Rapid-IO, and SpaceWire.

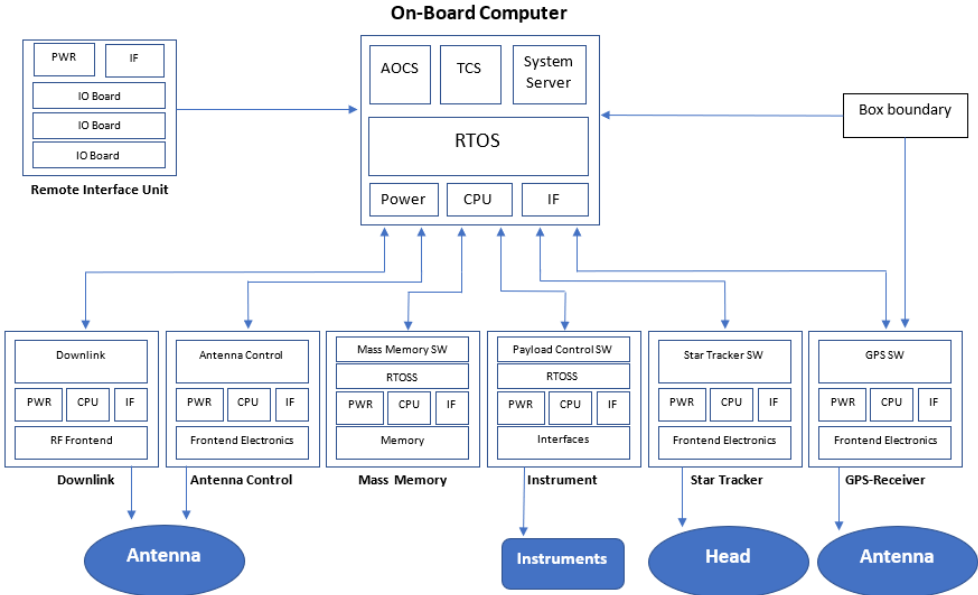


Figure 1.1: Command and Data Handling System Architecture of a Satellite

This rapid development in the space industry has led to increasing the demand for high-performance computer systems on board space missions applications, such as artificial intelligence and machine learning are drivers of the need for higher on-board command and data handling requirements. Consequently, a modular Command and Data Handling System (CDHS) is required in order to manage the complexity of future CDHS hardware. The CDHS serves as the satellite’s brain, controlling all of its operations. Among its functionalities are data management, data preparation for transmission to the earth, command maneuvers, and autonomous monitoring and response to a wide range of potential on-board issues [30]. Currently, CDHS hardware architectures have a limited ability to scale with mission needs. However, several modular standards began to introduce the concept of modularity in space applications where they aim to reduce the satellite development time and expenses. Despite the plethora of works on modular satellite systems, as

highlighted in section 1.1, to the best of our knowledge, there is no consolidated article that provides a comprehensive survey of modular CDHS. This thesis aims to provide modular CDHS hardware architectures that allow interfacing between up to four layers. This also includes interfacing between PC/104 based computing layers and cPCI based systems. We also aim to explore high data rate interfacing options. To do that, we look at systems level perspective of CDHS, particularly exploring their modularity aspects. Also, we answer some research questions that can provide insights about the need for modularity in satellites, for example, what is the importance of modularity, what are the benefits of modularity, and what modularity aspects can be implemented. Next, we discuss the different satellite modular standards and compare among them in terms of interfaces for the user to utilize, main software used in each of them, the scale of projects these standards are implemented in, and the fault tolerance schemes adopted in these standards. After, we identify several satellite missions that adopted modular architectures, going over the aspects of modularity implemented in each of them, while including brief background information about the missions, and focusing on the CDHS that is used in each of them. Furthermore, we present the data interfaces used in satellite systems, covering both, the conventional on-board communication interfaces and the high speed board-to-board communication interfaces, as well as comparing between different connectors used in the market currently to achieve such communication. Finally, we present some future improvements that can help in the development of the space field.

1.1 Related Work

The Modular Satellite Systems literature includes several review articles, as listed in Table 1.1. The application of product architectural

selection theory to small satellites, particularly the feasibility of adopting modular platform architecture for spacecraft is discussed in [70]. However, some concepts discussed in this paper might be outdated, as the study was published back in 2005, and this paper does not provide comprehensive insights on the architectures that are currently implemented. Two small satellites that were built using various methods were the subject of another study published in 2010 [51], which addressed the design challenges. The first one (PiCPoT) had a "custom" design because it was created in accordance with a precise set of criteria that identified the satellite's dimensions and functionality. The second one (ARaMiS), rather than a single satellite, represents an architecture. A fully modular approach results in the fabrication of totally reusable units (the "tiles"), allowing for the construction of a range of small satellites with varying sizes, weights, and functionalities. Within the MAESS-2006 initiative, "Regione Piemonte" is funding this research. Both projects have some issues in common, including: a focus on education, as master's and doctoral students participate in the design teams, and final projects often develop specific units or subsystems; and the use of commercial devices (COTS), with careful selection and design decisions to achieve the required reliability. Furthermore, the developers of the CubeSat Trailblazer, a 1U SPA-only spacecraft, launched in 2012 as a testbed for SPA technology, have discussed their experience. This included presenting the simplifications associated with software development of a Command and Data Handler (CDH) and the mechanisms of self-organization for independent modules as a cooperating communications system [31]. To increase the interoperability of spacecraft components, another article suggests that the space community create and adopt a global standard for spacecraft modularity. A global industry consensus standard for open and modular spaceship architecture will promote commerce, eliminate regulatory

hurdles to the market, and ultimately boost consumer value and the profitability of the space industry. This concept paper outlines the following topics: (1) the objectives of an SUMO standard and how the space community will benefit from it; (2) background information on spacecraft modularity and relevant existing standards; (3) the proposed technical scope of the current standardization effort; and (4) a methodology for developing an SUMO standard [13]. The design and construction of "Modular" and "Open" satellite buses and mission payload, as well as practical design concerns related to the usage of the Modular Open System Approach, are covered in [44], which was published in 2020. For typical commercial, civilian, and military satellite systems, existing modular Bus and mission payload architectures are reviewed. The chapter presents opinions from the space industry on "Open" versus "Closed" interface design and discusses the difficulties with Open System Architecture (OSA) approach employing MOSA principles. The system interfaces covered in this chapter include those that are internal to the satellite bus and mission payload (PL), between the bus and the payload, and external to both. This review can be really beneficial for the development of modular satellite system architecture; however, it is considered as a broad review as it focuses on the satellite buses and interfaces, unlike our review which dives into the details of the subsystem architecture of the CDHS.

Table 1.1: Overview of the existing surveys

Paper Title	Year	Area of focus
Modular Platform Architecture for Small Satellites: Evaluating Applicability and Strategic Issues [70]	2005	Investigating the feasibility of adopting modular platform architecture for spacecraft, specifically small satellites, through the application of product architectural selection theory.
Design Solutions for Modular Satellite Architectures [51]	2010	Discussing the implementation of a modular satellite design in satellite missions.
The advent of the PnP Cube satellite [31]	2012	Exploring the development process of an SPA-only single-unit small satellite. Specifically, focusing on the simplifications achieved through the software development of a Command and Data Handler (C&DH) for the satellite.
Development of a space universal modular architecture (SUMO) [13]	2013	Suggesting the development and implementation of a standard for spacecraft modularity aiming to enhance the interoperability of different spacecraft components.
Future Satellite System Architectures and Practical Design Issues: An Overview [44]	2020	Examining current and forthcoming patterns in the development and construction of satellite buses and mission payloads with a focus on "Modular" and "Open" approaches, as well as exploring practical design challenges.

1.2 A Review on Modular Command and Data Handling Systems

Many CDHS hardware architectures lack the appropriate level of modularity to enable scalability based on mission requirements. However, several modular standards began to introduce the concept of modularity in space applications where they aim to reduce the satellite development time and expenses, as well as to add further functionality to the satellite system with minimal or no adjustments done on the main satellite bus.

The single biggest determinant in deciding a product's flexibility and which attributes can be maximized is its architecture, which describes how its functions, interfaces, and components are specified [70]. Modularity is the ability to add additional features with the minimal need for customization. Applying modular architectures in satellites would have lots of advantages on subsystem level and the satellite as a whole. This is especially done in satellite constellations, where several satellites are launched to be operated collaboratively, each containing minimal requirements. Therefore, following modularity can lead to a significant reduction in the associated costs [61]. The following are some of the modular CDHS that were developed along with some of the missions associated with them for technical demonstration.

1.2.1 Space Plug-and-Play Avionics (SPA)

The Space Plug-and-Play Avionics (SPA) is a modular approach to developing satellite hardware and software architectures. It offers a way for developers to customize their architecture or use pre-developed solutions according to their needs. This approach allows for self-discovery and self-configuration of heterogeneous Plug-and-Play (PnP) networks where components can communicate without prior knowledge of the corresponding component's location or the type of interconnection network it uses [35]. The SPA technology supports five different subnets with their unique capabilities, including I2C, USB, SpaceWire, Optical, and local UDP sockets. However, no single subnet fulfills all the requirements for a SPA subnet, and each must be supplemented in its way [39]. The SPA network has standardized interfaces in usual software communication and network layers, such as RS-422, shown on the left side of Figure 1.2, which provide several benefits over typical RS-422 software, shown on the right side [12]. The modular, reconfigurable SPA interface can result in substantial cost and schedule savings during development and testing [40].

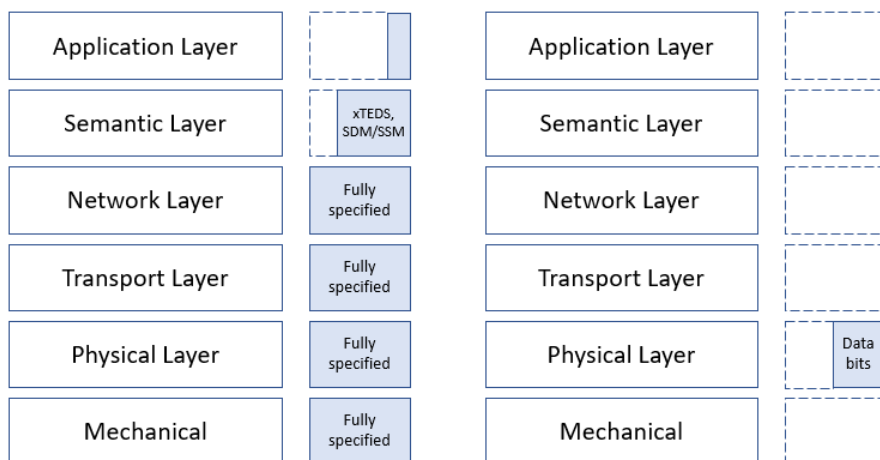


Figure 1.2: RS-422 software with additional specifications provided by SPA (Left) vs Typical RS-422 software (Right)

During the AFRL TacSat-3 spacecraft mission, the Spacecraft Avionics Experiment (SAE) was tested as a secondary experiment to showcase the SPA technology. The SPA interface was modular and reconfigurable, providing significant cost and time savings during SAE development and testing. The SAE flight experiment consisted of two distinct interfaces, the Smart Deck for SPA-U devices and the Intelligent Power and Data Ring (IPDR) PnP interface for all other sensors [12]. The PnPSat-1 mission was also planned to demonstrate the SPA technology, providing a standardized small spacecraft bus that can be quickly fitted with prefabricated modular sensors and avionic modules. The PnPSat-1 mission was expected to launch in 2008 on a Falcon-1 launch vehicle, but it was not successfully developed on time [65], serving only as an engineering model for the development of the Modular Space Vehicle (MSV) satellite [17].

1.2.2 Modular Architecture for Robust Computing (MARC)

The main objective of the Modular Architecture for Robust Computing (MARC) project is to demonstrate the basic characteristics of a reliable and distributed avionics system that uses a SpaceWire network [23]. The SpaceWire network used in MARC is centered around a High Flexibility Cluster (HFC), consisting of two 8-port routers that provide redundancy. Each module of the HFC is connected to both routers to support the failure of one router in the cluster. The HFC includes four modules that can perform any task that requires a network interface, including processing, memory, and I/O [57]. Additionally, it has four spare ports, each with a redundant SpaceWire link that can connect to clusters or other system components, such as the TMTC system or EGSE. The HFCs can be connected in different topologies, and the number of 8-port routers can be increased to increase bus bandwidth. MARC uses five types of SpW-modules to meet the basic

platform and payload computing requirements: I/Os Module (IOM), Telemetry/Telecommand Module (TTM), Core Computing Module (CCM), General Computing Module (CGM), and Memory Module (MM), as illustrated in Figure 1.3. The project applies a specific fault-tolerance method, implementing an n+m redundancy approach to processing modules, rather than the traditional 2n approach. This method takes advantage of the other features of GenFAS, a decentralized PUS-based data handling onboard software architecture, based on the SOIS and SpaceWire communication specifications [22].

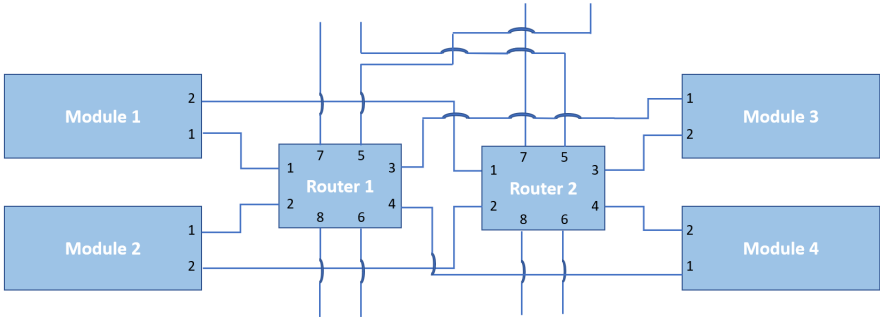


Figure 1.3: MARC SpW High Flexibility Cluster (SpW-HFC)

1.2.3 Integrated Modular Avionics (IMA)

Integrated Modular Avionics (IMA) is a system architecture commonly used in aircrafts that allows multiple independent functional chains to share a common computing resource while protecting each application from interference through memory protection strategies and controlled communication channels [49]. This architecture has caught the attention of the European Space Agency (ESA), which has proposed the use of Integrated Modular Avionics for Space (IMA-SP) in spacecraft flight software architecture. This would increase the reliability and security of space systems while improving the efficiency of software development and validation processes by integrating software partitioning technology [60]. The IMA-SP

could also benefit from using existing ARINC standards such as ARINC 429 Networking, ARINC 650, ARINC 651, and ARINC 653. While IMA has not yet been fully integrated into space applications, a similar standard, cPCI Serial Space, has been successfully implemented in large scale satellites, showing the potential for IMA-SP integration in the space industry [50].

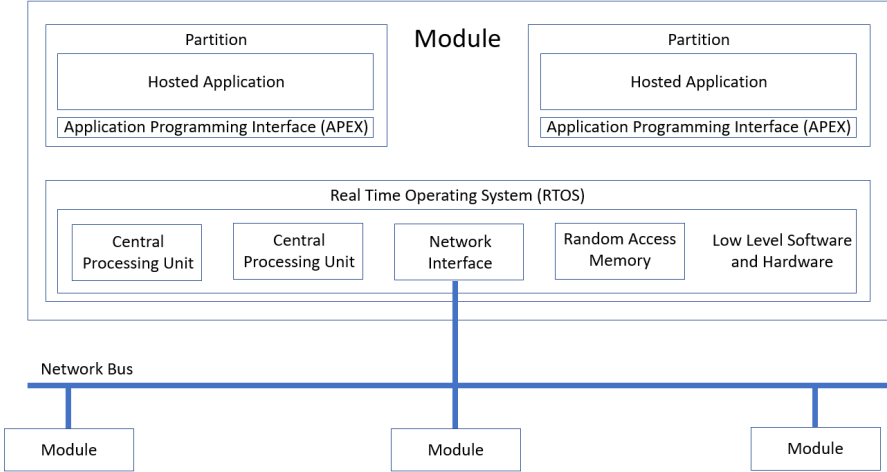


Figure 1.4: Integrated Modular Avionics Architecture

Table 1.2 shows a comparison between the three modular architectures mentioned above.

1.2.4 AraMiS

The AraMiS project was developed by a team of researchers at Turin Polytechnic, with the goal of creating a fault-tolerant system that uses commercial-off-the-shelf (COTS) components to keep costs reasonable. To achieve this, the project identified the most critical subsystems, including the mechanical subsystem, on-board processing subsystem (CDHS), power management subsystem (EPS), telecommunication subsystem, payload support, and ground segment [10]. The system uses an intelligent tile-based modular architecture, with inner tiles containing the on-board processor and payload support, and outer tiles including the power management and

Table 1.2: Detailed Comparison Between SPA, MARC, and IMA

	SPA [35]	MARC [23]	IMA [49, 60]
Voltage Interfaces	28v and 5v	3.3v and 5v (Additional 1.8v can also be supplied)	5V is required (An extra supply of +/-12V is needed only if required by mounted PMC module)
Data Interfaces	– SPA-O – SPA-S(LV) – SPA-U – SPA-1	– UART – CAN – I2C – SLINK – SpaceWire	–Ethernet – AFDX
Software	xTEDS (extended Transducer Electronic Datasheets)	GenFas Software (based on CCSDS-SOIS Architecture) including FDIR Manager	ARINC 653 (Avionics Application Software Standard Interface) operating system specification
Scale of Projects	Nano and Small Scale Satellites	Small and Medium Scale Satellites	Aircrafts

telecommunication subsystems. The CDHS is designed to perform regular tasks such as storing data, executing commands, and controlling payload boards, while also utilizing a RAID-1-like storage system for radiation hardness. The On-Board Computer (OBC) is triple modular redundant and can correct Single Event Upset errors. The project’s design enables graceful performance degradation while still keeping costs reasonable.

1.2.5 PiCPoT

After the successful AraMiS project, Turin Polytechnic started another student nanosatellite project called PiCPoT, which has a modular architecture and uses Commercial Off-The-Shelf (COTS) components for every system. The goal of this satellite is to transmit on-board telemetry measures, such as temperature, voltage, and current of solar panels and batteries, and photos taken with commercial cameras. Due to budget restrictions, the design had to be radiation-hardened by using C-MOS technology, but an anti-latch-up system was added to monitor the supply current of all ICs [16]. PiCPoT has a

special CDHS that consists of two independent subsystems, each with its own On-Board Computer, power supply (Li-Po and Ni-Cd batteries charged by 5 GaAs solar panels with 5 MPPT), time scheduler, and RF module (437 MHz and 2.44 GHz). One of the two board processors is called ProcA, which is responsible for acquiring analog signals from the power supply board and the number of latch-up events from the power switch board. The acquired signals are selected using an analog multiplexer controlled by a processor through an address decoder implemented with a Xilinx CPLD. The collected data is stored in a serial FERAM, and then telemetry packets are created using this data [15]. The battery status is checked, and the selected battery is charged using the solar panels. After completing all the housekeeping operations, ProcA waits for incoming commands from the ground station. If a correct command is received, the corresponding operation is performed; else, the housekeeping telemetry packet is sent as a beacon, and the "Switch-Off" command is sent to the Power Switch board to turn off the OBC. The second board processor is called ProcB, which has a different design from ProcA to ensure fault-free operation of at least one of them. ProcB uses the integrated FLASH memory of its MSP430 microcontroller to save space in the board, an external transceiver housed in the Tx-Rx board, and is connected to an electric motor with a reaction wheel to add active spin-axis stabilization [16]. Both ProcA and ProcB have the same commands, with only the addition of the reaction wheel command on ProcB.

1.2.6 BRAC Onnesha

BRAC Onnesha is a nanosatellite project that was built in Bangladesh in collaboration with the Kyushu Institute of Technology Birds-1 program. Originally, the satellite was planned to have a unique design consisting of six individual pyramid-shaped units that could be assembled to form a central

space for the battery. This modular design would allow for equipment to be set up on all six sides, making it easier to test each mission as a unit and shorten the development period [28]. The satellite also uses reaction wheels to achieve constant repositioning and maximize its exposure to the sun. However, the final design was a 1U CubeSat that was part of the Birds-1 program constellation, which provided valuable experience for students and allowed for communication with seven ground stations across several countries [24].

1.2.7 iBOSS

The iBOSS project, supported by the German Aerospace Center DLR, has two main components: modules containing both structural and functional elements and a standardized 4-in-1 interface for docking, power, data, and thermal interconnection. The project aims to enable in-orbit spacecraft servicing, as well as the replacement and improvement of standard in-orbit infrastructure elements. This will allow the fast development of iBOSS-based flexible space systems using prequalified modules and interfaces [32]. However, a major challenge is integrating a highly functional interface mechanism within limited building space while keeping the overall module dimensions reasonable. To ensure robotic servicers can manipulate these interfaces, mechanical interface designs are evaluated experimentally. Additionally, computer-aided satellite design using a module catalog makes the satellite design process more manageable by applying modularity [54].

1.2.8 SNAP-1

The Surrey Space Centre (SSC) and Surrey Satellite Technology Ltd (SSTL) designed and built the UK's first nano-satellite, SNAP-1, to demonstrate that a fully agile and sophisticated satellite can be constructed

rapidly and inexpensively using a modular, COTS-based design philosophy. The design philosophy relied on standardizing the electrical and mechanical interfaces of each module, which included regulated 5V and raw battery power connections, a single bidirectional Controller-Area-Network (CAN) bus for data transfer, and a standard 9-way D-type connector in all modules. In addition, each module included a standard 8-bit CAN-micro-controller, except for the on-board computer (OBC) and machine vision system (MVS) modules which were equipped with 32-bit StrongARM SA1100 RISC processors [66]. The SNAP-1 program defined a standard mechanical format for hosting standard Eurocard printed circuit boards, which allowed mechanics, avionics, and payload design to occur in parallel and enabled procurement to begin early in the program. The mechanical structure of SNAP-1 consists of three sets of three electronic module boxes connected together to form a triangular structure. The success of the SNAP-1 mission demonstrated that sophisticated mission objectives could be achieved using low-cost, modular, COTS-based nano-satellites constructed rapidly (in less than 9 months) [67].

1.2.9 UWE-3

The UWE-3 CubeSat, built by University of Würzburg students and funded by the German Federal Ministry of Economics, was launched in November 2013. The satellite's goal was to create a simple and robust infrastructure for developing, integrating, testing, maintaining, and replacing subsystems during flat-sat development or flight model integration. This was achieved by having a generalized backplane that connects boards with standardized connectors [7]. The UWE-3 bus includes redundancy features such as a dual-redundant low power onboard computer, a redundant and distributed electrical power system, a redundant UHF communication system

with separate monopole antennas, and an attitude determination and control system. The OBC enhances reliability by implementing two redundant processing units on a single subsystem board, using ultra-low power microcontrollers optimized for housekeeping and basic operations [8]. The design also reduces the target for radiation-induced errors and improves robustness through code execution from flash memory, warm-backup schemes, and independent toggle watchdog units. Additionally, the redundant devices can cross-connect and provide mutual aid and re-configuration if required [9].

1.3 Overall Comparison between the Surveyed Modular Architectures

Significant advancements have been made in the field of modular architectures for a variety of applications, including avionics, robust computing, and space systems. To give a thorough overview of the modularity features of various modular architectures, such as Space Plug-and-Play Avionics, Modular Architecture for Robust Computing, Integrated Modular Avionics, AraMIS, PiCPoT, BRAC Onnesha, iBOSS, SNAP-1, and UWE-3, Table 1.3 has been created. Standardization stands up as a key component among the examined factors, with Integrated Modular Avionics and Space Plug-and-Play Avionics both providing standardized interfaces. Another important factor is modularity, which is found in abundance in the Space Plug-and-Play Avionics, Modular Architecture for Robust Computing, BRAC Onnesha, and UWE-3 systems. The sizes of the architectures vary, with PiCPoT and UWE-3 emphasizing smaller form factors and Space Plug-and-Play Avionics and Modular Architecture for Robust Computing focused on larger systems. Each architecture has a different level of interconnection, scalability, and power consumption. Every system demonstrates distinctive qualities, such as diverse power consumption

rates, scalability possibilities, and various connectivity types, such as serial, Ethernet, PCIe, SPI, I2C, and UART. The systems' different data rate capabilities offer flexibility to meet the needs of various applications. The ecosystem that surrounds each architecture is also in a state of development and availability that varies. While AraMIS, PiCPoT, BRAC Onnesha, and iBOSS are in the development stage, Integrated Modular Avionics benefits from an established ecosystem. All of the aforementioned architectures have a limited commercial availability, which highlights the need for more development and market penetration.

It is crucial to remember that the data in this table and the discussion in this section is only a broad overview, and that the details of how each architecture has been implemented and advanced may have an impact on its qualities and applicability for certain applications, as mentioned in the sections that describe each of the modular architectures from satellite development perspective, focusing on Nano and Micro Satellites.

Table 1.3: Overall Comparison between the Surveyed Modular Architectures

Architecture	Standard-ization	Modularity	Size	Scal-ability	Inter-connect	Ecosystem	Application Suitability	Comme-rcial Avail-ability	Fault Tol-erance	Plug-and-Play Capa-bilities
Space Plug-and-Play Avionics	Yes	High	Small to Large	High	Serial, Ethernet, etc.	Expanding	Satellites, Spacecraft	Limited	Yes	Yes
Modular Architecture for Robust Computing	No	High	Large	High	Ethernet, PCIe, etc.	Developing	Robust Computing Systems	Limited	Yes	No
Integrated Modular Avionics	Yes	High	Large	High	Serial, Ethernet, etc.	Established	Avionics Systems	Limited	No	Yes
AraMIS	No	Medium	Medium	Medium	Ethernet	Developing	Military, Aerospace	Limited	Yes	No
PiCPoT	No	Medium	Small	Low	SPI	Developing	IoT Devices	Limited	No	No
BRAC Onnesha	Yes	High	Small	High	I2C, SPI	Developing	Small Satellites	Limited	No	Yes
iBOSS	Yes	Medium	Small	High	UART	Developing	Small Satellites	Limited	No	No
SNAP-1	Yes	High	Small	Medium	I2C	Established	Nano Satellites	Limited	No	No
UWE-3	Yes	High	Small	Medium	I2C	Established	Nano Satellites	Limited	No	No

1.4 Modularity Standards

1.4.1 Data Interfaces to Support Modularity

There is a set of communication protocols that allow the command and data interfacing within subsystems and between systems. There communication protocols are categorized into typical command and data interfaces and high speed communication protocols.

1.4.2 Low Speed Communication Protocols

The Inter-Integrated Circuit (I2C) is a type of serial communication bus that is often used for connecting slower peripheral integrated circuits (ICs) to processors or microcontrollers over short distances [58]. It is a synchronous protocol, meaning that data is sent using a shared clock signal between the sender and receiver, and can support multiple devices connected to a single or multiple master devices. The I2C is similar to other buses like UART and SPI, but combines their best features by using only two wires to transmit data like UART and synchronizing output and input using a clock signal like SPI. The UART protocol is also a type of asynchronous serial communication that uses two signals, Tx and Rx, to communicate, and can support maximum data rates of up to 460 kbps [71]. Another protocol, the Controller Area Network (CAN), was initially developed for the automotive industry and is used for time-critical functions, supporting data rates of up to 5 Mb/s, if it is CAN FD [52]. The Serial Peripheral Interface (SPI) is another type of data bus that is similar to I2C, but it uses a dedicated slave select wire per device and is a full duplex bus, meaning that data can be transmitted in both directions simultaneously [58]. The data rate of SPI is limited by the clock speeds of the master or slave device and can be up to several hundred megabits per second, but it is best to stay at least one order of magnitude below the slowest controller clock speed for robustness.

These interfaces are compared in terms of speed, type of duplexity, and whether being synchronous or asynchronous. As illustrated in Table 1.4, we can identify that in terms of speed, SPI has the highest, CAN bus comes next, then UART and I2C. Furthermore, the I2C is half-duplex while the rest are full-duplex. The rest of the details are included in the table.

Table 1.4: Comparison Between Low-Speed Communication Interfaces

	I2C [71]	UART [58]	SPI [71]	CAN [52]
Speed	3.4 Mbps	5 Mbps	60 Mbps	5 Mbps
Protocol (Sync / Async)	Synchronous	Asynchronous (Serial protocol)	Synchronous	Synchronous (Multimaster protocol)
Duplex	Half duplex	Full duplex	Full duplex	Full duplex
Types of Lines / Ports	Two Lines: SCL (serial clock line) SDA (serial data line acceptance port)	Two Lines: TX RX	Four ports: MOSI, MISO, SCLK, and NSS	Two Lines: CAN High CAN low

1.4.3 High Speed Communication Protocols

Also, in terms of high speed communication protocols, Spacewire, Ethernet, RapidIO, and PCIe are used. SpaceWire is a point-to-point data bus (directly connects two devices) that is designed by the European Space Agency (ESA) specifically for space applications. One of the devices connected by SpaceWire can be a router which connects several other devices via SpaceWire or other data busses [46]. SpaceWire is also a full duplex bus that uses differential signaling and has separate data and strobe signal, supporting data rates of up to 400 Mb/s [47]. However, due to its higher power consumption when compared to I2C and other low power busses, SpaceWire is less suitable for small factor satellites, such as CubeSats [14]. Ethernet is another protocol used for high speed interfacing, as it is the traditional technology for connecting devices in a wired local area network (LAN) or wide area network (WAN) allowing them to transmit data so other

devices on the same LAN or campus network can recognize, receive and process the information. An Ethernet cable is the physical, encased wiring over which the data travels, where it is used by connected devices that use cables to access a geographically localized network, instead of a wireless connection. When compared to other technologies, Ethernet is less vulnerable to disruptions making it highly reliable and secure. It can also offer a greater degree of network security and control than wireless technology because devices must connect using physical cabling [68]. A third protocol used for high speed interfacing is Rapid-I-O, which is a data bus for time critical computer systems with extreme performance requirements. The throughput of this bus is up to 10 Gb/s for a single lane and can be multiplied by increasing the number of lanes [21]. Rapid-I-O is most commonly used in the terrestrial telecommunications industry, such as in equipment at cellular towers. The performance and its robustness make this point-to-point data bus interesting for some dedicated space instrumentation with very high performance requirements [6]. The fourth protocol is Peripheral Component Interconnect Express (PCIe), which is a high-speed serial bus standard that is commonly used in computers and other electronic devices to connect peripheral devices to the motherboard. PCIe provides a high-speed, scalable, and flexible interface that allows for efficient communication and data transfer between different components in a system. In satellite applications, PCIe is often used in conjunction with a bus architecture such as SpaceVPX or cPCI Serial Space. This provides a standardized and scalable interface that allows for easy integration and interoperability between different subsystems on the satellite. PCIe is particularly well-suited for use in satellites due to its high data rates and being configurable, supporting a wide range of data rates, from 4 Gb/s for a single lane and can be multiplied by increasing the number of lanes, reaching up to 64 Gb/s [43]. This allows satellite designers to use

PCIe to support a wide range of applications and subsystems, from low-speed peripherals such as sensors and cameras, to high-speed data transfer and processing systems.

Table 1.5: High Speed Communication Protocols

	Ethernet [68]	SpaceWire [47]	Rapid-IO [21]	PCIe [43]
Bandwidth	10 Mbps to 400 Gbps	2 to 400 Mbps	25 Gbps/lane, 100 Gbps/port	4 Gbps/lane, 64 Gbps/port
Duplex	Full-duplex	Full-duplex	Full-duplex	Full-duplex
Links	Point-to-point	Point-to-point	Point-to-point	Point-to-point
Packet Size	1500 bytes	4 to 64 bytes	4 to 64 bytes	32 to 512 bytes

Ethernet, SpaceWire, RapidIO, and PCIe are all high-speed communication protocols that are commonly used in computer networking and data transfer. Each of these protocols has its own characteristics in terms of packet size, bandwidth, and robustness, and they are designed for different applications and environments. As illustrated in Table 1.5, we can identify that each of these protocols has its own characteristics in terms of packet size, bandwidth, and robustness, and they are designed for different applications and environments. Ethernet is a widely used networking protocol with a relatively large packet size and robust error checking and recovery mechanisms, while SpaceWire is a highly reliable protocol designed for use in space applications with a small fixed packet size. RapidIO and PCIe are both high-speed interconnect protocols with a wide range of packet sizes and built-in error checking and recovery mechanisms.

1.4.4 Connectors

In satellite systems nowadays, there is an increasing demand for high speed connectors and cables for board-to-board interfacing, as they carry data and power between different subsystems of a spacecraft or satellite. As a result, space-qualified connectors must meet demanding standards such as

radiation resistance, high temperature and vacuum conditions, and long-term reliability. The connections in Table 1.6 are some of the most widely utilized in space applications. Because of its excellent reliability and robustness, the D-Subminiature connector is commonly utilized in space applications. Another typical connector that can withstand high vibration and shock loads is MIL-C-55302. MIL-C-83513 Space Grade is a small connector that is extremely dependable and can be utilized in high-density applications. MIL-C-38999 is a tough connector that can withstand extreme temperatures and has a good radiation resistance. Newer connector types, such as the PCIe connector, SODIMM, and Bergstak Mezzanine connector, are also included in the chart, and can provide high-speed data transmission rates for space applications [20]. These connectors' transmission speeds can vary depending on their model and setup. As a result, choosing the right connector necessitates careful consideration of various criteria, including data transfer rate, contact arrangement, mating cycles, size, and environmental requirements.

Table 1.6: Comparison between space qualified connectors

Connector Name	Pitch	Number of Contacts	Voltage Rating	Current Rating	Temp. Range	Transfer Speed Limit (Gbps)
Micro-D [27]	1.27mm	9-51	200V	3A	-55°C to +125°C	1.5
Nano-D	0.635mm	9-65	200V	1A	-55°C to +125°C	6
D-Subminiature	2.54mm	9-104	1000V	5A	-55°C to +125°C	6
Space Grade MIL-C-55302	2.54mm	6-96	200V	5A	-65°C to +125°C	6
Space Grade MIL-C-83513	1.27mm	9-100	200V	3A	-65°C to +125°C	10
Space Grade MIL-C-38999	10-22mm	3-128	200V	23A	-65°C to +175°C	3
PCIe [20]	1.00mm	36-164	30V	1A	-55°C to +125°C	32
SODIMM	2.54mm	144-260	200V	1.5A	-55°C to +85°C	21.3
BergStak	0.50mm	20-240	50V	0.5A	-55°C to +85°C	6

1.5 Standard Architectures

CPCI Serial Space and SpaceVPX are two different technologies that are used in high-performance computing applications, such as in aerospace and defense. cPCI serial space is a type of computer bus used in some types of industrial and commercial systems. It is an extension of the CompactPCI (cPCI) bus, which is a standardized format for computer boards and systems that is widely used in a variety of applications. CPCI Serial Space is based on the PCI Express (PCIe) standard and is designed to provide a high-speed, serial communication link, allowing devices to communicate with each other using serial protocols such as RS-232 and RS-485 [25]. This enables the bus to support a wider range of devices and communication methods, making it more versatile and flexible than the original cPCI bus. CPCI Serial Space differentiates between three slot profiles: Power, System, and Peripheral, along with an additional Shelf Controller, which is not part of the standard, as illustrated in Figure 1.5. The Controller Switch Module and Switch Module are typically used in system slots, while the Data In Module, Storage Module, Processing Module, and Controller Module are typically used in peripheral slots. Each peripheral slot is connected to the C&C bus of the corresponding system slot. So, a dual star architecture is put into practice. If one component fails, the backup component takes over. A full mesh design, in which each slot is connected to every other slot (up to 8) using four differential pairs, is used by cPCI for high-speed interconnects. It is feasible to send data at a 10 Gbps rate over the backplane. Also, a dual star architecture is also feasible when used with a switch. Any of the peripheral slots may get the switch. Each of the slots has six connectors. The connector P6 is given access to the entire mesh network. The pins for the SpaceWire star architecture are provided by the connectors P2, P4, and P5 in a system slot, therefore each peripheral slot is connected to the system slot by connector P2 [34]. The

redundant CAN bus is also connected via the connector P2. P1 has access to all power and management signals. Up to 180 user-definable pins are provided by each peripheral slot. These could be utilized for peripheral slot connectivity or back connectors.

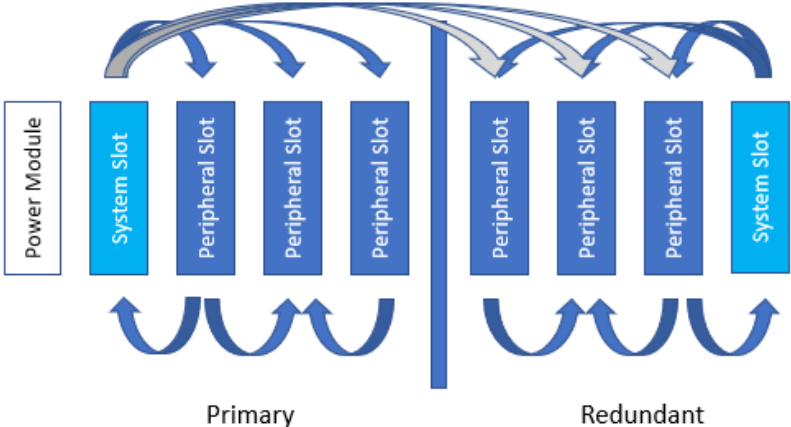


Figure 1.5: CPCI Serial Space slot profiles

However, SpaceVPX is a system architecture for space-based computing platforms that is based on the VPX (VITA 46) open standard [26]. VPX is a standard for modular embedded computing systems that is widely used in military and aerospace applications. SpaceVPX builds on this standard to provide a scalable and flexible architecture for space-based computing platforms, such as satellites and spacecraft. The goal of SpaceVPX is to enable the development of more advanced space systems that can be easily integrated, updated, and expanded over time. The unit is known as Space Utility Module (SpaceUM) in Space VPX [48]. The goal is comparable to that of the cPCI shelf controller. System management, reference clocks, reset, and power signals are all routed over the Switched Utility Plane, and the Space Utility Module switches between redundant units. Space VPX differentiates between 9 slot profiles: Power, DataIn Module, Processing Module, Storage Module, DataOut Module, Controller Module, Controller Switch Module, DataSwitch Module, and Space Utility

Module, as illustrated in Figure 1.6.

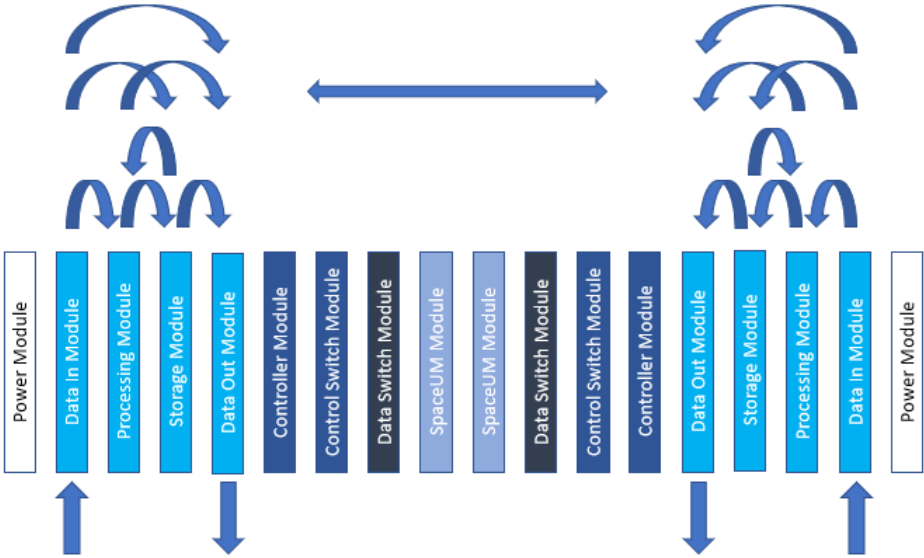


Figure 1.6: SpaceVPX slot profiles

In summary, cPCI Serial Space and SpaceVPX are two technologies that are used in high-performance computing applications, but they have different architectures, interfaces, form factors, and levels of scalability. Here is a detailed technical comparison between the two:

- Architecture: CPCI Serial Space is a bus architecture that adds a high-speed serial bus to the traditional parallel bus architecture of cPCI. SpaceVPX, on the other hand, is a modular architecture based on the VITA 46 (VPX) standard.
- Interface: CPCI Serial Space uses the PCIe interface for the serial bus, which provides high-speed communication between different components of a system. SpaceVPX, on the other hand, uses the Serial RapidIO (SRIO) interface for high-speed communication between modules.
- Form factor: Both CPCI Serial Space and Space VPX use the 3U form factor, which is a standard size for electronic modules that are used in

computer systems. Also, both of them can also be of the 6U form factor, which is 2.7 times larger than the 3U form factor and allows for more complex and powerful electronic systems.

- **Scalability:** Both CPCI Serial Space and SpaceVPX are scalable technologies, meaning that they can be easily expanded to support additional components or modules. However, the modular architecture of SpaceVPX allows for more flexibility and scalability compared to the bus architecture of CPCI Serial.

Table 1.7: Backplane Standards: CPCI Serial Space vs SpaceVPX

Criteria	cPCI Serial Space [25]	SpaceVPX [26]
Backplane Layout (Number of slots)	Number of peripheral slots varies from 0 to 7 [34]	Number of slot types per backplane may vary for different configurations [34]
Form factor	3U / 6U	3U / 6U
Connector	Airmax VS (Proven in harsh environment, lower complexity, and lower cost)	MultiGig RT (Proven in space)
Total number of pins per board	Up to 184 pin pairs (3U/6U)	Up to 192 pin pairs and 48 single-ended (6U)
Command & Control link	SpW dual star; switch fabric in system slot	SpW dual star; switch fabric in dedicated slot
C&C or Management bus	CAN, I2C	I2C
Secondary Power Voltage	12V and 5V auxiliary voltage, POL	3.3V, 5V, 12V, 3.3V AUX, and 48V, POL for big racks
Slot types	Power, System, and Peripheral slots	Power, System, Control, Payload, Storage, and Switching slots
Power dissipation	Up to 80 W per slot	Capable of dissipating more than 80W per slot
Standardization	Fully standardized	Partly standardized (provides a lot of flexibility)

Chapter 2: Design

The aim of this thesis is to design and implement a modular Command and Data Handling System that includes four layers and is capable to interface with a CompactPCI (cPCI) serial backplane, which includes both PCIe and Ethernet interfaces. The system also has the ability to also be modified to interface with SpaceVPX. The cPCI Serial Space standard defines a serial interconnect system for embedded systems, which supports a range of interfaces including PCIe, Ethernet, USB, and others. The cPCI serial backplane provides a high-speed interconnect between cPCI serial boards, with the PCIe and Ethernet interfaces being the most commonly used. Figure 2.1 presents an example of cPCI-based On-board Computer architecture that contains an ASIC, Memories, and Power converters [19]. Therefore, to apply the same thing to our architecture, we are aiming to use the PC104-based computing layer (layer 3) as our main computing unit equipped with the Ethernet transceiver, a daughterboard that contains the Ethernet-to-Pcie converter (layers 1 and 2), and a 3U form factor board as the host board (layer 4) for the other layers and contains the cPCI connectors that are connected to the cPCI-based backplane. This 3U board would mount an FPGA that is used for extra functionality.

2.1 Design Requirements

This section lists the design requirements and explains them. The general architecture and the components used shall fulfill these requirements.

2.1.1 PCIe Interface

The design shall be compatible with a PCIe Gen2 x1 interface, which has at least one data lane operating at 5 GT/s. Future generations are preferred, but not required. To accommodate increasing data rate

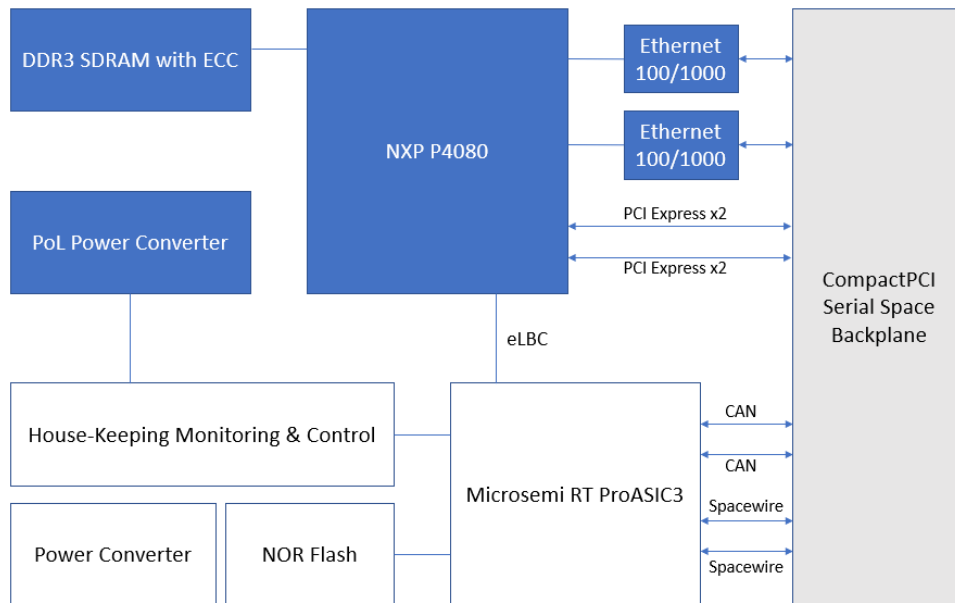


Figure 2.1: A cPCI-based On-board Computer Architecture

requirements, the lane arrangement can be expanded based on future specific system requirements. The architecture ought to support PCIe data transmission and reception.

2.1.2 Physical Size and Characteristics

The modular system should consist of four layers, each containing boards of different sizes, as presented in Figure 2.2. These are the size requirements of the four layers:

- Layer 1: This layer shall contain components of different sizes. The total size of these components shall be smaller than the size of the layer 2 board.
- Layer 2: This layer shall contain a board of one of the M.2 expansion cards standard sizes (ranging from 22x30x0.8 mm to 22x80x0.8 mm).
- Layer 3: This layer shall contain a board of the PC/104 standard size (90x90x1.6 mm).
- Layer 4: This layer shall contain a board of the cPCI standard size, following either the 3U or 6U form factor (160x100x1.6 mm).

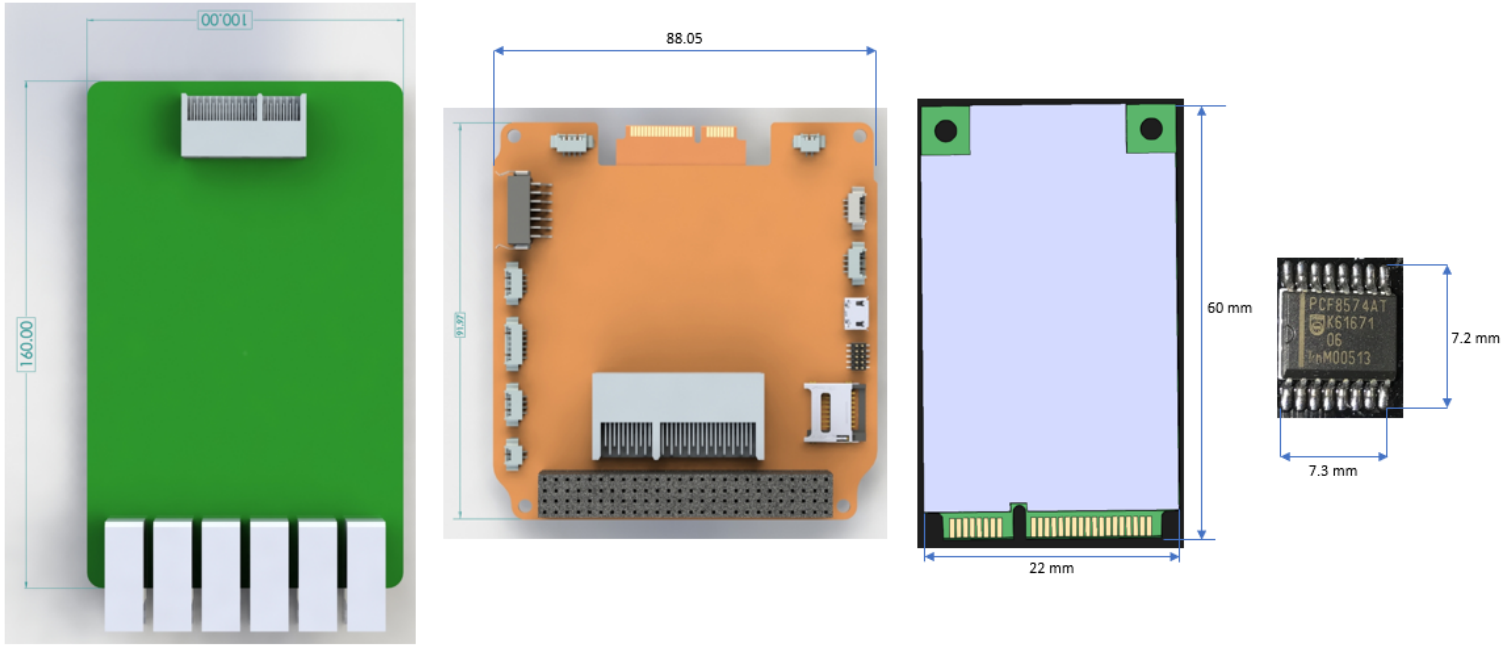


Figure 2.2: The physical size of the four layers

2.1.3 Power Consumption

The design shall be operational with minimal power requirements.

2.1.4 Heat Dissipation

The design shall fit into a compact space with few cooling and heating options, which means that the solutions shall not require active cooling or heating.

2.2 Prestudy of System Alternatives

An examination of the possibilities that can provide the functionality that was previously defined makes up the prestudy. The study comprises weighing the advantages and disadvantages of the various alternatives and choosing which one will be put into practice. The PCIe standard and data flow between devices were studied to learn about potential solutions. When sufficient understanding of the standard had been attained, other options for accomplishing the needed functionality were looked into. The choices are listed below, and the best choice is determined by comparison. The decision is influenced by a number of factors, including: Power requirements, solution's complexity, components' price, components' physical size, and key components' life cycle.

2.2.1 System Sketch

To achieve this study's objectives, a method to convert between PCIe and CMOS/LVDS should be chosen. There are different methods that can be used to accomplish this, while taking into account factors like cost, complexity, physical dimensions, and longevity. The data will be supplied to the bridge where the following actions are performed through a rapid PCIe link: (1) The overhead is eliminated from data transmitted via PCIe before it

is transmitted as LVDS/CMOS over a user-defined interface. (2) After acknowledging receipt, the recipient transmits the data back to the root complex after reinserting the overhead.

The bridge should be able to communicate quickly with many different devices. The system sketch, with the bridge block representing the system that manages PCIe to LVDS/CMOS conversion is shown in Figure 2.3.

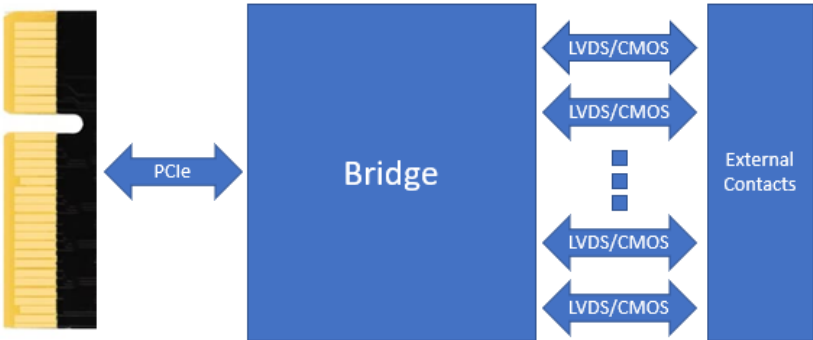


Figure 2.3: The system’s high-level sketch

The possibilities are severely constrained by the requirement for a PCIe endpoint because a sizable number of the devices on the market are only built to support root complicated mode. Additionally, the need for data conversion between interfaces further reduces the available options. The use of a PCIe switch with multiple hardware bridges, which would allow each interface to appear as a separate endpoint, was one suggested option. Due of the limited number of hardware bridges that can handle different interfaces and the complexity of modifying such a solution after it has been built, this idea was finally abandoned. Therefore, only two implementation choices remained: a Central Processing Unit (CPU) or a Field Programmable Gate Array (FPGA).

2.2.2 Potential Solutions

To achieve this thesis’ objectives, which are mainly establishing communication between the cPCI backplane and the four layers of the

modular CDHS architecture, there are several alternative solutions that have to be evaluated based on their advantages/disadvantages and one of them has to be chosen.

1. Programmable logic: FPGAs and other programmable logic are frequently utilized for accelerating computing in high-speed applications. Many vendors provide value-line and low-power FPGAs appropriate for digital processing. Figure 2.4 depicts the system sketch when an FPGA is used.

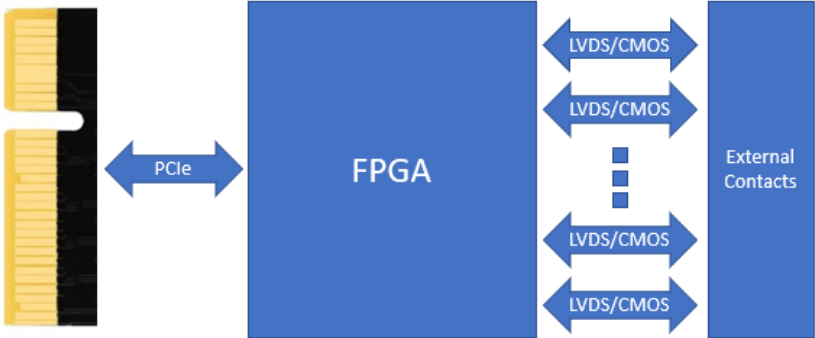


Figure 2.4: The solution’s system sketch when using a FPGA

There are several advantages of using FPGAs, such as they frequently have a lot (100+) of GPIOs, and the majority of them allow both single ended and differential termination choices. Some FPGAs can function as endpoints and offer complete hardware support for all PCIe levels. It is also possible to configure FPGAs that lack hardware PCIe capability to support it due to the way they function [69]. To meet this requirement, such FPGAs, with the addition of transceivers, must be able to communicate at PCIe speeds. The higher levels are implemented using the transceiver in conjunction with intellectual property (IP) cores. While extremely flexible, such a method comes at the expense of FPGA logic components. The size of the FPGA configuration and the speed of the

programming interface both affect boot times. As a result, many smaller FPGAs can start up faster than the 100 ms time limit specified in the PCIe base specification [53]. A FPGA-based system can achieve high throughput and low and predictable latency since FPGAs are typically configured to carry out highly particular jobs. Both of these features are desired because the bridge should provide fast communication between the root complex and several LVDS/CMOS ports. The fact that FPGAs may be configured to support virtually any interface is their fundamental advantage over alternative options. FPGAs don't, however, have the same variety of peripherals or software-implemented interfaces. The need for a Non-Volatile Storage (NVS) to save the settings while utilizing an FPGA is one design consideration. The programming of FPGAs differs greatly from MPUs and CPUs due to the operating principle, which is another significant factor to take into account. Hardware Description Languages (HDLs), such as Very High Speed Integrated Circuit HDL (VHDL) or Verilog, are frequently used to program FPGAs. These languages are very different from sequential programming languages.

2. CPU: When data processing and acquisition are necessary, a CPU is frequently a wise choice. They have a track record of dependability and provide fast clock speeds, which are essential for the bridge's deployment. Figure 2.5 depicts a system sketch for a bridge that uses a CPU. The SSI to LVDS/CMOS bridge block represents an optional interface that, if the CPU is unable to do so directly, enables communication with the external contact [56].

An operating system like Linux can be used with many CPUs. This indicates that there are more programming language options available. The fact that few CPUs may be used in an endpoint setup despite the fact

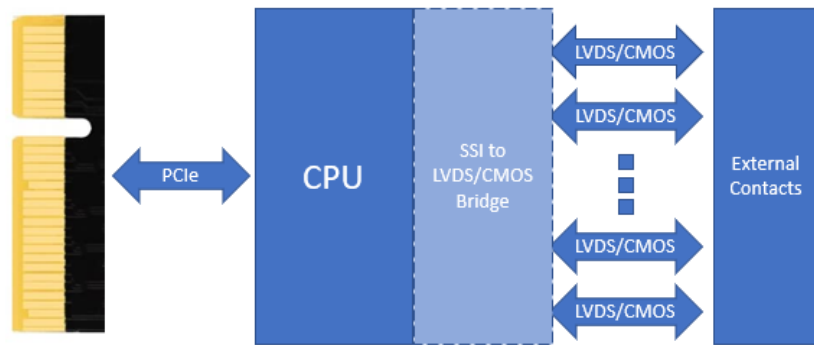


Figure 2.5: The solution’s system sketch when using a CPU with native PCIe capabilities

that many of them support PCIe is a drawback. Texas Instruments and Broadcom are two companies that make endpoint competent CPUs [59]. Another problem is that LVDS output capabilities is frequently absent from CPUs. Fortunately, lots of Central Processing Units (CPUs) are equipped with high-speed serial interfaces, such as Quad Serial Peripheral Interface (Quad SPI), that enable them to establish connections with external devices capable of operating as LVDS/CMOS interfaces.

3. PCIe to PHY interface: If the computing device lacks support for PCIe, an integrated circuit (IC) that includes a PCIe to PHY bridge can be employed to bridge the connection between the PCIe interface and the computing device, as shown in Figure 2.6. This IC acts as an endpoint device and facilitates connection to the root complex. It comprises a Serializer/Deserializer (SerDes) consisting of a Parallel In Serial Out (PISO) block and a Serial In Parallel Out (SIPO) block, which enables bi-directional data transfer between a serial and a parallel interface. Given that the current layers inside the PCIe standard are software-based, the adoption of a PCIe PHY bridge, specifically a PCIe-Ethernet bridge significantly expands the range of CPUs that are readily available [33].

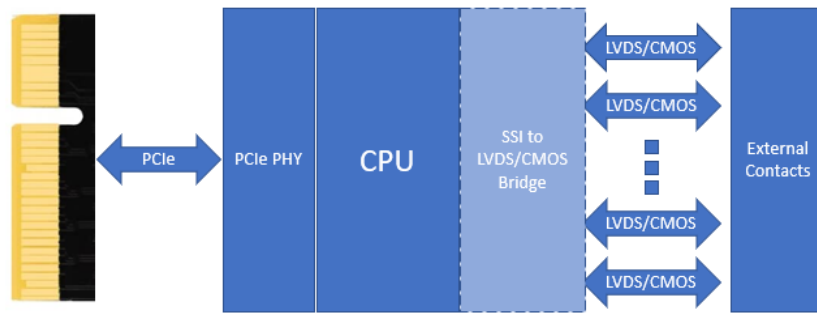


Figure 2.6: The solution’s system sketch when using a CPU without native PCIe capabilities

After going through these three alternatives, there are several points that should be taken into account when choosing the most suitable alternative among them. The programmable logic devices such as FPGAs have various useful characteristics and are usually used in endpoint configuration as most vendors offer endpoint IP cores; however, they might not be easy to use by programmers who are used to working with CPUs. Also, an external memory is required to be used with FPGAs due to its extensive connectivity options. Furthermore, the CPUs, that directly support PCIe, are strong yet complex and slow to boot. However, a CPU, typically a microcontroller with additional PCIe PHY bridge has relatively lower power consumption and can satisfy the requirements mentioned above. Therefore, considering these points, we decided that the most suitable option to consider in this design is using a CPU, typically a microcontroller that is commonly used in CubeSat designs along with a PCIe-Ethernet bridge to enable PCIe communication; however, as we aim to test the PCIe connectivity through this thesis, we aim to simulate the cPCI backplane by using an FPGA with PCIe compatibility as an end-device.

2.3 Selection of Hardware

To choose an appropriate FPGA as well as peripheral components, different manufacturers and product lines were compared. This part will go

over how to identify, evaluate, and select various hardware.

2.3.1 FPGA Selection

A number of product series from several manufacturers were reviewed in order to determine which FPGA would work best for the initial testing of the practicality of the design, being cPCI compatible, and possibly being included in the design in the future to add more computing power to it. The following three FPGA producers were taken into consideration: Intel (formerly Altera), Microsemi, and Xilinx. First, various product series were looked into, and then particular device packages. The product series must be flight-proven and support at least PCIe Gen 2 x1 to be considered as a potential contender. Table 2.1 lists the contenders once they were collected into a list.

Table 2.1: Comparison between flight-proven FPGAs

FPGA	Manufacturer	Logic Elements	DSP Blocks	Memory	Power Consumption (W)
Stratix IV GX	Intel/Altera	530K	2,520	10 GB	69
Stratix V GX	Intel/Altera	1.1M	3,288	10 GB	58
Kintex-7	Xilinx	478K	1,760	4.4 MB	31
Artix-7	Xilinx	215K	740	1.5 MB	14
Cyclone V GX	Intel/Altera	220K	1,288	4.3 MB	7
Cyclone 10 GX	Intel/Altera	220K	1,288	4.3 MB	7
SmartFusion2	Microsemi	120K	166	1.1 MB	2.2
Zynq-7000	Xilinx	85K	220	512 MB	3.5

Among the FPGAs included, the Artix-7 stood among the other alternatives due to its radiation resilience and low power consumption, as it is also a cost-effective FPGA that is frequently utilized in space applications. Numerous space missions, such as the ExoMars mission and the Mars Atmosphere and Volatile Evolution mission, have made use of it. Artix-7 is appropriate for a variety of space applications because it strikes a reasonable mix between performance and price. Artix-7 provides up to 215,000 logic cells, 740 DSP slices, and 52 Mb of block RAM in terms of performance. This is equivalent to other FPGAs that have been used in flight, like the

Kintex-7, Cyclone V GX, and Cyclone 10 GX. In comparison to these FPGAs, Artix-7 also consumes less power, which may be favorable in applications requiring limited power in space. Overall, Artix-7 stands out as a cost-effective and dependable solution that provides a reasonable balance between performance and power consumption, despite the fact that there are a number of flight-proven FPGAs available for space applications.

2.3.2 Microcontroller Selection

There are several flight-proven microcontrollers that can be chosen among based on the specific application requirements. Each microcontroller has its own advantages and disadvantages, and the best choice for a particular application depends on factors such as performance requirements, power consumption constraints, available budget, and other specific features. Table 2.2 presents a comparison between the main flight-proven types of microcontrollers, where it can be seen that the RH850 is a good choice for safety-critical applications due to its hardware-based functional safety features. The STM32 is a good balance between performance, power consumption, and cost, while the MSP430 is a low-power option suitable for battery-powered applications. The PIC32 offers high performance and a wide range of peripherals and interfaces. According to our needs, we have decided to use the STM32 microcontrollers due to the balanced features that they have.

STM32 microcontrollers also have various flight-proven options to choose among, as illustrated in Table 2.3. The STM32L4 is a low-power microprocessor built on the ARM Cortex-M4 architecture. It provides up to 2 MB of flash memory, up to 640 KB of RAM, and a maximum operating frequency of up to 120 MHz. Another low-power microprocessor is the STM32L5, which is built on the ARM Cortex-M33 architecture. It provides

Table 2.2: Comparison between flight-proven microcontrollers

MCU	Architecture	Maximum Operating Frequency	Flash Memory	RAM	Power Consumption
RH850	32-bit RISC	400 MHz	Up to 4 MB	Up to 768 KB	Low
STM32	32-bit ARM Cortex-M	400 MHz	Up to 2 MB	Up to 512 KB	Low-medium
MSP430	16-bit RISC	25 MHz	Up to 512 KB	Up to 64 KB	Very low
PIC32	MIPS	200 MHz	Up to 512 KB	Up to 128 KB	Low-medium

up to 2 MB of flash memory, up to 640 KB of RAM, and a maximum operating frequency of up to 110 MHz. The STM32G4 is a high-performance microprocessor built on the ARM Cortex-M4 architecture. It provides up to 2 MB of flash memory, up to 640 KB of RAM, and a maximum operating frequency of up to 170 MHz. However, a heterogeneous dual-core microprocessor built using the ARM Cortex-A7 and Cortex-M4 architectures is the STM32MP1. It provides up to 1 GB of flash memory, up to 512 MB of RAM, and a maximum operating frequency of up to 800 MHz. With a maximum operating frequency of up to 480 MHz, the STM32H7 microcontroller is the most powerful in terms of performance. Additionally, it provides up to 1 MB of RAM and up to 2 MB of flash memory. High-performance microcontrollers with highest operating frequencies of up to 216 MHz and 180 MHz, respectively, include the STM32F7 and STM32F4. It is also important to mention that all of these microcontrollers have additional security options that include memory protection and error correction. When compared to the other flight-proven STM32 microcontrollers that are listed in the table, STM32H7 provides better performance than most of them, with a higher maximum operating frequency and bigger flash memory and RAM capacities, except the STM32MP1. The STM32H7 microcontroller provides comparable optional safety features and consumes comparable power to other microcontrollers. Therefore, as we

would demand quick processing and lots of memory with lower power consumption than what the STM32MP1 consumes, we have chosen the STM32H7 microcontroller as our main microcontroller.

Table 2.3: Comparison between flight-proven STM32 microcontrollers

MCU	Architecture	Maximum Operating Frequency	Flash Memory	RAM	Power Consumption
STM32H7	ARM Cortex-M7	Up to 480 MHz	Up to 2 MB	Up to 1 MB	Low-medium
STM32F7	ARM Cortex-M7	Up to 216 MHz	Up to 2 MB	Up to 512 KB	Low-medium
STM32F4	ARM Cortex-M4	Up to 180 MHz	Up to 1 MB	Up to 192 KB	Low-medium
STM32L4	ARM Cortex-M4	Up to 120 MHz	Up to 2 MB	Up to 640 KB	Low
STM32L5	ARM Cortex-M33	Up to 110 MHz	Up to 2 MB	Up to 640 KB	Low-medium
STM32G4	ARM Cortex-M4	Up to 170 MHz	Up to 2 MB	Up to 640 KB	Low-medium
STM32MP1	Dual-core ARM Cortex-A7 & Cortex-M4	Up to 800 MHz	Up to 1 GB	Up to 512 MB	Low-medium

2.4 Ethernet

As our system architecture needs to be able to communicate through Ethernet in order to do send data over PCIe using an Ethernet-to-PCIe converter, it is important to discuss the most important aspects that control Ethernet communication. Networking cables and Ethernet cables almost always use RJ45, or Registered Jack 45 connectors. Eight pins, each about 1 mm apart, make up an RJ45 connector, and the wires are crimped together for a strong link, which is called an 8P8C (eight positions - eight contacts). Ethernet Category 3 through Category 6 use the RJ45 specification. Cat 7 Ethernet connections can use RJ45 connectors, but GigaGate45 (GG45) connectors are more frequently used. Fortunately, these are backward compatible with the RJ45, so upgrading to Cat 7 does not require a totally new installation. There are various types of Ethernet cables used as illustrated

in table 2.4.

Table 2.4: Ethernet Cables Categories

Category	Max. Bandwidth	Max. Data Rate	Max. Cable Length	Typical Use Case
Cat3	16 MHz	10 Mbps	100 meters	Voice and data communications in older networks
Cat5	100 MHz	100 Mbps	100 meters	Basic home networking
Cat5e	100 MHz	1 Gbps	100 meters	Home and small business networking
Cat6	250 MHz	10 Gbps	55 meters	High-performance networks, data centers
Cat6a	500 MHz	10 Gbps	100 meters	High-performance networks, data centers
Cat7	600 MHz	10 Gbps	100 meters	Industrial and commercial applications
Cat8	2 GHz	40 Gbps	30 meters	Data centers and high-performance networks

There are two main types of cables used in Ethernet communication, crossover cables and straight through lines which are wired differently from each other. Checking the arrangement of the colored wires inside the RJ45 connector can be a simple method to identify what you have. A straight-through cable is one that has the same arrangement of wires on both ends. If not, it was probably wired improperly or is a crossover connection. Contrary to devices, straight-through cables are typically used mainly for connecting. Additionally, crossover connections are employed when connecting related devices. The Ethernet PHY is a transceiver that connects the analog world to the digital world, which includes processors, FPGAs, and application-specific integrated circuits (ASICs). Over a variety of media, an Ethernet PHY is made to provide error-free transmission over lengths greater than 100 meters. Figure 2.7 displays a sample block diagram of how data is sent to and received from a processor using a standard RJ45 Ethernet cable.

A media access controller is connected to the Ethernet PHY (MAC). The MAC regulates the data-link layer of the OSI model and is typically built

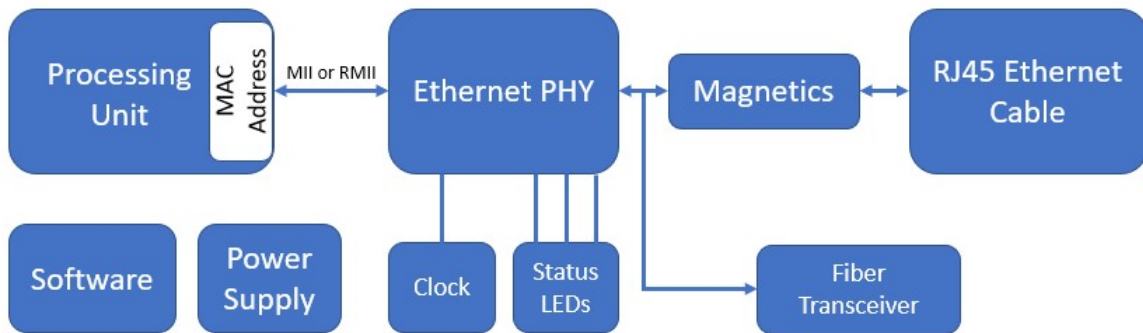


Figure 2.7: A block diagram that represents data transfer using Ethernet

into a processor, FPGA, or ASIC. The link between the MAC and the PHY is specified by the media-independent interface (MII). Depending on system needs, there are MII variants that offer a low pin count and a range of data rates. The two types of interfaces used to connect a PHY to a MAC are MII and RMII:

- **MII:** A media access control (MAC) block for Fast Ethernet (100 Mbit/s) was originally connected to a PHY chip using the media-independent interface (MII), which was initially specified as a standard interface. The MII, which connects various kinds of PHYs to MACs, is standardized by IEEE 802.3u. Being media autonomous allows for the use of various PHY devices (such as twisted pair, fiber optic, etc.) without having to redesign or replace the MAC hardware. As a result, regardless of the network data transmission medium, any MAC may be used with any PHY.
- **RMII:** A standard called reduced media-independent interface (RMII) was created to minimize the amount of signals needed to link a PHY to a MAC. The cost and complexity of network hardware can be decreased by reducing the number of pins, particularly when it comes to microcontrollers with integrated MACs, FPGAs, multiport switches or

repeaters, and PC motherboard chipsets. To accomplish this, four items were modified from the MII standard. These modifications result in roughly half as many signals being used by RMII as compared to MII.

2.5 System Architecture

To restate what was mentioned in the sections above, the proposed system, illustrated in Figure 2.8, consists of up to four layers, providing higher modularity. The four layers are defined as the following:

- Layer 1: A micro-module designed for a specific application. It is mounted on layer 2 boards.
- Layer 2: Boards with SODIMM or Bergstak connectors can also include the mounting of layer 1 modules on them.
- Layer 3: The motherboard that hosts the layer 2 board and interfaces with it using one of the connectors(SODIMM, Bergstack or other high-speed Mezzanine connectors). It follows the PC104 standard that is commonly used in CubeSats.
- Layer 4: It is an external motherboard that usually hosts the layer 3 board and interfaces with it using a high-speed Mezzanine connector. The layer 4 boards usually come in larger form factors (3U, 6U, etc.) and follows the CompactPCI (cPCI) Serial Space. This layer can also be reconfigured to match the SpaceVPX standard.

A detailed diagram that can help in illustrating the system's main elements and how they are interchanged is shown in Figure 2.9.






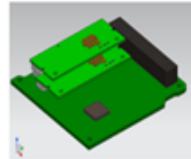

	Layer 1	Layer 2	Layer 3	Layer 4
	Micro-module	Interface boards	PC-104 Boards	Larger form factors (3U, 6U etc.)
Module				
Dimensions	Variable	67.6 mm x 30 mm	90 mm x 96 mm	100 x 160 mm
Standard used	To be Defined	SODIMM, Bergstak, or PCIe	PC104	cPCI Serial Space or SpaceVPX
Interfacing with upper layer (illustrations)				

Figure 2.8: Description of the system's layers

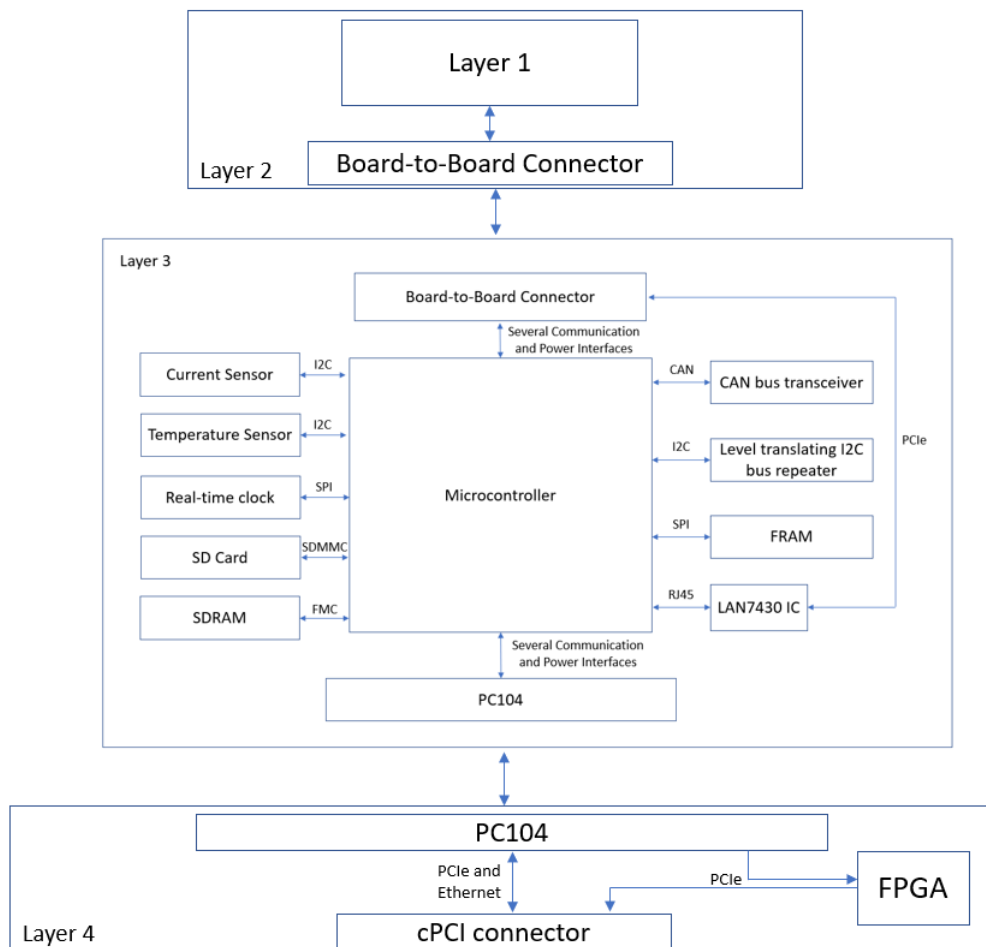


Figure 2.9: The system architecture of the modular CDHS

The proposed system can be done with one of the two configurations shown below. Where the first configuration, illustrated in Figures 2.10 & 2.11, includes the four layers connected to each other through Bergstak connectors. Also, the practical demonstration of this configuration is through Zigbee communication. This configuration can be used for transmission speeds that are up to 6 GB/s.

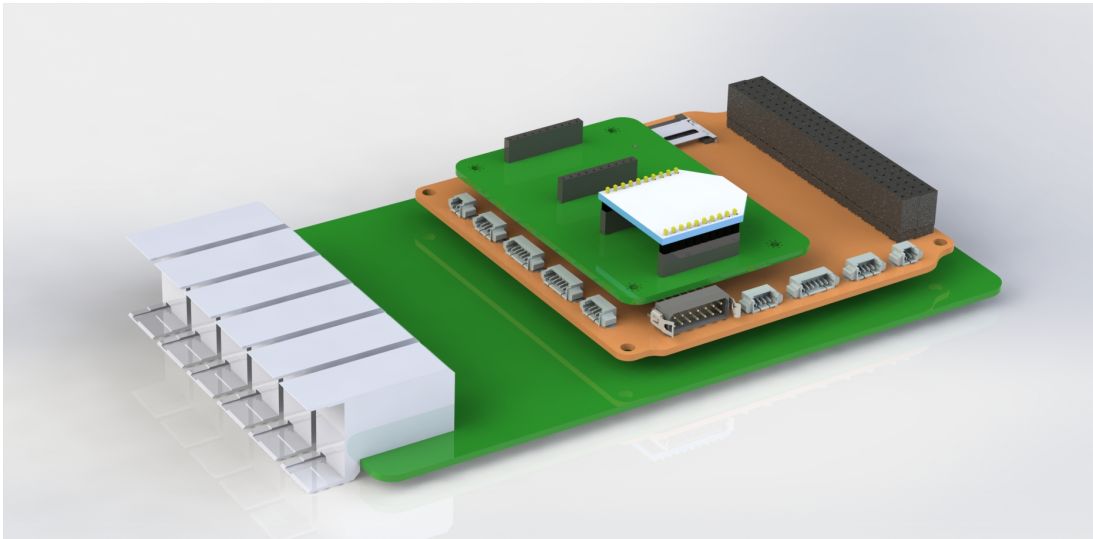


Figure 2.10: The system architecture of the modular CDHS (using BergStak connectors)

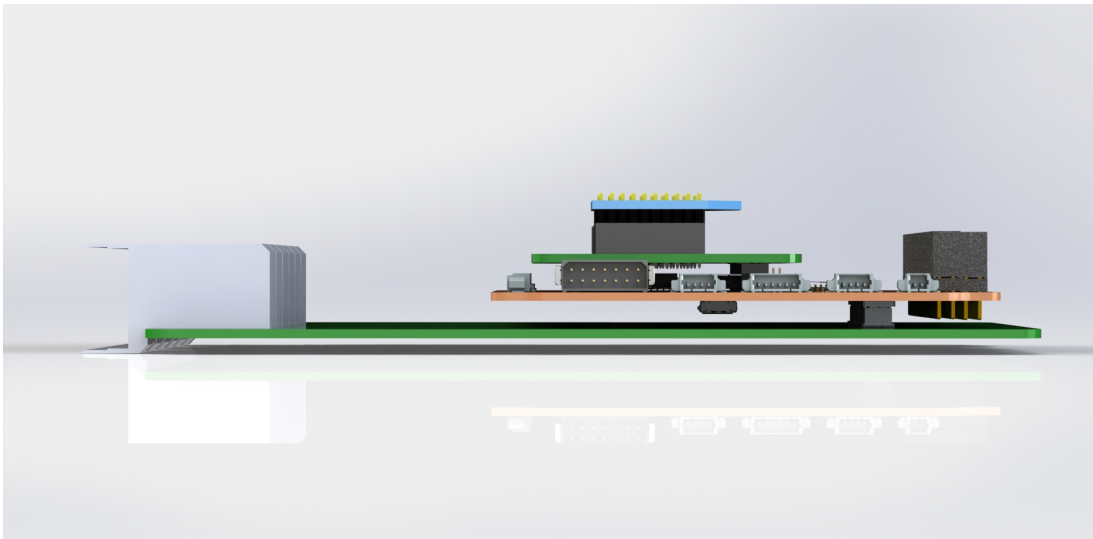


Figure 2.11: The system architecture of the modular CDHS (using BergStak connectors)

However, the second configuration, illustrated in Figures 2.12 & 2.13, can be used for scenarios that require higher transmission speeds, typically up to 32 GB/s or even more, using the PCIe connector between the layers.

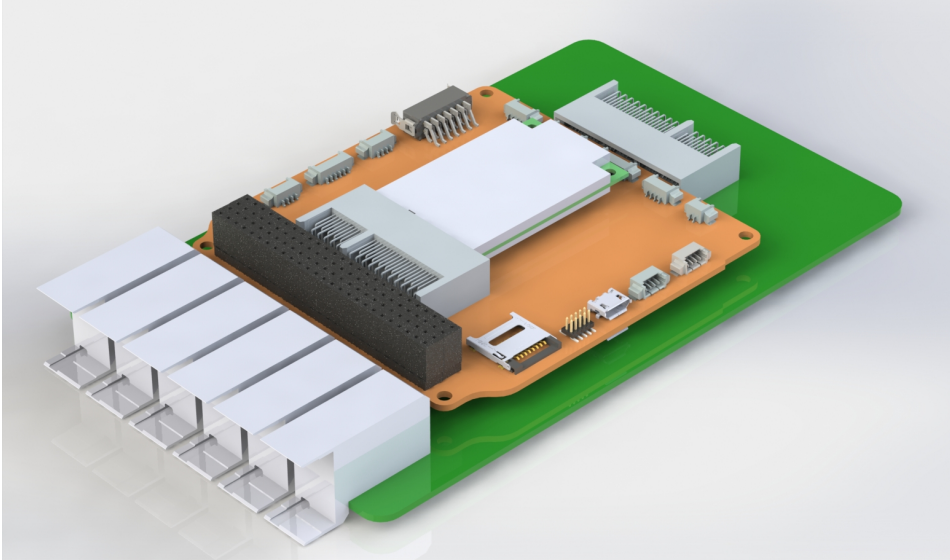


Figure 2.12: The system architecture of the modular CDHS (using PCIe connectors)

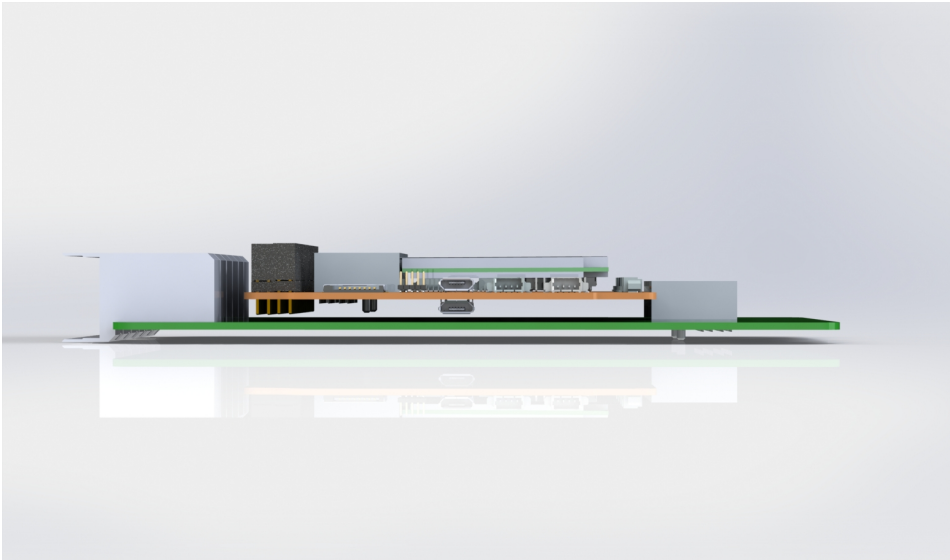


Figure 2.13: The system architecture of the modular CDHS (using PCIe connectors)

2.6 PCB Design

To manufacture the PCBs mentioned in the modular architecture, several manufacturers have been considered; however, JLCPCB was chosen

for manufacturing most of the PCBs, as they provide rapid prototyping with low minimum quantities on orders. To guarantee that the PCB gets manufactured correctly, it is crucial to abide by the manufacturer’s guidelines and opt for a stackup that is compatible with JLCPCB’s production process. The 1.6 mm JLC2313 stackup used in the PCB design is listed in Table 2.5. Despite the fact that JLCPCB offers numerous stackups with various prepreg thicknesses and dielectric constants, we chose the JLC2313 stackup due to its wide core and 6-layer stackup. Since the majority of the ground and power planes are located in the two inner planes, this enables greater inter-plane capacitance [29].

Table 2.5: JLC2313 layer stackup

Layer	Material Type	Thickness (mm)	Dielectric Constant
Top solder mask	Solder mask	0.0127-0.0203	3.8
Top layer 1	Copper	0.035	
Prepreg	2313	0.1	4.05
Inner layer 2	Copper	0.0175	
Core		0.565-0.6	
Inner layer 3	Copper	0.0175	
Prepreg	2116	0.127	4.25
Bottom layer 4	Copper	0.035	
Core		0.565-0.6	
Inner layer 5	Copper	0.0175	
Prepreg	2313	0.1	4.05
Bottom layer 6	Copper	0.035	
Bottom solder mask	Solder mask	0.0127-0.0203	3.8

2.7 Layout

There are several points that should be taken into consideration when designing PCBs that include PCIe lanes. As illustrated in Figure 2.14, the PCIe signals are transmitted as differential pairs, and for the best performance, they need a close coupling and an environment with a controlled impedance. To reduce crosstalk and maintain a constant impedance along the whole length of the trace, the differential pairs should be routed as closely as possible. Also, PCIe signal lanes should be routed in a straight or gently curved path, avoiding sharp bends that might cause

reflections and signal degradation, resulting in errors and reduced performance. Furthermore, it is essential to make sure that each differential pair's trace lengths match as closely as possible. Any length difference can result in signal skew, which can result in errors and poor performance. Moreover, PCIe signals need to be routed on separate signal layers from ground and power planes [2]. The isolation reduces crosstalk and interference. To preserve signal integrity when a signal layer is shared with other signals or planes, guard traces or shields must be used. Besides, to stop other signals or components from interfering with PCIe signals, keep out zones should be placed around them. The keep out zones ought to be wide enough to allow for sufficient clearance and shield against unintentional contact with other traces or parts. Finally, to avoid reflections and signal degradation, the PCIe signals should be correctly terminated at both ends. The termination should be positioned as close to the receiver as possible and should be impedance-matched to the trace [55]. Figure 2.15 presents some of the manufactured boards that represent layers 1, 2, and 3, mentioning that the layer 1 module included is a COTS DIGI XBee3 module.

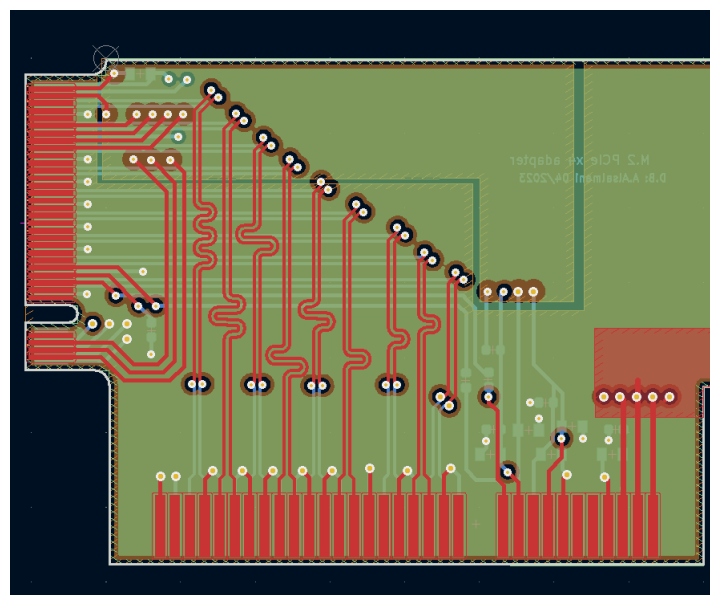


Figure 2.14: PCIe Layout

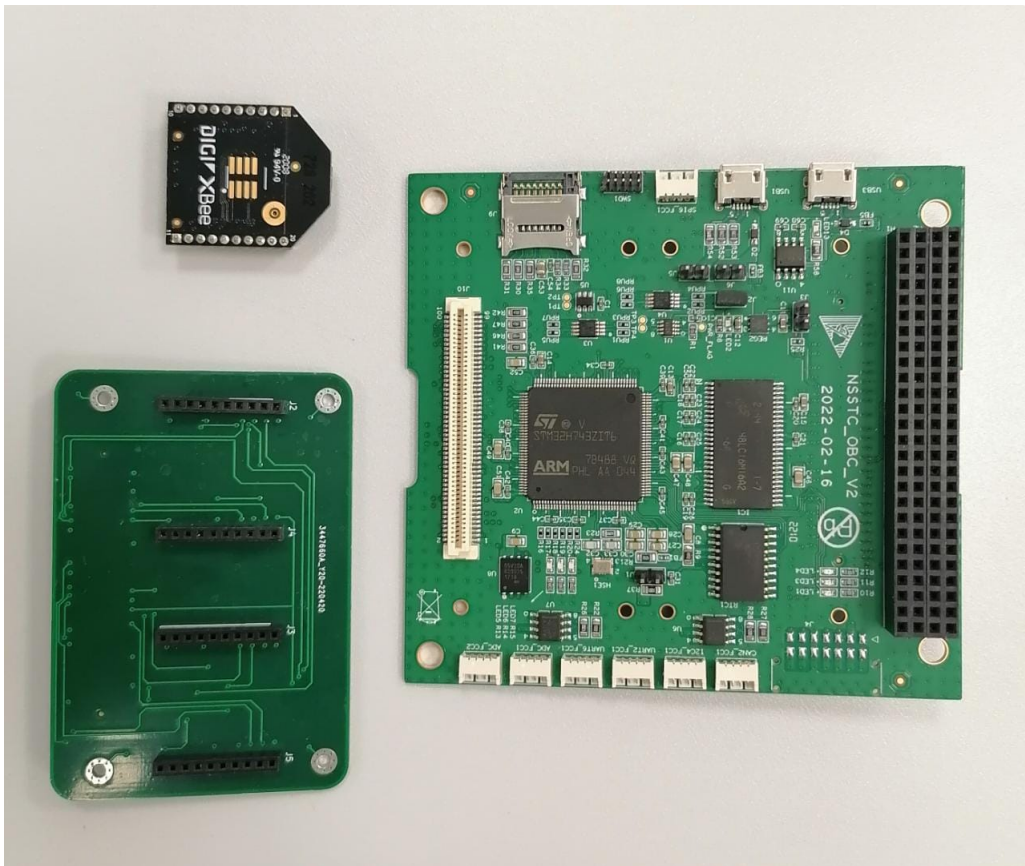


Figure 2.15: Representation of the individual modules of the system's layers

Chapter 3: Experimental Testing and Results

In this section, we describe the different experiments that were done to prove the reliability of the proposed modular system. These experiments include the functional tests, transmission tests, thermal vacuum test, vibration test, and other tests. The results of each of these tests is described in details in the following sections.

3.1 Communication Tests

This section describes the communication tests that were done to compare the performance of our available interfaces with the actual data rate characteristics of each of them, which was measured using a logic analyzer.

3.1.1 UART Communication

UART (Universal Asynchronous Receiver-Transmitter) is a type of serial communication protocol commonly used for board-to-board communication. It is a simple and reliable way to transmit data between two devices over a single wire. UART works by sending data in a series of bits, typically one byte at a time. It uses two wires for communication: one for transmitting data (TX) and one for receiving data (RX). The data is transmitted asynchronously, which means that there is no fixed timing relationship between the sender and receiver. To use UART for board-to-board communication, both devices must be configured with the same baud rate, which is the speed at which the data is transmitted because the UART is asynchronous. The baud rate is typically set in software, and the most common baud rates are 9600, 19200, 38400, and 115200 [45]. In the following UART test, we have set the UART at 115200 baud rate. One advantage of UART is that it is a simple and widely supported protocol, so it can be easily integrated into many different types of devices. Another

advantage is that it requires very little overhead, so it is often used in applications where data throughput is not a critical factor. However, one limitation of UART is that it is relatively slow compared to other communication protocols. The maximum baud rate for UART is typically around 4 Mbps. Additionally, UART is not well-suited for applications that require long-distance communication or immunity to high levels of noise, as it does not have built-in error correction or detection mechanisms. In the UART communication, the test was done using two of the PC/104 computing unit boards. Figure 3.1 illustrates the connections done between the two PC/104 computing unit boards, connecting the UART TX and RX pins to each other in cross-over configuration.

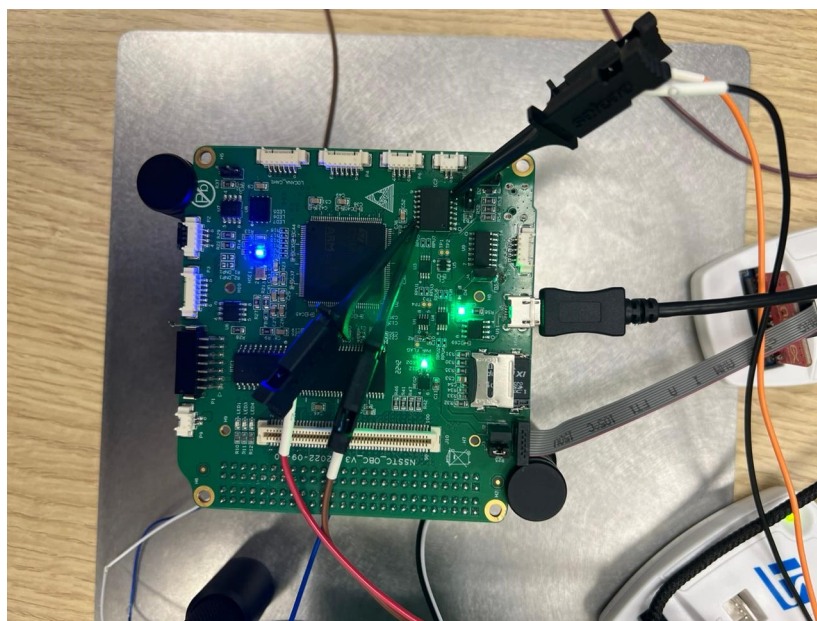


Figure 3.1: UART communication setup between two PC/104 computing units

As shown in Figure 3.2, the UART transmission and reception was successful using two of our PC/104-based computing units and each byte takes $82.4\mu\text{s}$ to be transmitted, which results in a communication speed of 94.8125 kbps.

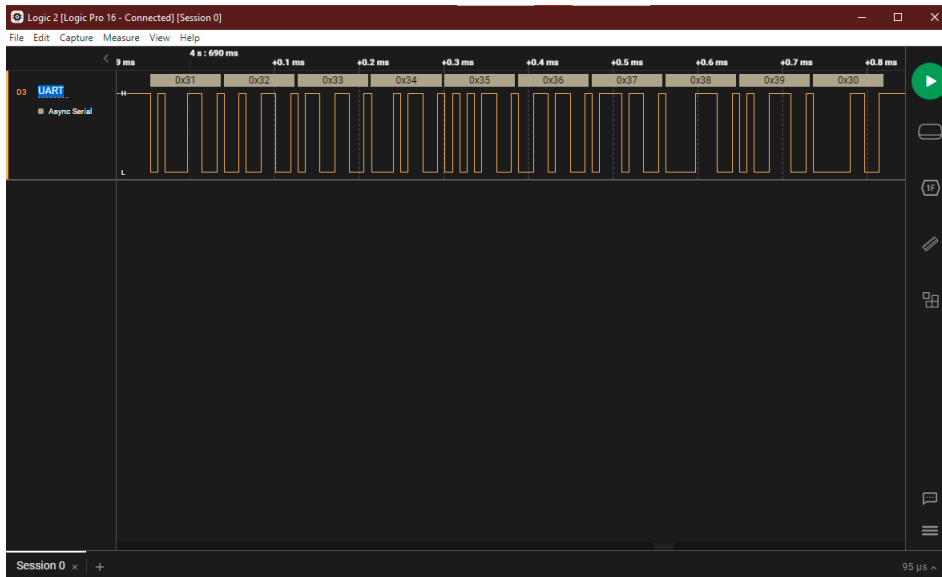


Figure 3.2: UART communication test results

3.1.2 I2C Bus Communication

The I2C component is a two-wire hardware interface developed by Philips that supports I2C Slave, Master, and Multi-Master configurations. It can be used to network multiple devices on a single board or small system with a single master and multiple slaves, multiple masters, or a combination of masters and slaves. The I2C component is compatible with other third-party slave and master devices and supports standard clock speeds up to 1000 kbps. The I2C data signal is serial data (SDA), and it should be configured as Open-Drain-Drives-Low. The master-generated I2C clock is serial clock (SCL), and it should also be configured as Open-Drain-Drives-Low. The UDB version requires a clock to provide 16 times oversampling, which is available when the Implementation parameter is set to UDB. The I2C block can be reset with the reset input available only when the Implementation parameter is set to UDB. If the reset pin is held to logic high, communication over I2C stops. Slave Address is a parameter that selects the I2C address recognized by the slave, and the default is 4. A slave address between 0 and 127 (0x00 and 0x7F) can be selected, and the value may be entered as decimal or hexadecimal. This component provides a

parameter that allows choosing between software and hardware address decoding. Hardware is the default, and it automatically NAKs addresses that are not its own without CPU intervention [62]. The UDB Clock Source parameter allows choosing between an internally configured clock and an externally configured clock for data rate generation. When set to Internal Clock, the clock frequency is calculated and configured by PSoC Creator based on the Data Rate parameter and taking into account 16 times oversampling. Master and Multi-Master operation are similar, but in Multi-Master mode, the program must wait until the current operation is complete before issuing a start transaction, and two masters can start at the exact same time, which leads to arbitration loss. The I2C master has two options: manual and automatic. In automatic mode, a buffer is created to hold the entire transfer. The initial test of the I2C transmission involved internal components, including a temperature sensor and current sensor. However, to achieve higher data rates, board-to-board communication was employed via the I2C protocol, where the setup is illustrated in Figure 3.3. Subsequently, the results of the test were found to be successful, as shown in Figure 3.4, as each byte took about 200 μ s to be transmitted, resulting in a communication speed of 38.6911 kbps.

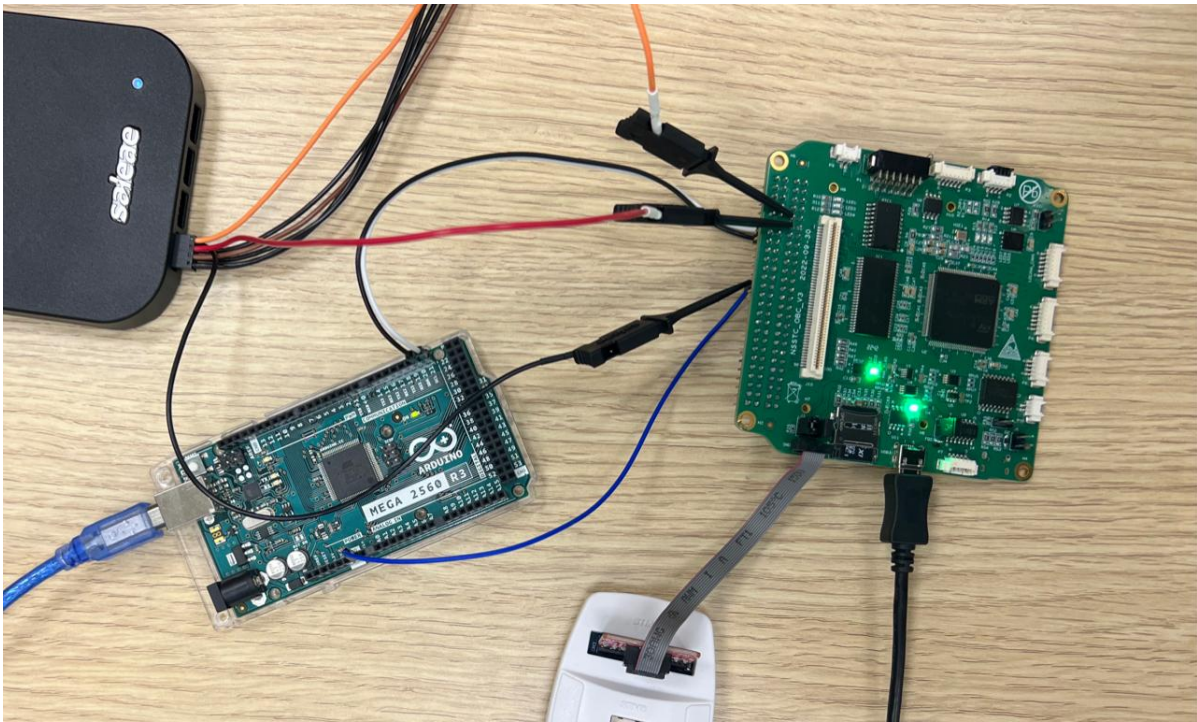


Figure 3.3: I2C bus communication setup between an PC/104 computing unit and an Arduino Mega board

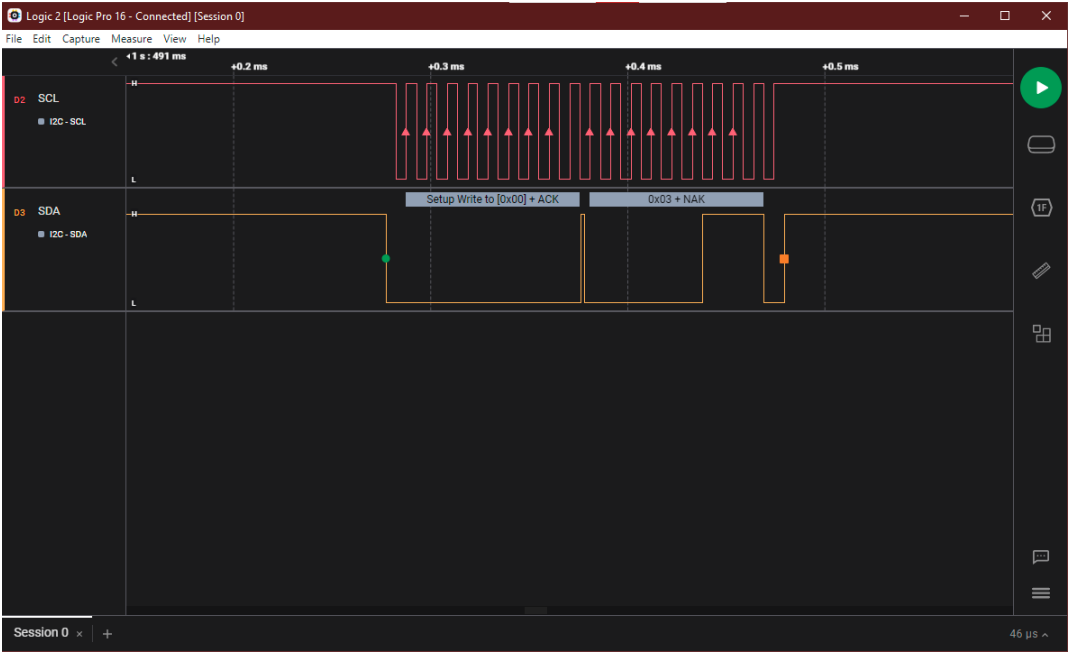


Figure 3.4: I2C bus communication results

3.1.3 SPI Communication

SPI (Serial Peripheral Interface) is a communication protocol commonly used in embedded systems to enable communication between microcontrollers and peripheral devices. SPI is a full-duplex, synchronous serial communication interface, which means that data is transmitted simultaneously in both directions and at a fixed timing between a master and one or more slave devices. One of the advantages of SPI is its capability to achieve high-speed data transmission rates, which is particularly useful in applications where real-time data processing is required. To achieve high-speed data transmission using SPI, there are several techniques that can be employed, such as increasing the clock frequency, optimizing the data transfer protocol, and minimizing the signal propagation delay [62]. Increasing the clock frequency is the most straightforward way to achieve high-speed data transmission in SPI. The clock frequency determines the rate at which data is transferred between the master and the slave devices. By increasing the clock frequency, the data transfer rate can be increased, and more data can be transmitted in a shorter time period. Optimizing the data transfer protocol can also help to achieve higher data transmission rates in SPI. This involves reducing the overhead associated with data transmission, such as the number of bits used to indicate the start and end of data transmission [1]. By minimizing the overhead, more data can be transmitted in a shorter time, resulting in higher data transmission rates. Minimizing signal propagation delay is also critical to achieving high-speed data transmission in SPI. Signal propagation delay refers to the time it takes for a signal to travel from the master to the slave device, and vice versa. By minimizing this delay, the overall time required for data transmission can be reduced, resulting in higher data transmission rates. To test SPI communication, data was transmitted from an STM32-based MCU to a flash

memory, as illustrated in the setup in Figure 3.5.

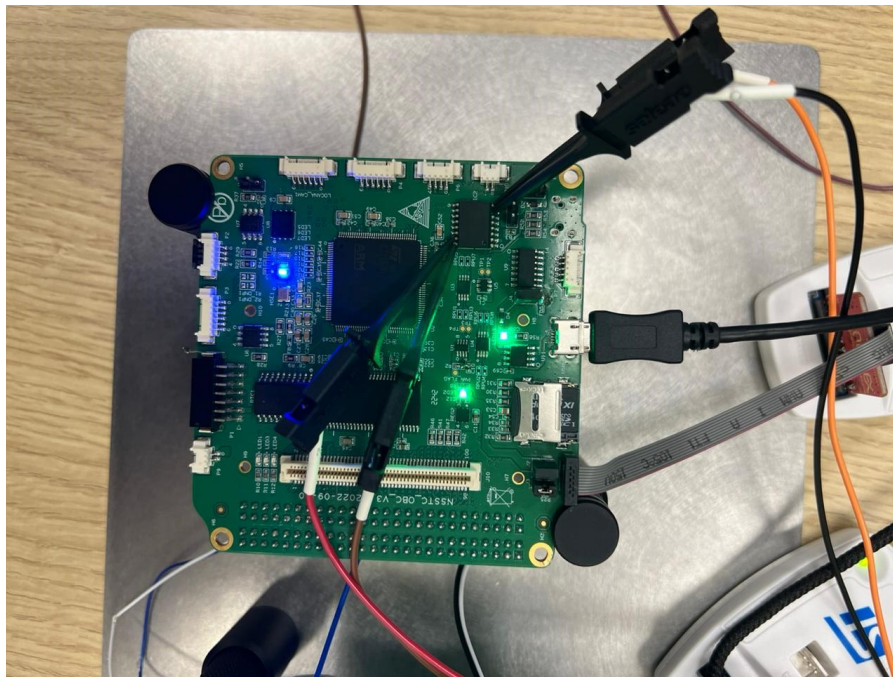


Figure 3.5: SPI communication setup between a flash memory and an MCU

Furthermore, as presented in Figure 3.6, the data transmission over SPI was successful using an STM32-based MCU and a flash memory, as each byte takes $7.6075\mu\text{s}$ to be transmitted, which results in a communication speed of 1.0515938216 Mbps. Also, as illustrated through the test results, the first part of the signal informs the memory that we want to write to it, the second includes three address bytes, and the third includes four data bytes.

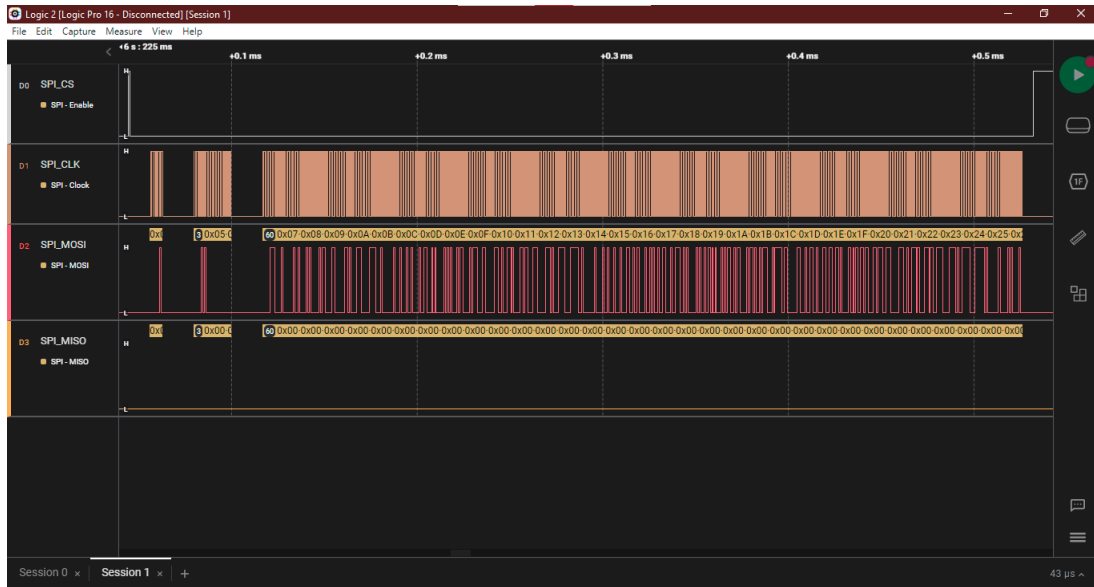


Figure 3.6: SPI communication results

3.1.4 CAN Bus Communication

The CAN bus protocol has two different physical layers, namely low-speed and high-speed, which have different architectures. The low-speed CAN is based on the ISO-11898-3 CAN standard with a speed of 128 kbps and is terminated at each node by a 100Ω resistor. On the other hand, high-speed CAN is based on the ISO-11898-2 CAN standard with a speed of 512 kbps and is terminated at each end of data buses by a 120Ω resistor. To transmit data to the transceiver, the microcontroller would have a CAN controller where data is driven from/to the bus. Moreover, the CAN protocol has important characteristics that make it more reliable for real-time systems. For instance, the transmission privileges higher priority messages to transmit before lower priority messages to ensure important priority tasks are responded on time without delay. The CAN protocol uses a lossless bitwise arbitration method to synchronize the sample data bit on the CAN network. During transmission, the transmitters check the ID bits on the CAN network, and the node with a recessive (1) bit is stopped if there is a node with a dominant (0) bit on the bus. This ensures that higher priority messages are not delayed by lower priorities, and the higher priority ID has a lower

number. The CAN bus protocol has three different layers, namely Object, Transfer, and Physical Layer, each responsible for different functions. For instance, the Object layer filters the noise on the CAN network, while the Transfer layer transmits messages, detects errors, and performs other functions for transmitting messages between nodes on the CAN network. There are four types of frames in the CAN protocol: Data Frame, Remote Frame, Error Frame, and Overload Frame. The Data Frame has two different message formats due to having two different identifiers. The Remote Frame requests data from the source by transmitting RTR-bit as a recessive (1) bit and no data field. The Error Frame is transmitted if any frame detects an error, while the Overload Frame is transmitted to delay transmitting if the receiver requires a delay before receiving the next data frame or remote frame or a higher priority message requires the bus [18]. In summary, the CAN bus protocol has privileges for high priority messages, which are essential for real-time systems without delay. Additionally, the CAN protocol provides high-speed transmission and is convenient for connecting nodes. These are the undeniable reasons why the CAN protocol dominates in the automobile industry and has become widely used in embedded software fields. The high-speed mode of the CAN bus was initially tested using the internal loopback, which was found to be successful. Subsequently, board-to-board communication was carried out using the high-speed mode of the CAN bus, using the setup presented in Figure 3.7, and the results of the test were also successful resulting in a communication speed of 842.5334112 kbps, as each byte takes $9.495\mu\text{s}$ to be transmitted and received, as presented in Figure 3.8.

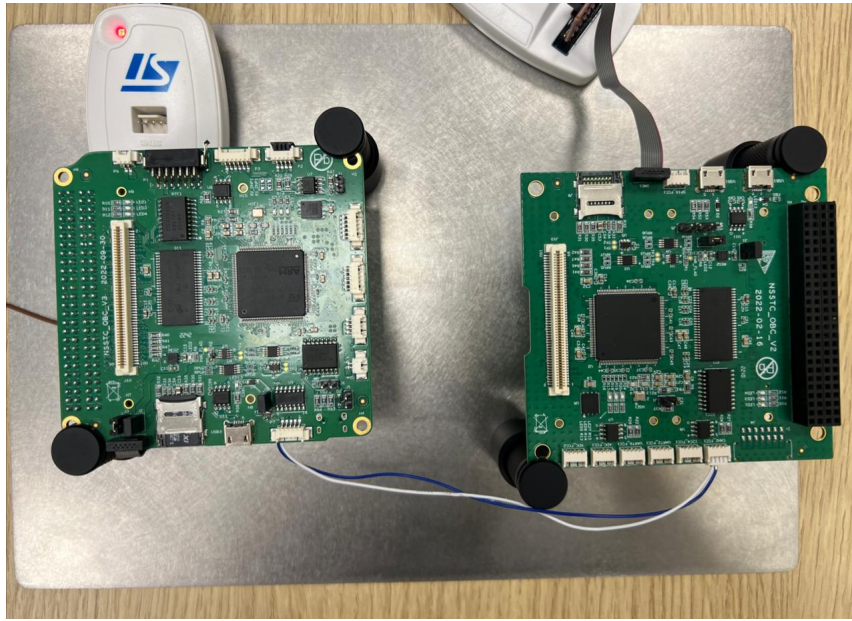


Figure 3.7: CAN bus communication setup between two PC/104 computing unit boards

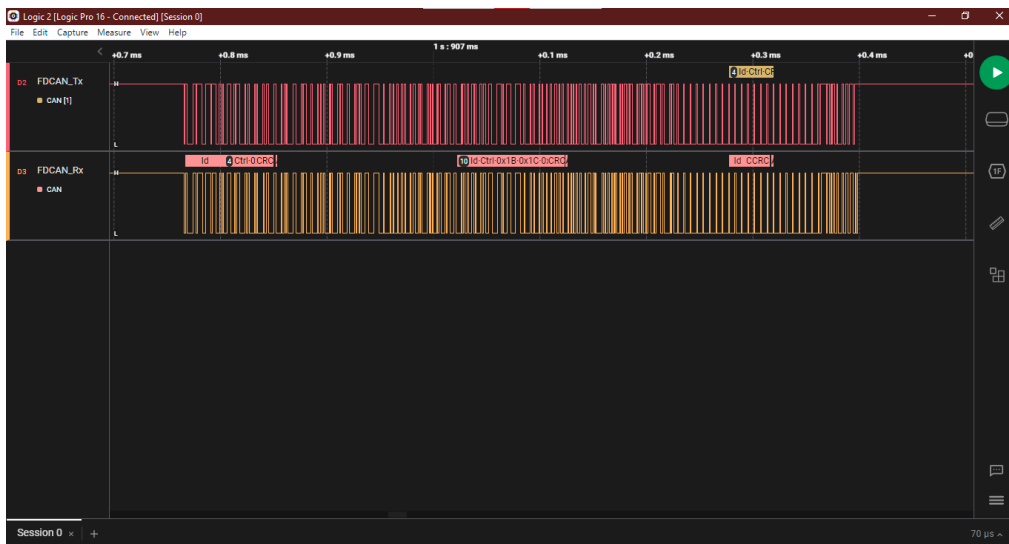


Figure 3.8: CAN bus communication results

3.1.5 Ethernet Communication

This subsection aims to test the Ethernet communication between a PC (server) and an STM32 microcontroller (client). To do so, an STM32F746ZG Nucleo board was used. The testing aimed to demonstrate the importance of configuring IP addresses and traffic flow through network layers to ensure successful data transfer between devices. Several challenges were encountered throughout the testing process, including setting up the IP

addresses for both the server and client devices, configuring network layers to allow for proper traffic flow, and ensuring proper network security protocols were in place. These challenges required careful troubleshooting and attention to detail to overcome. The testing utilized the Hercules software to visualize the data being transmitted and received in real-time, providing a clear representation of the success of the data transfer.

Firstly, the adapter settings were changed, as the Internet Protocol Version 4 (TCP/IPv4) settings were edited as shown in Figure 3.9. Next, the firmware in the STM32F746ZG Nucleo board was flashed and the IP address 192.168.1.11 was verified. Then, the command prompt on Windows was used to ping the ip address of MCU with this command: **192.168.1.11**. Once this was verified, we moved to the second phase of the testing.

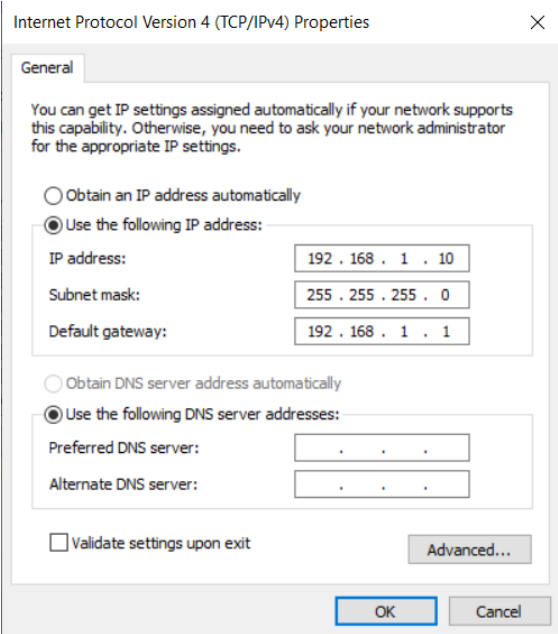


Figure 3.9: The IPV4 Settings

In the second phase of the testing, the lwIP stack was initialized and a TCP client that sends a string to a remote server and waits for a response was created. The Lightweight IP (lwIP) stack is a widely used open-source TCP/IP

stack designed for embedded systems. To send data over Ethernet, Figure 3.10 shows the Command Prompt view along with the executed ipconfig command that is used to verify the windows ip address which shall be the same IP address for TCP server created on Hercules.

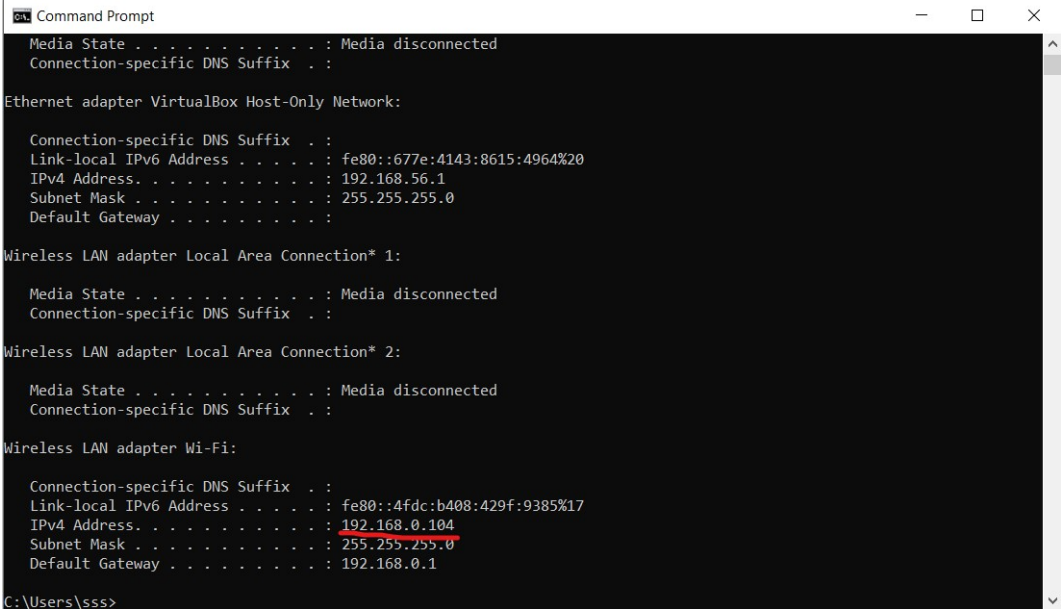


Figure 3.10: The Command Prompt (cmd) view

Next, the TCP server port number and IP address shown in Figure 3.10 are added to the http_test.c file in the dedicated section in the firmware to connect to the server, as illustrated in Figure 3.11.


```

http_test - Notepad
File Edit Format View Help
#include <string.h>
#include "lwip/netif.h"
#include "lwip/inet.h"
#include "lwip/tcpip.h"
#include "lwip/sockets.h"
#include "lwip/netdb.h"
#include "lwip/ip_addr.h"
#include "lwip/tcp.h"
#include "TRACE.h"
#include <stdbool.h>

#define TCP_SERVER_IP_ADDRESS "192.168.1.68" // change this ip address as per your windows ip address to which MCU is connected
#define TCP_CONN_PORT 23

struct tcp_pcb *testpcb;

static void tcp_client_close(struct tcp_pcb *pcb)
{
    err_t err;

    if (pcb != NULL)
    {
        tcp_sent(pcb, NULL);
        tcp_err(pcb, NULL);
        err = tcp_close(pcb);
        if (err != ERR_OK)
        {
            /* Free memory with abort */
            tcp_abort(pcb);
        }
    }
}

```

Figure 3.11: TCP server port number and IP address added to the http_test.c file

There are two options of sending the data, one is through a code that is compiled to the STM32 MCU and the other is by using the Hercules TCP server to send data that is written manually. Figure 3.12 presents the Hercules TCP server window of the first option where the script that flashes the firmware in the Nucleo-746ZG board and enables it to send data is used.

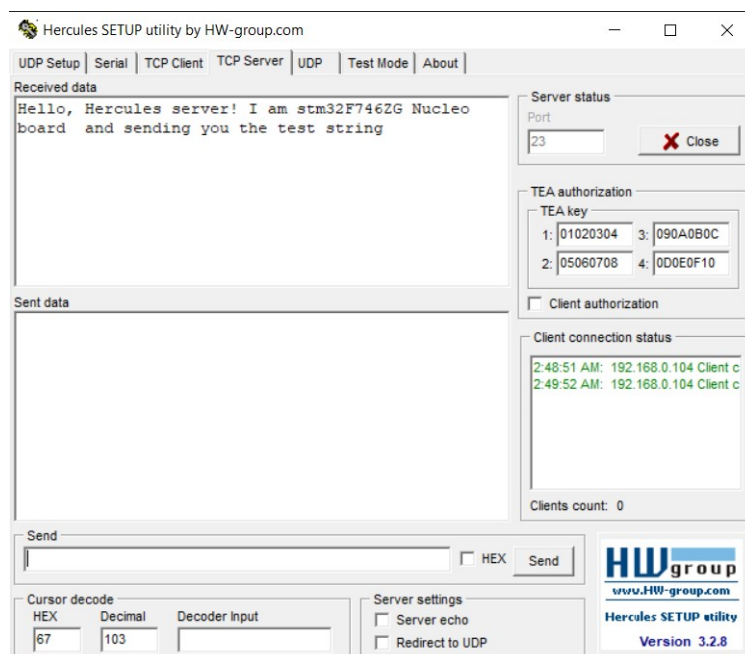


Figure 3.12: Running the script that writes to the PC

While the second option is the one that sends data directly through the

Hercules TCP server window, as shown in Figure 3.13.

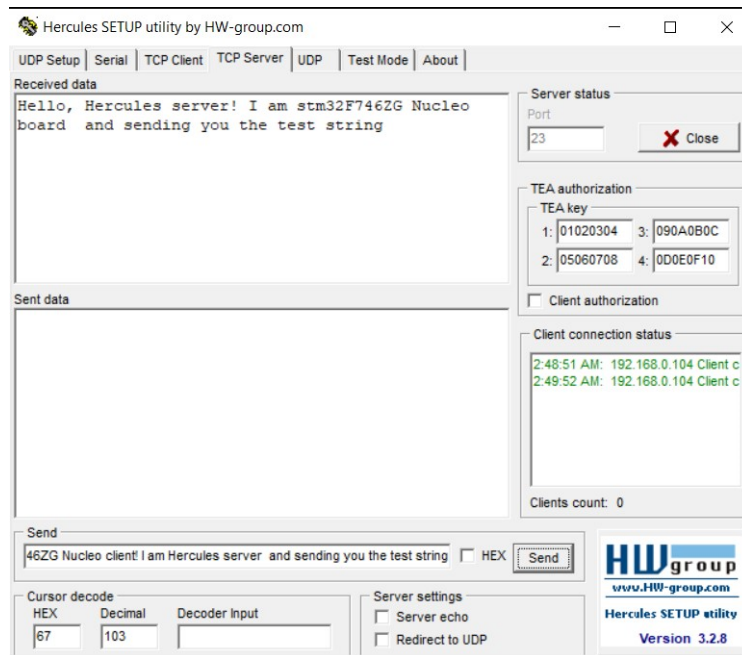


Figure 3.13: Writing data manually

3.1.6 PCIe Communication

The FPGA firmware was developed in the Xilinx Vivado Design Suite HLx 2018.3, a comprehensive development environment for VHDL programming. The suite is equipped with several valuable features, such as simulation and debugging capabilities, and a wide range of pre-programmed Intellectual Property (IP) cores that can aid in accelerating the development process. The firmware was designed using a block design structure in VHDL, which is a graphical method of representing hardware descriptions using interconnected blocks [37]. Figure 3.14 presents an overview of the simplified PCIe VHDL block design [53].

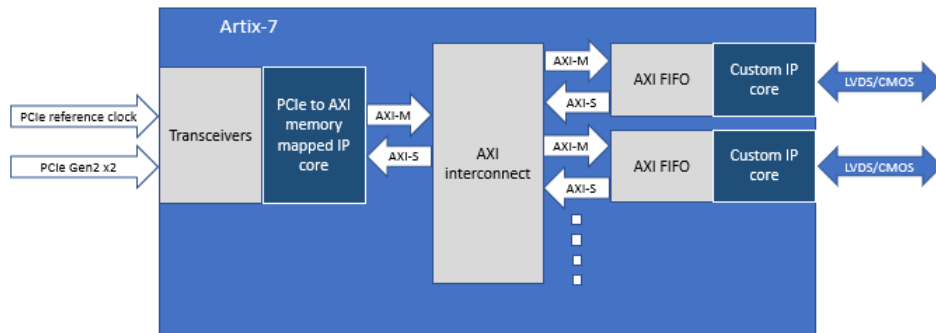


Figure 3.14: The PCIe VHDL block design overview

3.1.6.1 AXI4 Protocol

The ARM AMBA specification includes the AXI protocol, which has different variations and is used for high-performance communication from a master to a slave [64]. Xilinx uses AXI4 as their preferred communication protocol, and a VHDL solution combining AXI4 and AXI4-Lite interfaces is presented in the thesis. Each block has its own dedicated address span [41]. AXI interconnects can be used to connect multiple slaves to a single master or vice versa, simplifying the design and improving performance.

3.1.6.2 PCIe

The PCIe functionality was implemented using the AXI Memory Mapped to PCIe IP core. The core acts as an intermediary between PCIe and AXI4, converting packets between the two protocols. Block Address Registers (BARs) are used to map data to memory. Each VHDL block is assigned a BAR that contains information for data routing through address translation, ensuring it reaches the intended destination [3]. The IP core has inputs and outputs that must be constrained to the physical ports of the FPGA associated with the M.2 PCIe pins. Once the device is programmed and a platform device requests a PCIe enumeration, the system detects the device's PCIe Endpoint functionality and reports the correct configuration indicating that the link is operating as expected.

3.1.6.3 Clock Buffers

To establish a PCIe link, a reference clock is required as per the PCIe standard. The M.2 connector provides this clock signal as part of the standard. However, to use the clock signal within the device's internal circuitry, it needs to be converted from a differential signal to a suitable format [4]. To accomplish this conversion, a Utility Buffer IP is employed, which converts the differential inputs into a buffered single-ended clock signal. The resulting signal is then connected directly to the reference clock input of the PCIe IP [11].

3.1.6.4 Block RAM

This thesis implements a FIFO buffer to store data using a block RAM (BRAM) on an FPGA, which can have different sizes. The BRAM has two ports, Port A and Port B, each with its own clock, that access the same data. Port A is for writing data, while Port B is for reading data via GPIOs. Data is loaded into the BRAM at a user-defined address location through the PCIe to AXI4 link [42]. The BRAM is generated using the IP Block RAM Generator, which optimizes memory allocation. The BRAM is connected to the AXI4 interface through the Block RAM Controller IP, which translates AXI4 to the BRAM interface and provides direct access to memory via AXI4.

3.1.6.5 Custom IP

To test the functionality of the PCIe Bridge, a custom IP block, as shown in Figure 3.15, was created with three main objectives: to read a control register from the AXI-bus, to read the data register, and to shift the data on the output. Vivado includes a utility to easily create custom AXI peripherals which takes care of the AXI communication and leaves the user to

implement the custom logic.

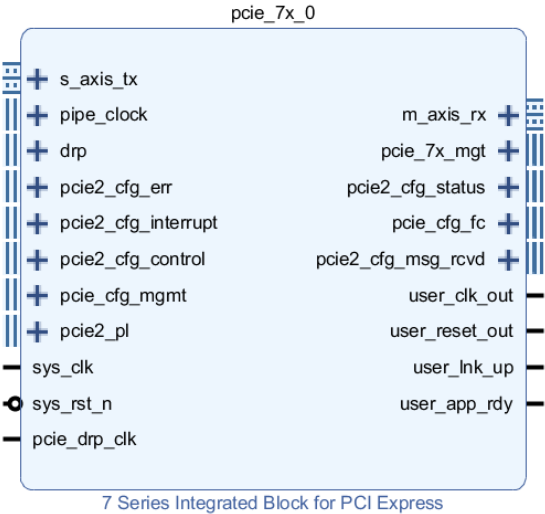


Figure 3.15: Basic PCIe IP block on Vivado design suite

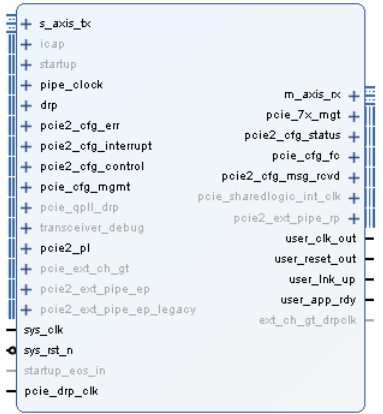
The data rate of the output is adjustable via a configurable PLL, resulting in synchronous serial communication suitable for testing. The bridge generates a 50 MHz square wave by sending 1111000011110000... with a 400 MHz data rate, while 110011001100... generates a 100 MHz square wave. The LVDS/CMOS side was tested up to a maximum frequency of 200 MHz, and a customized PCIe block was created to supply up to 5 GT/s link speed, as illustrated in Figure 3.16.

Re-customize IP

7 Series Integrated Block for PCI Express (3.3)

Documentation IP Location

Show disabled ports



Port list (Left):

- + s_axis_bc
- + loap
- + startup
- + pipe_clock
- + dp
- + pcie2_cfg_err
- + pcie2_cfg_interrupt
- + pcie2_cfg_control
- + pcie2_cfg_mgmt
- + pcie_qpll_drp
- + transceiver_debug
- + pcie2_pl
- + pcie_ext_ch_gt
- + pcie2_ext_pipe_ep
- + pcie2_ext_pipe_ep_legacy
- sys_clk
- sys_tst_n
- startup_eos_in
- pcie_drp_clk

Port list (Right):

- m_axis_rc
- pcie7x_mgt
- pcie2_cfg_status
- pcie_cfg_fc
- pcie2_cfg_msg_rouv
- pcie_sharedlogic_int_clk
- pcie2_ext_pipe_rp
- user_clk_out
- user_reset_out
- user_lnk_up
- user_app_rdy
- ext_ch_gt_drpclk

Component Name:

Basic | IDs | BARs | Core Capabilities | Interrupts

Mode: Basic

Device Port Type: PCI Express Endpoint device | Xilinx Development Board: None

PCIe Block Location: X0Y0 | Silicon Revision: GES and Production

Number of Lanes | **Maximum Link Speed**

Lane Width: X4 | 2.5 GT/s | 5.0 GT/s

AXI Interface Frequency | **AXI Interface Width**

Frequency (MHz): 250 | 64 bit | 128 bit

Reference Clock Frequency (MHz): 100 MHz

Tandem Configuration

None | Tandem PROM (Refer PG054) | Tandem PCIe (Refer PG054)

PIPE Mode Simulations

None | Enable Pipe Simulation | Enable External PIPE Interface

Enable External STARTUP primitive | Enable External GT Channel DRP

Additional Transceiver Control and Status Ports

OK Cancel

This opt

Figure 3.16: Customized PCIe block to supply link speed of 5 GT/s

3.1.6.6 Constraints

In the design of FPGA circuits, constraints play a crucial role in defining the behavior of each pin. These constraints are used to connect the internal ports of the FPGA to the external pins and can determine the data flow direction, voltage levels, and termination type. Additionally, certain pins are assigned specific functions, and constraints related to these pins are limited to their intended use.

3.1.6.7 Simulation and Debug

The Vivado design suite offers the ability to simulate either individual blocks or the entire firmware, making development faster. Simulations display input and output signals, while the debug core shows signal values on the actual FPGA. Additionally, the suite includes a debug IP called Virtual Input/Output (VIO) that lets the user read or set specific ports in real-time [69]. To simulate the PCIe IP block before uploading it to the FPGA, several steps were taken. The .xdc file from the "Constraints" category was edited before generating the bitstream and checking the simulation results. During initial simulation, various variables were checked to ensure proper functionality, as shown in Figure 3.17.

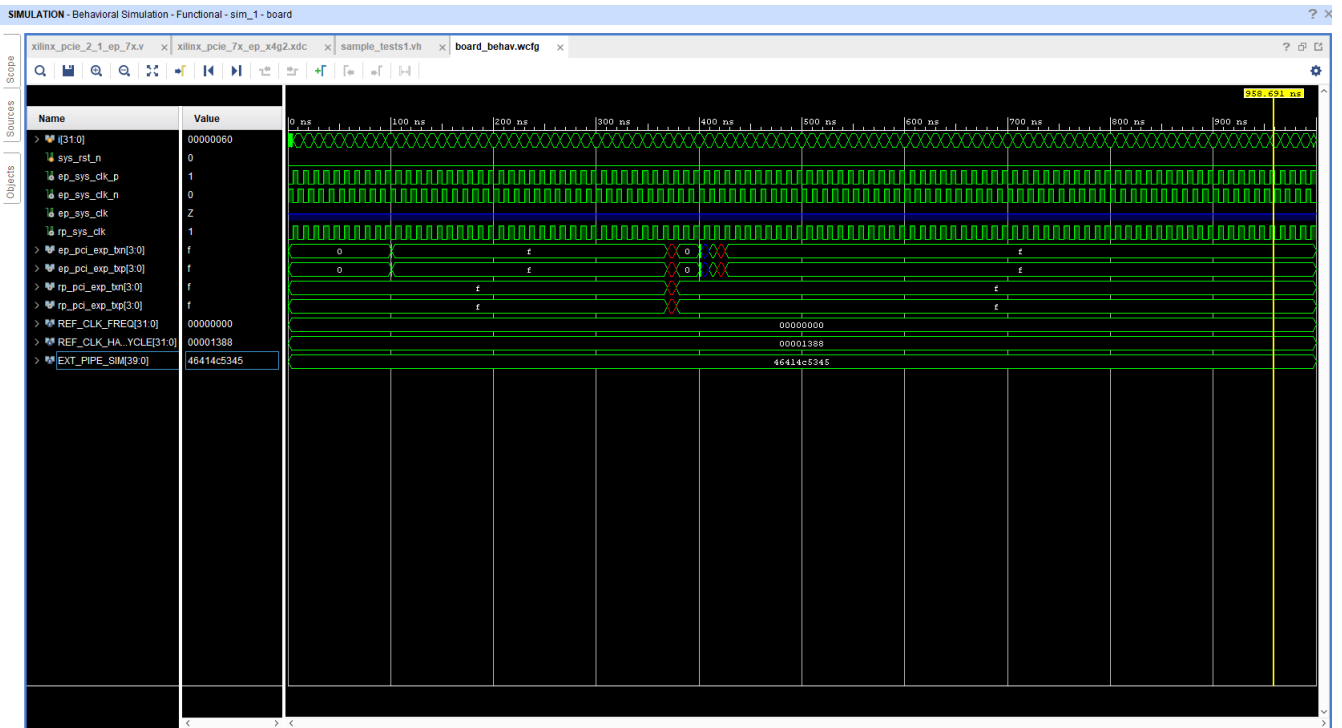


Figure 3.17: Initial PCIe communication

Then, initial tests were conducted by writing strings using the PCIe link, illustrated in Figure 3.18.

```
[
    0] board.RP.rport.pcie_top_i.pcie_7x_i.pcie_bram_top ROWS_RX 1 COLS_RX 4
Running default test (pio_writeReadBack_test0).....
[
    0] : System Reset Asserted abdallah test...
xsim: Time (s): cpu = 00:00:04 ; elapsed = 00:00:09 . Memory (MB): peak = 2068.523 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'board_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:25 ; elapsed = 00:14:06 . Memory (MB): peak = 2068.523 ; gain = 0.000
run 10 us
[
    4995000] : System Reset De-asserted...
run: Time (s): cpu = 00:00:18 ; elapsed = 00:01:03 . Memory (MB): peak = 2068.523 ; gain = 0.000
restart
INFO: [Simtcl 6-17] Simulation restarted
run 6 us
```

Figure 3.18: Writing strings using the PCIe link

After, when the FPGA board is connected to the PCIe slot of the motherboard, the user_ink_up logic signals that the PCIe link between the host PC and the FPGA is operational and ready for data exchange. The FPGA board includes an RGB led, so the blue and green led outputs are connected to the user_ink_up logic and its complement output, respectively. When the blue led is on, it indicates that the PCIe link is ready to exchange data, and when the green led is on, it means that the link is not ready, or the host PC

and FPGA are establishing communication, or there are transmission errors. Thus, the user can observe the status of the PCIe link by monitoring the LED outputs. Additionally, a counter is added to verify the functionality of the PCIe clock, and its output is assigned to the red led. This allows for observation of the RGB led blinking, indicating that the board is detected by the host PC. This was added to the design by declaring the ‘counter’ variable as register and assigning the ‘user_lnk_up’, ‘compliment of user_lnk_up’ and ‘counter’ output to the RGB led. Then, this counter implementation is added after the initialization of I/O BUFFERS. Figure 3.19 presents the simulation done after adding the led indicators and the counter, but as the green led is on, it indicates that the link is not ready.

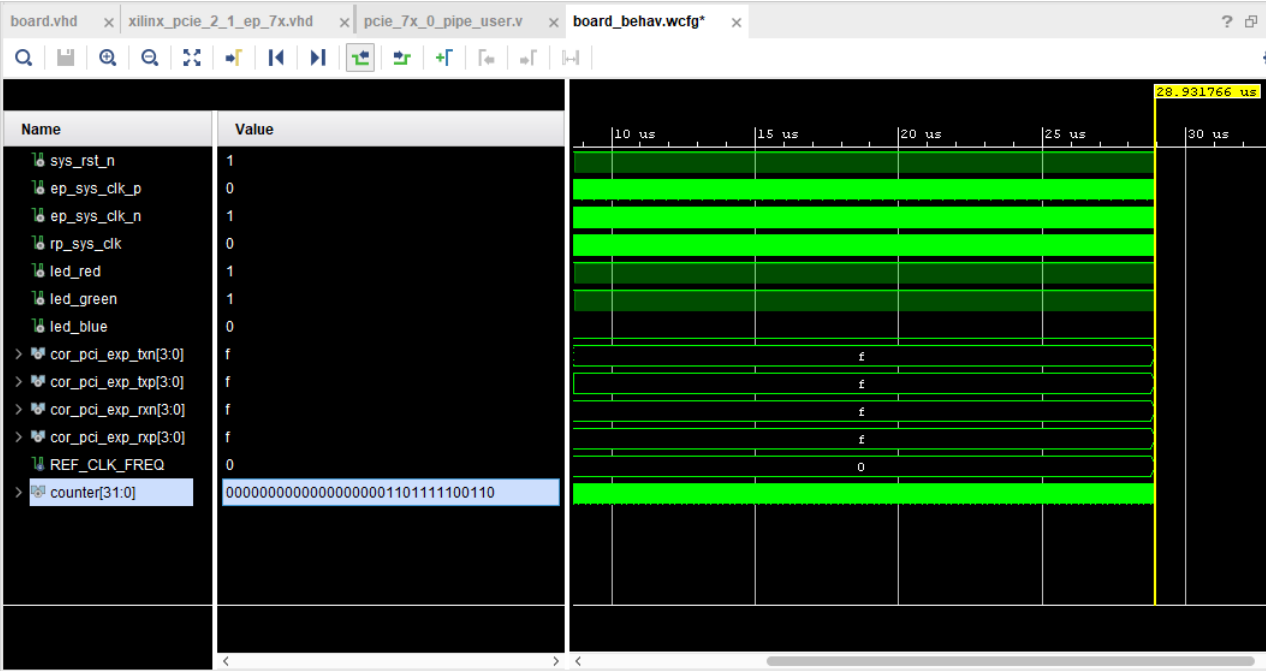


Figure 3.19: Simulation done after adding the led indicators and the counter - link is not ready

Furthermore, Figure 3.20 presents the part of the simulation that indicates that data is being sent and received shown through the txn, txp, rxn, and rxp, as well as having the blue led on and the red led blinking. The simulation of the data transfer using PCIe consumed around 1 W of power,

which is relatively acceptable and shows the possibility of including the FPGA in the future in the modular design to provide more computing power.

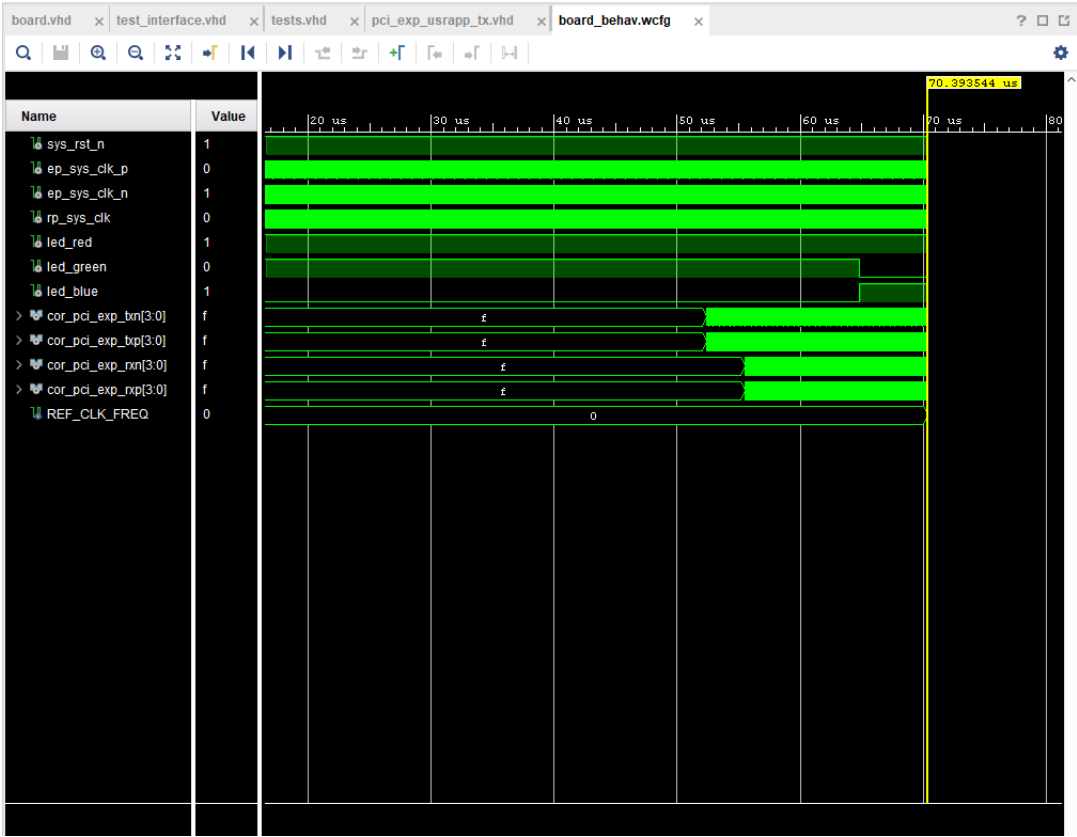


Figure 3.20: Data is being sent and received using PCIe

3.1.6.8 Test Application

The device’s potential to perform future tasks may vary, but it should have the ability to generate serial protocols at acceptable speeds. To verify this ability, the user can input data into a register on the platform system, and then specify the number of bytes to be clocked out and trigger an event through a control register. The FPGA then uses a separate clock signal to transmit the data bit by bit at the specified rate. The application reads the transmitted data on a separate pin and verifies that it matches the input data. This process is repeated to assess the maximum achievable throughput on the GPIO side.

3.1.6.9 Platform Application

When a PCIe device is enumerated, it is assigned a base address that includes all the addressable memory within the FPGA. For example, if the BAR is set to address a specific block at 0x00001000, and the assigned base address is 0x60200000, then the resulting memory location would be 0x60201000. This represents the physical memory of the platform device, which is limited by the amount of RAM available. The physical memory can be accessed directly through the /dev/mem location in Linux or by using RW-Everything software in Windows. Writing to a specific register can trigger an event in the FPGA, and the device can respond accordingly. However, the device's functionality may vary depending on the application. In the current context, RW-Everything was used to program the FPGA flash, as illustrated in Figure 3.21.

```
264 :          DATA_STORE(12) := X"0C";
265 :          DATA_STORE(13) := X"0D";
266 :          DATA_STORE(14) := X"0E";
267 :          DATA_STORE(15) := X"0F";
268 :          -- end usr
269 :
270 :          PROC_TX_MEMORY_WRITE_32 (
271 :              X"02", "000", "0000000100", BAR(i) (31 downto 0), X"0", X"F", '0',
272 :              trn_td_c, trn_tsof_n, trn_teof_n, trn_trem_n_c, trn_tsrc_rdy_n, trn_terrfwd_n,
273 :              trn_lnk_up_n, trn_tdst_rdy_n, trn_clk);
274 :
275 :          P_READ
276 :
277 :          PROC_TX
278 :          X"03"
279 :          trn_t
280 :
281 :          PROC_WA
282 :
283 :
284 :          -- if (P_READ_DATA = X"03020100") then
285 :
286 :              writeNowToScreen(String'("Test PASSED. Completion Data = 0x01020304"));
287 :
288 :          else
289 :
290 :              success := false;
291 :              writeNowToScreen(String'("Test FAILED. Completion Data = 0x01020304"));
```

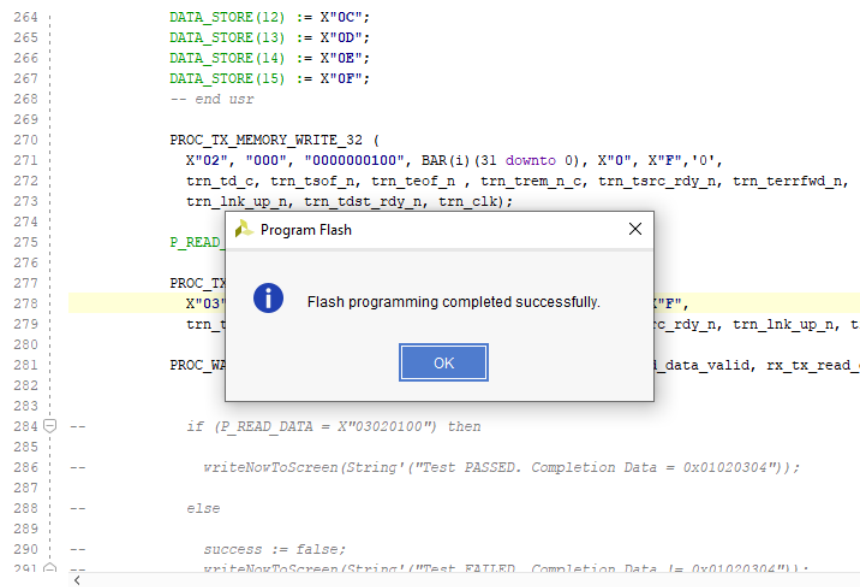


Figure 3.21: Flash programming is successful

Next, the FPGA was inserted in the PCIe slot of the host system's motherboard. After, the host was powered up to a soft restart again before booting into Windows. Once this was done, RW-Everything software was opened and the Xilinx PCIe device was selected from the PCI devices list.

Then, as mentioned above, the BAR Address from the addresses section was located and data was written to one of the address memory locations before proceeding with doing high data transfer using the FPGA.

3.1.7 Bridge Performance

The PCIe communication was tested and verified using two methods, the first was through the simulation that was done using Vivado Design Suite, while the second was done by connecting the FPGA to the PC through the PCIe enabled M.2 slot, as illustrated in Figure 3.22.



Figure 3.22: The FPGA connected to the PC through the PCIe enabled M.2 slot

The PCIe Bridge was recognized by the platform as a PCIe 5GT/s x2 device, and behaved as expected. Due to the 8/10 bit encoding used by PCIe, the maximum transfer rate achieved was 8 Gb/s. During testing, the Artix-7 temperature was continuously monitored using the Vivado IDE. Usually, the temperature remains consistently below 75°C; however, as a fan along with a heat sink was added on the FPGA, the temperature remained below 37°C.

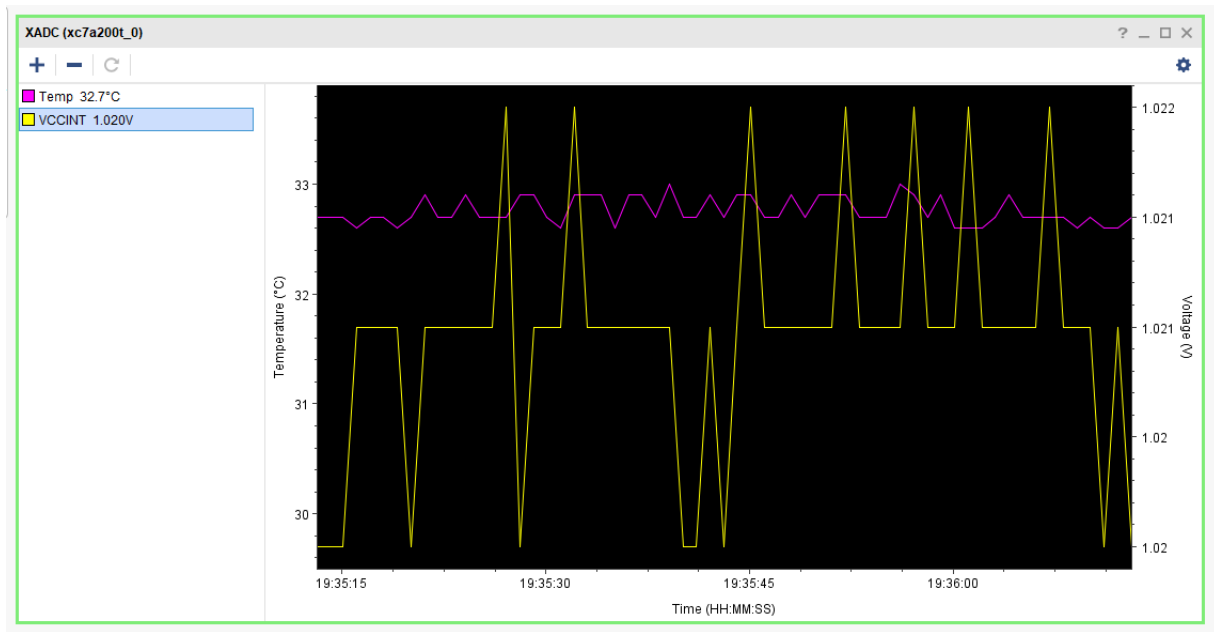


Figure 3.23: Temperature and Voltage Values of the FPGA

3.2 Functional Tests

The full functional test was divided to several tests: Functional test of the PC/104 computing layer of the CDHS architecture (representing layer 3), functional test of layers 1 and 2 mounted on layer 3, and the fully integrated system test (all four layers).

3.2.1 Functional Test of the PC/104 Computing Layer of the CDHS Architecture

To perform the functional test of our PC/104 computing layer of the CDHS architecture, the ST-Link was used to upload the code along with the STM32 CubeIDE. The code contained functions that aim to test each of the main components in this layer, such components include the FRAM, SDRAM, SD Card, Flash Memory, Temperature Sensor, Current Sensor, RTC, and CAN transceiver. After several iterations of this layer, the full functional test was run successfully with all the components satisfying the expected outputs, thus being a fully functional PC/104 computing unit that can be flown in CubeSat missions. Figures 3.24, 3.25, and 3.26 present the results of the testing.

```
COM10 - Tera Term VT
File Edit Setup Control Window Help
a
SD Card Test
-SD Card mounted
-File opened
  Writing...
  Reading...
-Unmounted
  Data sent = Data received
- SD Card Test successful -
+Enter your command: b
  Writing to SD Card...
-Bytes written = 16
+Enter your command: c
  Reading from SD Card...
-Bytes read = 16
  Text: Testing SD Card!
+Enter your command: d
Temp sensor Test
-Temperature = 19.0
+Enter your command: e
Internal Temp Sensor Test
-Internal Temp = 32.27
+Enter your command: f
Current Sensor Test
-Calibrated
-Configured
-Current = -5 nA
+Enter your command: g
SPI-FRAM Test
  Writing...
  Reading...
  Data sent = Data received
- SPI-FRAM Test successful -
```

Figure 3.24: PC/104 computing unit full functional test results - part 1

```
COM10 - Tera Term VT
File Edit Setup Control Window Help

+Enter your command: h
Writing to FRAM...

+Enter your command: i
Reading from FRAM...
-[0] = 0xa
-[1] = 0xb
-[2] = 0xc

+Enter your command: j
Internal RTC Test
-Date, time, & alarm set.

+Enter your command: k
-Date&Time: 05-04-2022 10:00:01

+Enter your command: l
External RTC Test
-Date & time set.

+Enter your command: m
-Ex Time: Wed 12/12/2022 10:00:01

+Enter your command: n
SDRAM Test
Writing...
Reading...

Data sent = Data received
- SDRAM Test successful -

+Enter your command: o
Writing to SDRAM...

+Enter your command: p
Reading from SDRAM...
-Bytes read = 331

+Enter your command: q
CAN bus Test
-Sending data through CAN bus...
```

Figure 3.25: PC/104 computing unit full functional test results - part 2

```
COM10 - Tera Term VT
File Edit Setup Control Window Help
Reading from SDRAM...
-Bytes read = a31
+Enter your command: q
CAN bus Test
-Sending data through CAN bus...
+Enter your command: r
-Receiving data through CAN bus...
-Data Received: 0x4 0x64
+Enter your command: s
Flash memory Test
-Device ID = 01 02 20 4d 00 80
Writing...
Reading...
-Bytes read = 0a 0b 01 02 ff
Data sent = Data received
- Flash Memory Test successful -
+Enter your command: t
Writing to the Flash memory...
+Enter your command: u
Reading from the Flash memory...
-Bytes read = ff ff ff ff
+Enter your command: t
Writing to the Flash memory...
+Enter your command: u
Reading from the Flash memory...
-Bytes read = 01 02 0f 00 ff
+Enter your command: v
Invalid selection.
+Enter your command:
```

Figure 3.26: PC/104 computing unit full functional test results - part 3

Moreover, as CubeSat missions are very sensitive to the available power, having a PC/104 computing layer with low power consumption is important. As shown in Figure 3.27, our PC/104 computing layer runs on 3.3V and consumes about 80 mA in nominal operations mode, that includes the RTC, temperature sensor, and current sensor being turned on.

```

Tera Term - [disconnected] VT
File Edit Setup Control Window Help

External RTC Test
- Control reg = 23
- Ex Time: Sat 12/12/2022 10:35:40

SDRAM (FMC) Test
- Writing done
- Bytes read = 38591

Current Sensor Test
- Calibrated
- Configured
- Current = 87 nA

Flash memory Test
- Device ID = 01 02 20 4d 00 80
- Bytes read = 01 02 03 10 ff

Test* Temp = 24.50 In Temp = 39.34 Time = 10:00:33 Ex Time: 10:35:43 Current = 82 nA
Test* Temp = 24.50 In Temp = 38.2 Time = 10:00:35 Ex Time: 10:35:45 Current = 77 nA
Test* Temp = 24.50 In Temp = 38.44 Time = 10:00:37 Ex Time: 10:35:47 Current = 80 nA
Test* Temp = 24.50 In Temp = 38.15 Time = 10:00:39 Ex Time: 10:35:49 Current = 77 nA
Test* Temp = 24.50 In Temp = 39.20 Time = 10:00:41 Ex Time: 10:35:51 Current = 81 nA
Test* Temp = 24.50 In Temp = 38.85 Time = 10:00:43 Ex Time: 10:35:53 Current = 79 nA
Test* Temp = 24.50 In Temp = 40.88 Time = 10:00:45 Ex Time: 10:35:55 Current = 84 nA
Test* Temp = 24.50 In Temp = 38.79 Time = 10:00:47 Ex Time: 10:35:57 Current = 81 nA
Test* Temp = 24.50 In Temp = 39.15 Time = 10:00:49 Ex Time: 10:35:59 Current = 81 nA
Test* Temp = 24.50 In Temp = 38.79 Time = 10:00:51 Ex Time: 10:36:01 Current = 80 nA
Test* Temp = 24.50 In Temp = 39.19 Time = 10:00:53 Ex Time: 10:36:03 Current = 83 nA
Test* Temp = 24.50 In Temp = 38.69 Time = 10:00:55 Ex Time: 10:36:05 Current = 77 nA
Test* Temp = 24.50 In Temp = 39.4 Time = 10:00:57 Ex Time: 10:36:07 Current = 83 nA
Test* Temp = 24.50 In Temp = 38.57 Time = 10:00:59 Ex Time: 10:36:09 Current = 80 nA
Test* Temp = 24.50 In Temp = 39.15 Time = 10:01:01 Ex Time: 10:36:11 Current = 82 nA
Test* Temp = 24.50 In Temp = 38.70 Time = 10:01:03 Ex Time: 10:36:13 Current = 81 nA
Test* Temp = 24.50 In Temp = 39.15 Time = 10:01:05 Ex Time: 10:36:15 Current = 82 nA
Test* Temp = 24.50 In Temp = 38.80 Time = 10:01:06 Ex Time: 10:36:17 Current = 79 nA
Test*
Test* Temp = 24.50 In Temp = 38.76 Time = 10:01:10 Ex Time: 10:36:21 Current = 80 nA
Test* Temp = 24.50 In Temp = 39.35 Time = 10:01:12 Ex Time: 10:36:23 Current = 84 nA
Test* Temp = 24.50 In Temp = 38.76 Time = 10:01:14 Ex Time: 10:36:25 Current = 78 nA
Test* Temp = 24.50 In Temp = 39.15 Time = 10:01:16 Ex Time: 10:36:28 Current = 82 nA
Test* Temp = 24.50 In Temp = 38.80 Time = 10:01:18 Ex Time: 10:36:30 Current = 81 nA
Test* Temp = 24.50 In Temp = 39.28 Time = 10:01:20 Ex Time: 10:36:32 Current = 83 nA
Test* Temp = 24.50 In Temp = 38.80 Time = 10:01:22 Ex Time: 10:36:34 Current = 80 nA
Test* Temp = 24.50 In Temp = 39.34 Time = 10:01:24 Ex Time: 10:36:36 Current = 81 nA
Test* Temp = 24.50 In Temp = 38.67 Time = 10:01:26 Ex Time: 10:36:38 Current = 79 nA
Test* Temp = 24.50 In Temp = 39.34 Time = 10:01:28 Ex Time: 10:36:40 Current = 81 nA
Test* Temp = 24.50 In Temp = 38.80 Time = 10:01:30 Ex Time: 10:36:42 Current = 81 nA

```

Figure 3.27: Power Consumption of the PC/104 computing unit

3.2.2 Functional Test of Layers 1 and 2 Mounted on Layer 3

To test the functionality of mounting layers 1 & 2 on layer 3, the setup shown in Figure 3.28 was used. The setup included a DIGI XBee 3 module (layer 1), an interface board equipped with a Bergstak connector (layer 2), and a PC/104-based computing layer (layer 3). The data was successfully transmitted and received at both, the system's side and the PC's side.

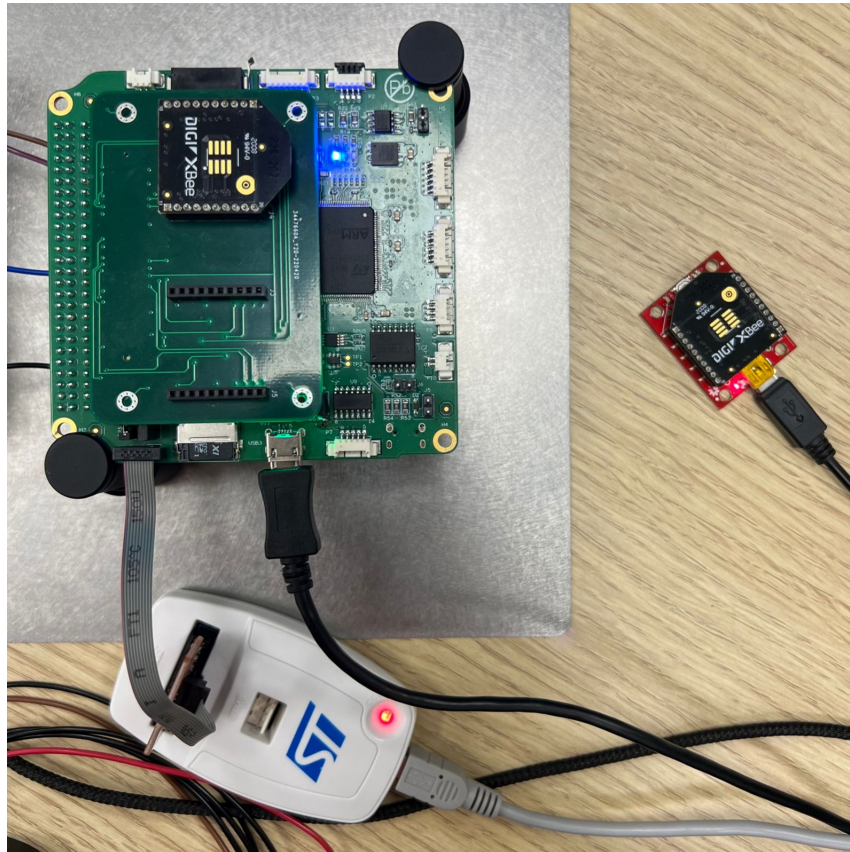


Figure 3.28: Zigbee Communication Test Setup

3.2.3 Functional Test of the Fully Integrated System

The full system test, that included interfacing between an MCU and a cPCI based backplane, includes the setup shown in Figure 3.29, where on the leftmost side, the PC/104-based computing unit is used and it communicates through Ethernet, which gets converted to PCIe using a PCIe-Ethernet bridge. Next, a cPCI-based interface board is used to connect the left side with the cPCI-backplane.

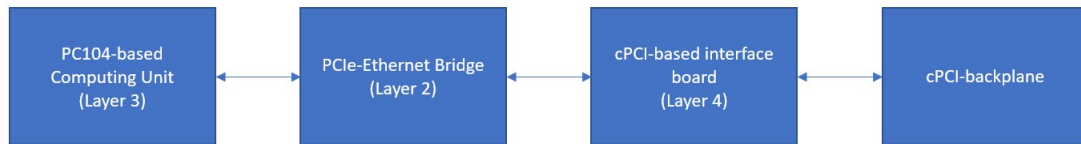


Figure 3.29: Setup used for the cPCI interfacing

Moreover, to practically test the system, a Nucleo-F746ZG board is used to represent the PC/104-based computing unit, a LAN7430 evaluation board is used, and a PCIe x1 - PCIe M.2 adapter is used to connect the left side with the right side that includes an Artix-7 FPGA, as illustrated in Figure 3.30.

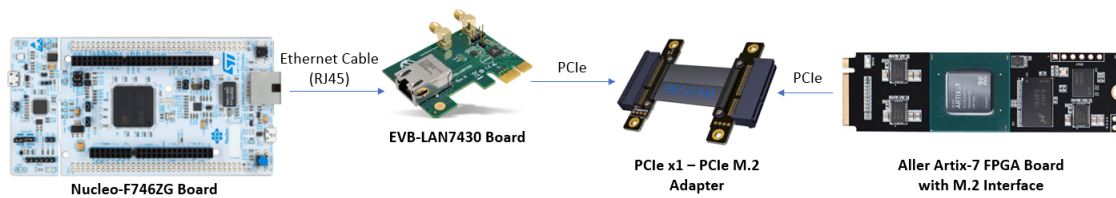


Figure 3.30: Actual setup used for the cPCI interfacing

Due to unforeseen delays in the manufacturing of the custom-made PCIe-M.2 adapter PCB, an alternative solution was sought, potentially involving any PCIe compatible device. Consequently, a personal computer (PC) was chosen as a substitute, representing the cPCI backplane. The test setup utilized for this purpose is presented in Figure 3.31, where the Nucleo-F746ZG board was connected to a PCIe-Ethernet adapter known as the "LAN7430 Evaluation Board", which was then linked to the PC's PCIe connector.



Figure 3.31: The microcontroller and cPCI backplane (represented by a PC) data transfer test setup

To begin the test, we chose an image to send using high data rate transmission. The transmitted image is shown in Figure 3.32.

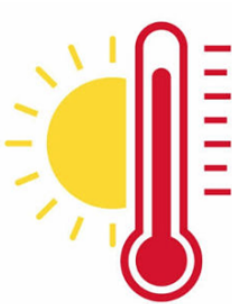


Figure 3.32: The image sent during the test

IDLE, which is an integrated development environment for Python, was used to run a Python code that converts the image from binary to hex format. The converted data is saved in a C header source file, as presented in figure 3.33.

Name	Date modified	Type	Size
binary_to_image_V2	16/05/2023 12:33 AM	Python Source File	1 KB
image_converter_V3	16/05/2023 12:27 AM	Python Source File	3 KB
my_hex_image.bin	16/05/2023 12:51 AM	BIN File	98 KB
my_hex_image	16/05/2023 12:51 AM	C Header Source F...	80 KB
my_hex_image	16/05/2023 12:51 AM	Text Document	25 KB
server_V3	16/05/2023 12:32 AM	Python Source File	1 KB
temp	25/04/2023 9:22 PM	PNG File	48 KB
updated_image	16/05/2023 12:51 AM	JPG File	2 KB

Figure 3.33: The C header file

This file is then imported to STM32CubeIDE, which includes the Ethernet transmission code. To receive the data using the PC, the IP address in server_V3.py file is set according to the IP address of our system, as shown in Figure 3.34.

```
server_V3.py - C:\Users\PC\OneDrive\Desktop\Python_Image_Processing\Python_Image_Pro...
File Edit Format Run Options Window Help
import socket

# Server IP and Port
IP = '192.168.1.68' # Loopback IP address for localhost
PORT = 23 # Port number for the server

# Create a TCP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((IP, PORT))
server_socket.listen(5)

print(f"Server listening on (IP):(PORT)")

# Accept incoming connections
client_socket, client_address = server_socket.accept()
print(f"Connection from (client_address[0]):(client_address[1])")

# Create or open the file for writing
with open('received_data.txt', 'w') as file:
    while True:
        # Receive data from client
        data = client_socket.recv(16129)
        if not data:
            break
        # Convert binary data to hex string
        hex_data = data.hex()
        # Write hex string to file
        file.write(hex_data)
        print("Data received and written to 'received_data.txt'")

# Close client socket and server socket
client_socket.close()
server_socket.close()
```

Figure 3.34: The changed IP address

Then, the data is received and presented using PuTTY, as shown in Figure 3.35, as well as in IDLE Shell.

```
COM3 - PuTTY
Board IP: 192.168.1.87
Netmask : 255.255.255.0
Gateway : 192.168.1.1
158
Connection established
Now sending a packet
0
Data[0] transmitted successfully
Data received.
The remote host closed the connection.
Now I'm closing the connection.
Board IP: 192.168.1.87
Netmask : 255.255.255.0
Gateway : 192.168.1.1
158
Connection established
Now sending a packet
0
Data[0] transmitted successfully
Data received.
The remote host closed the connection.
Now I'm closing the connection.
```

Figure 3.35: Data received (verified using PuTTY)

Finally, the received data is converted from hex back to binary to reconstruct the image, which matches the image that was initially sent. These results prove the reliability of the proposed architecture and its compliance with the cPCI Serial Space standard.

3.3 Thermal Vacuum Test of the PC/104 Computing Layer of the CDHS architecture

This section describes the thermal vacuum test’s plan, setup, and results that were performed to test the performance and rigidness of the PC/104 computing layer of the CDHS architecture in the thermal ranges and vacuum state that simulates the space environment. This test was done using NSSTC’s Small Thermal Vacuum Chamber (STVAC).

3.3.1 Test Plan

The test was planned to run for two cycles, each having a cold state and a hot state that lasted for one hour, under the vacuum state. The process starts with preparing the cryogenic pump and decreasing the pressure to reaching an ultimate vacuum pressure of less than $1e-5$ mbar. Next, the first cycle starts by increasing the STVAC’s temperature from 24°C (ambient temperature) to 50°C (hot state). The temperature is then maintained at 50°C for an hour before dropping down to -20°C (cold state) with a 1°C/minute rate and staying at the cold state for an hour. Then, the second cycle starts by increasing the temperature from -20°C to 50°C and repeating the process as what was done in cycle 1. The temperature profile of the test is illustrated in Figure 3.36.

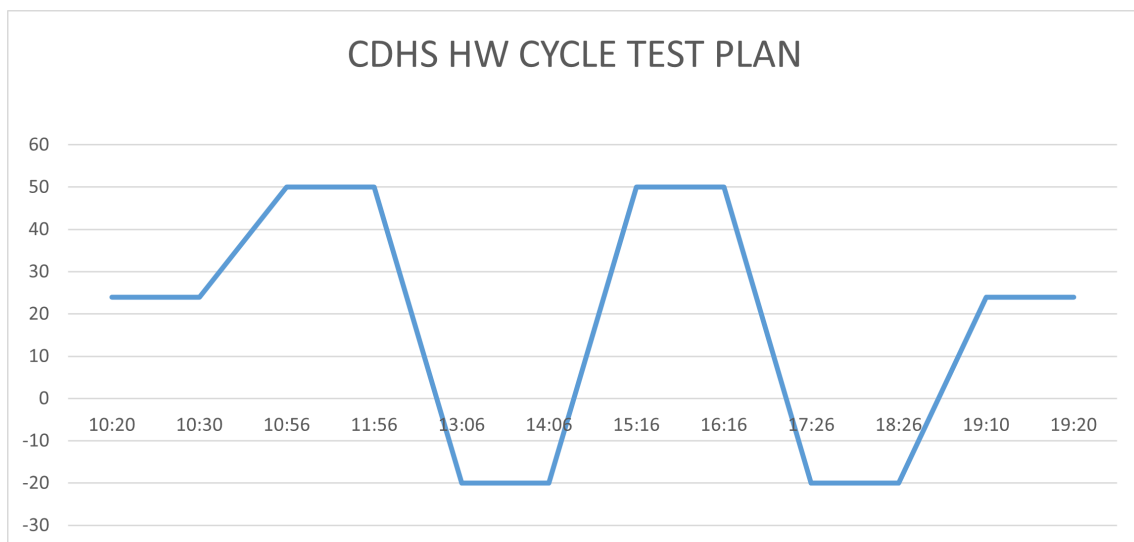


Figure 3.36: Temperature Profile

We have followed a specific procedure for the preparation of the test and the functional testing that ran on the board during the thermal vacuum testing. Due to the lack of any outgassing sensors at the NSSTC facility currently, the success criteria of the test was based on the following pressure measurements of the STVAC:

1. Ramp-up Starting Pressure: $< 1e-5$ mbar
2. After Reaching Bakeout Temp: $< 1e-4$ mbar
3. Test End Pressure: $< 1e-5$ mbar

The test is considered to have ended successfully when the pressure returns to $< 1e-5$ mbar before the completion of the 24 hours dwell time.

3.3.2 Test Setup

The test's setup contains cleaning the STVAC using Isopropyl Alcohol (IPA), placing the board on the STVAC's table using stands made of Teflon, which can withstand extreme temperatures and is an excellent insulator, as illustrated in Figure 3.37, distributing thermocouples around the STVAC and on the board to get temperature measurements at these areas, as illustrated in Figures 3.38, 3.39 and 3.40, and making sure that the STVAC is closed properly to avoid leakage of air into the STVAC during the vacuum preparation.

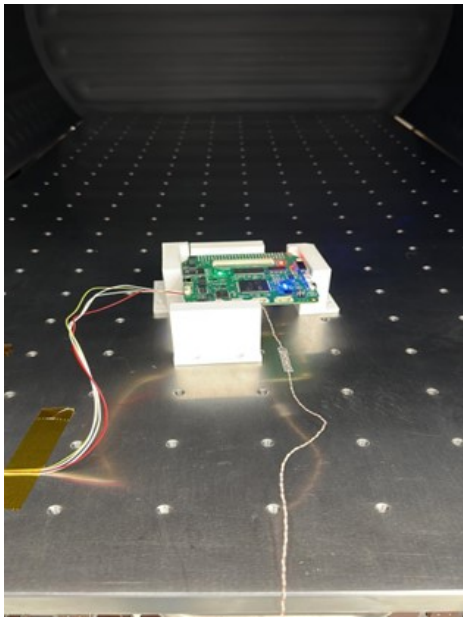


Figure 3.37: Board's setup in the STVAC

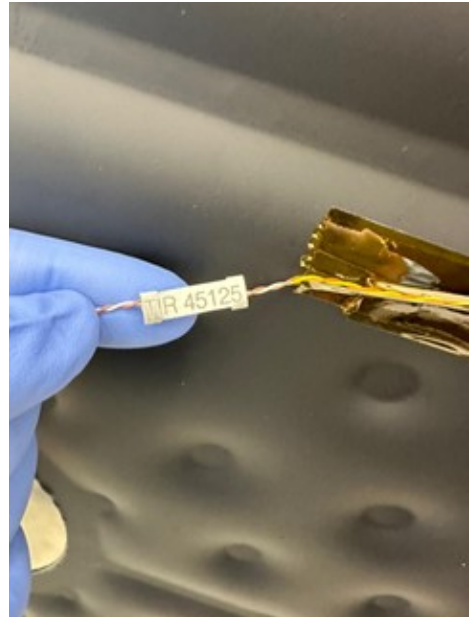


Figure 3.38: Thermocouple on the door

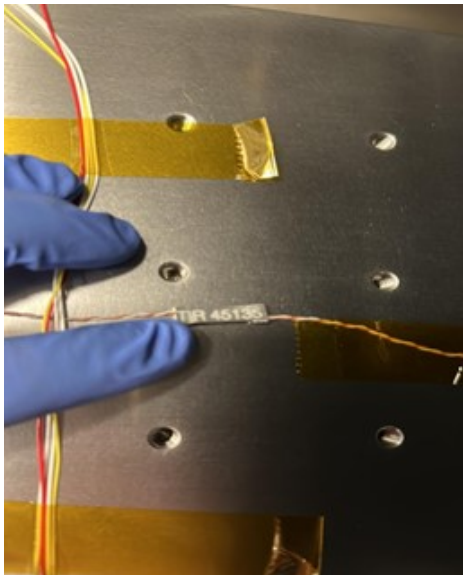


Figure 3.39: Thermocouple on the base plate

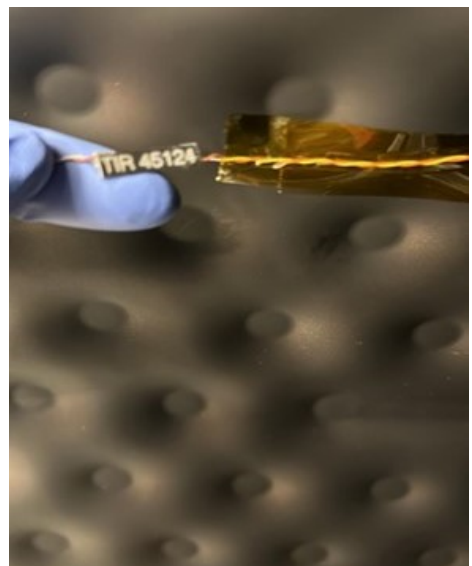


Figure 3.40: Thermocouple on the cylindrical shroud

3.3.3 Test Results

After performing the thermal vacuum test as described in section 3.3.1, we got the following results. The chamber's starting pressure was 1000 mbar, then it decreased to $1e-2$ mbar after primary pumping. Next, the cryogenic pump was on and the pressure decreased to less than $1e-5$ mbar before starting the thermal control phase. During the beginning of the temperature increase phase and when the temperature got stabilized at 50°C , the pressure increased up to slightly less than $1e-4$ mbar due to the acceleration of outgassing, then it decreased with time to reached $1e-6$ mbar during the test up until the end. This is illustrated in Figure 3.41.

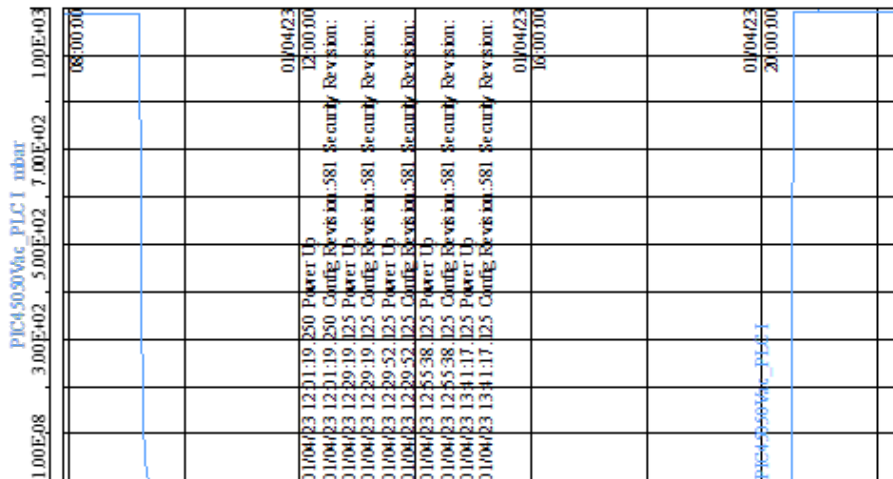


Figure 3.41: Test Pressure Chart

Also, Figure 3.42 illustrates the temperatures of the shroud and base plate, as it shows that the temperatures of both are almost identical throughout the test.

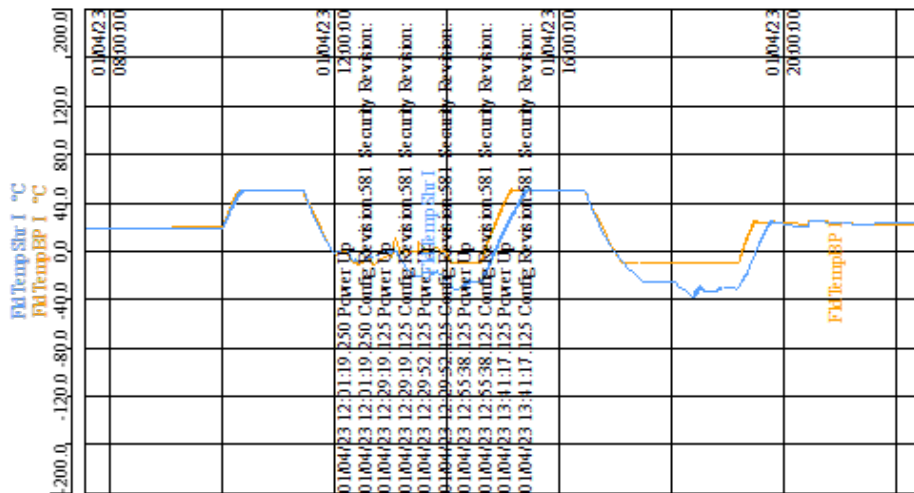


Figure 3.42: Test Shroud and Base Plate Temperature Chart

The temperature values of the thermocouples during the test are presented in Figure 3.43. The graph shows that the temperature started from 24°C, then it increased until it reached 50°C within 26 minutes and it maintained that temperature for 1 hour, then it decreased to around -20°C within 70 minutes and this temperature was also maintained for 1 hour. Then the same procedure got repeated for the second cycle, as the temperature got increased to 50°C within 70 minutes and got maintained at it for an hour, then it got decreased to around -20°C within another 70 minutes and it also got maintained at this temperature for 1 hour. Finally, the temperature was raised to the ambient temperature (24°C).

By subjecting the PC/104 computing layer of the CDHS architecture to these temperature extremes under vacuum conditions, the test aimed to evaluate its thermal stability, assess the effects of thermal cycling, and determine its resilience to the challenging space environment. This layer has successfully demonstrated its ability to operate within the specified temperature range, showcasing its thermal resilience and readiness for future space missions.

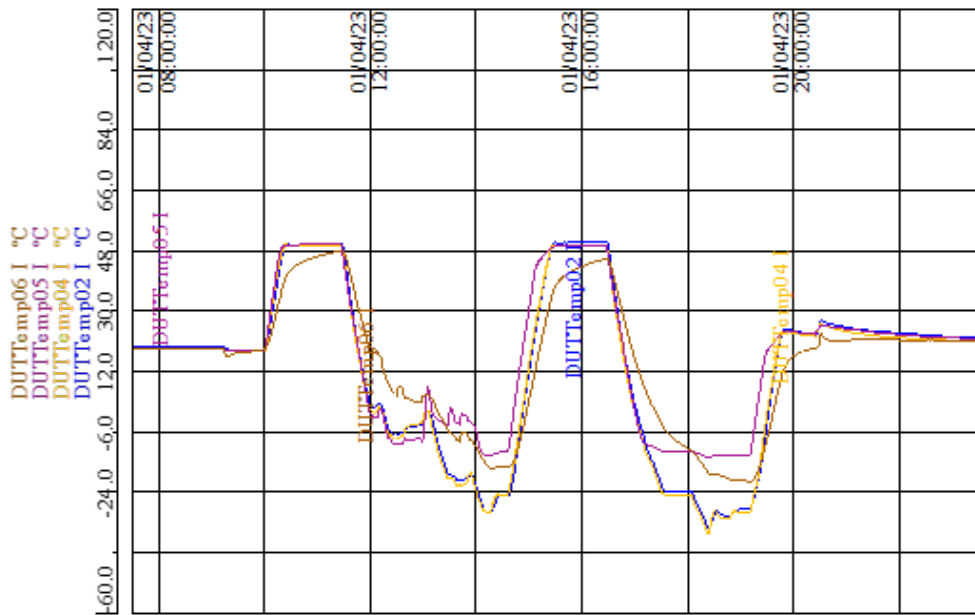


Figure 3.43: Thermocouple Temperature Chart

3.3.4 Use Cases

Our modular architecture has several use cases that are planned for it, which makes it useful for upcoming satellite missions with minimum customization. Each of the following use cases has its own thermal considerations, PCB design guidelines, power consumption, and amount of layers utilized within the modular architecture.

3.3.4.1 Synthetic Aperture Radar (SAR)

One of the use cases of this modular architecture can be utilizing it to create a robust SAR system for hyperspectral imaging that generates high-resolution images by transmitting and receiving radar signals. This system can be used for a variety of applications, such as Earth observation, weather forecasting, and disaster management. The modular architecture can be used to design SAR systems that are scalable, flexible, and reliable. The system can be scaled to meet the needs of different applications, and the modular design makes it easy to upgrade or replace components as needed. The modular architecture also makes the system more reliable, as failures in

individual components can be easily isolated and repaired.

- **Thermal considerations:** SAR systems generate a lot of heat, so it is important to design the system with adequate thermal management. This may include using heat pipes, heat sinks, or other cooling options.
- **PCB design:** The PCB for a SAR system must be designed to handle the high frequencies used by the system. This may require using special materials or techniques.
- **Power consumption:** SAR systems can be very power-hungry, so it is important to design the system to minimize power consumption. This may involve using low-power components or techniques.
- **Layers utilized:** A SAR system will typically utilize all four layers of the modular architecture, if the SAR system is used in satellites that support the use of 3U form factor boards; however, if it doesn't, certain considerations might need to be taken to satisfy the computational requirements depending only on three layers. The layers can be used as the following: Layer 1 accommodates control and interface components, layer 2 hosts specialized signal processing modules, layer 3 houses the OBC for data processing, and layer 4 provides additional computational resources if needed.

3.3.4.2 LoRa-based Payload

LoRa (Long Range) is a low-power, long-range wireless communication technology ideal for IoT (Internet of Things) applications, even in areas with poor signal reception.. In this use case, the modular architecture can be employed to develop a LoRa-Based Payload system with downlink capabilities, enabling communication between the satellite and ground stations. The designed payload can be small, lightweight, and low-power. The system can be designed to meet the specific needs of the

application, and the modular design makes it easy to upgrade or replace components as needed.

- **Thermal considerations:** LoRa systems do not generate a lot of heat, so thermal management is not a major concern.
- **PCB design:** The PCB for a LoRa-based payload can be designed using standard techniques.
- **Power consumption:** LoRa systems are very power-efficient, so power consumption is not a major concern.
- **Layers utilized:** A LoRa-based payload will typically utilize layers 1, 2, and 3 of the modular architecture. The layers can be used as the following: Layer 1 handles control and management, Layer 2 integrates the LoRa transceiver module, Layer 3 incorporates the LoRa-specific OBC. However, layer 4 can be utilized to provide additional functionality or expansion options, if the LoRa system is used in satellites that support the use of 3U form factor boards.

3.3.4.3 On-Board Computer (OBC) for CubeSats

An OBC (On-Board Computer) is a crucial subsystem responsible for controlling and managing the satellite's overall operations. The modular architecture can be employed to develop an OBC specifically designed for CubeSats, taking into account the unique requirements and constraints of these miniature spacecraft. The OBC can be small, lightweight, and low-power. The system can be designed to meet the specific needs of the CubeSat, and the modular design makes it easy to upgrade or replace components as needed.

- **Thermal considerations:** CubeSats are typically small and have limited thermal management capabilities, so it is important to design the OBC to minimize heat generation. This may involve using low-power components

or thermal management techniques.

- **PCB design:** The PCB for a CubeSat OBC can be designed using standard techniques, taking into account the PC/104 form factor which is typically 90 x 96 mm.
- **Power consumption:** CubeSats have very limited power budgets, so it is important to design the OBC to minimize power consumption. This may involve using low-power components or techniques.
- **Layers utilized:** A CubeSat OBC will typically utilize layer 3 of the modular architecture, as it was initially designed to be used as an On-Board Computer (OBC) for CubeSats, passing the qualification testing. However, further layers, especially layers 1 and 2 can be utilized to add further functionality or computing power to the CubeSat OBC.

3.3.4.4 Computer for Microsatellites

A microsatellite is a small satellite that is typically between 10 and 100 kilograms in mass. Microsatellites are used for a variety of applications, such as Earth observation, weather forecasting, and communications. The computer system in a microsatellite is responsible for handling various tasks, such as data processing, storage, and communication. The modular architecture can be leveraged to develop a computer system specifically tailored for the requirements of microsatellites, being powerful, reliable, and easy to maintain, as well as giving it the ability to easily upgrade or replace components as needed. Specifically, this microsatellite computer can be used in the upcoming 813-Sat mission that is hosted by the National Space Science and Technology Center (NSSTC).

- **Thermal considerations:** Microsatellites are typically larger than CubeSats and have more thermal management capabilities, so it is not as critical to minimize heat generation. However, it is still important to

design the system with adequate thermal management.

- **PCB design:** The PCB for a microsatellite computer can be designed using standard techniques, based on the backplane used, whether it is cPCI Serial Space based or SpaceVPX. Typically, both standards follow either a 3U form factor (having a form factor of 100 x 160 mm) or a 6U (having a form factor of 230 x 160 mm).
- **Power consumption:** Microsatellites have larger power budgets than CubeSats, so it is not as critical to minimize power consumption. However, it is still important to design the system to minimize power consumption as much as possible.
- **Layers utilized:** A microsatellite computer will typically utilize layers 3 and 4 of the modular architecture. Layer 3 accommodates the foundational computing elements, and layer 4 allows for vertical expansion and specialized computing tasks, including microsatellite-specific features. Additionally, layers 1 and 2 can be also used to add features and computing capabilities to this microsatellite computer.

3.3.4.5 High Performance Computing (HPC)

High Performance Computing systems are designed to handle complex computational tasks and perform parallel processing to achieve significant computational power. By incorporating the modular architecture, an HPC system can be enhanced with additional computing capabilities using GPUs (Graphics Processing Units) and FPGAs (Field-Programmable Gate Arrays). This HPC system can be used for a variety of applications, such as scientific computing, machine learning, and artificial intelligence.

- **Thermal considerations:** High-performance computing systems can generate a lot of heat, so it is important to design the system with

adequate thermal management. This may include using active cooling.

- **PCB design:** The PCB for a high-performance computing system must be designed to handle the high frequencies used by the system. This may require using special materials or techniques.
- **Power consumption:** High-performance computing systems can be very power-hungry, so it is important to design the system to minimize power consumption by using low-power components or techniques.
- **Layers utilized:** A high-performance computing system will typically utilize all four layers of the modular architecture, especially if it is integrated in microsatellites or larger form satellites that support 3U or 6U form factors. Layers 1 and 2 integrate GPUs and FPGAs for parallel processing, Layer 3 includes the primary computational components, and Layer 4 supports further expansion with additional GPUs or FPGAs. It is important to mention that the modular architecture can host several layers 1 and 2 boards, which can be helpful, specifically in this use case.

In summary, the modular architecture presents a pivotal advancement for satellite systems, offering unparalleled adaptability, scalability, and enhanced performance capabilities across diverse applications. By effectively addressing critical factors such as thermal considerations, PCB design, power consumption, and the strategic utilization of multiple layers within the architecture, it serves as a robust foundation for various use cases. While the aforementioned examples highlight the versatility of this modular architecture, it is important to note that its application extends beyond these instances, as it can be tailored to meet the specific requirements of different missions and satellite endeavors. To provide a concise summary of the comparative analysis among the surveyed modular architectures, as presented in Table 1.3, the modular system in this thesis exhibits the following characteristics: standardized interfaces and sizes, high level of modularity (up to four layers), scalability from Nano to Micro Satellites with potential for further expansion, diverse interconnect options including UART, I2C, CAN, SPI, Ethernet, and PCIe, an expanding and established ecosystem, suitability for small satellites, accessible and well-documented resources for student and researcher utilization, fault tolerance, and plug-and-play capabilities. Also, the following scatter plot, presented in Figure 3.44 can help further clarify how this architecture compares to the rest in terms of system performance (including computational power) and modularity, according to what was mentioned in the literature review about each of the other architectures and this thesis in specific about the proposed architecture.

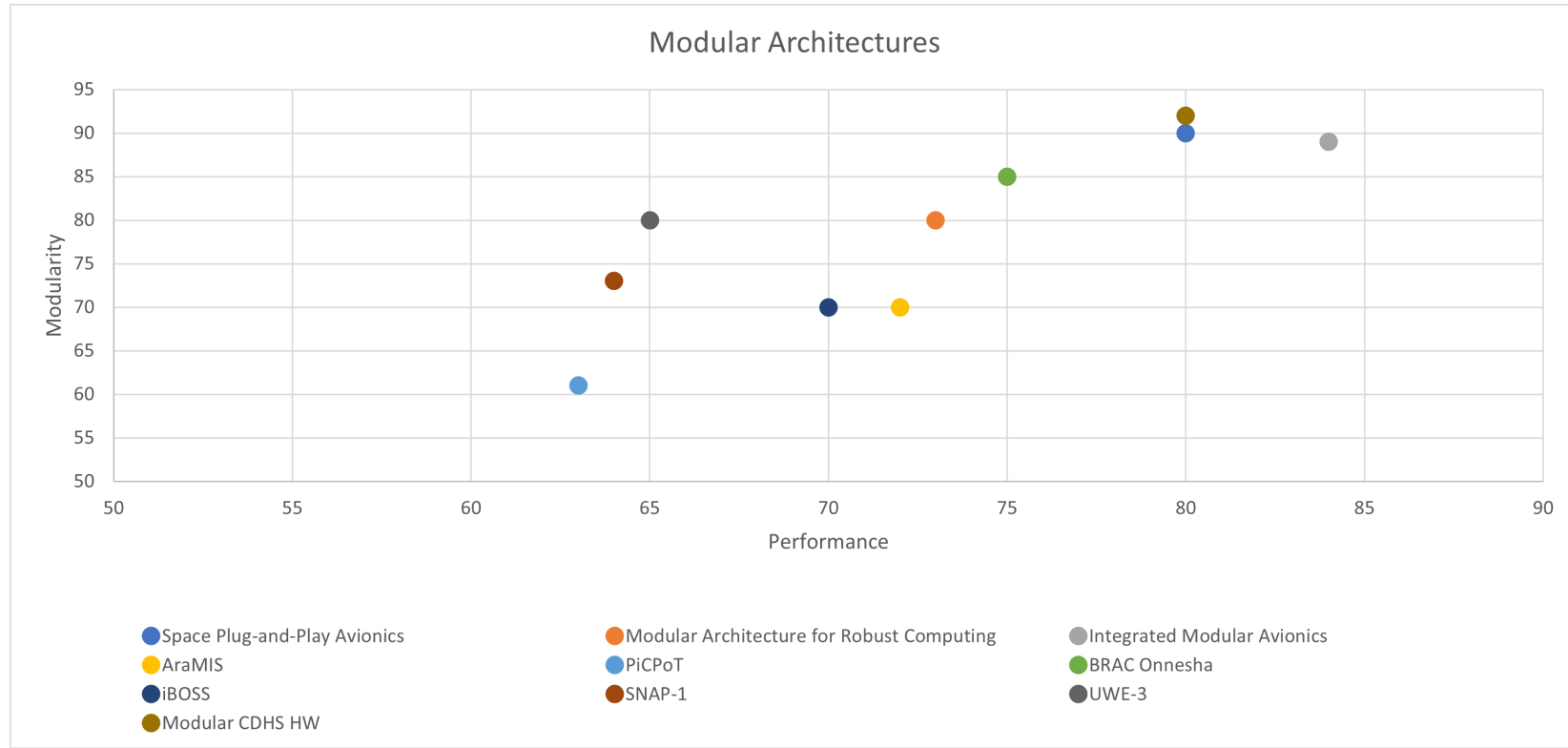


Figure 3.44: Scatter plot of the modular architectures: Performance vs Modularity

Chapter 4: Conclusion

In this thesis, we have conducted a comprehensive review of modular standards applicable to satellite systems, exploring various missions that have adopted such standards, along with other relevant considerations. We have delved into different intrasatellite communication interfaces that facilitate the implementation of modular approaches in satellite development. Additionally, we have compared industry-standard connectors, presenting satellite developers with a range of options to choose from. Our CDHS architecture consists of four layers of modularity, offering expandability and scalability to accommodate larger satellite missions, particularly microsattellites. During our exploration, we evaluated two prominent CDHS standards for larger space missions: cPCI Serial Space and SpaceVPX. While both standards provide modular design and high data rate interfacing capabilities, cPCI Serial Space stands out due to its extensive adoption of the PCI Express (PCIe) standard. Leveraging the large ecosystem of off-the-shelf components and tools available for cPCI Serial Space facilitates its practical implementation. Moreover, cPCI Serial Space supports both 3U and 6U form factors, providing greater flexibility in system design. In contrast, SpaceWire, a specialized space communication standard, offers lower data rates and presents higher complexity in its implementation. We have also highlighted the significance of specific interfaces such as UART, I2C, CAN, and SPI within the CDHS, chosen based on system requirements and the nature of the transmitted data. For instance, UART serves well for serial communication, I2C proves useful for low-speed communication with small data volumes, CAN is suitable for real-time communication, and SPI facilitates high-speed communication with a limited number of devices. One significant challenge we encountered was establishing PCIe communication with a non-PCIe compatible microcontroller present in our PC/104 computing layer of the

modular CDHS architecture. To address this, we explored several potential solutions, ultimately selecting a microcontroller commonly used in CubeSat designs alongside a PCIe-Ethernet bridge to enable PCIe communication. For the purpose of testing PCIe connectivity in this thesis, we simulated the cPCI backplane by employing an FPGA with PCIe compatibility as an end-device. The selection of a suitable FPGA, as well as a flight-proven microcontroller, was carefully considered during this process. Throughout the testing phase, we practically evaluated multiple communication protocols, including UART, I2C, SPI, CAN, Ethernet, and PCIe. Subsequently, we conducted various tests to assess the functionality of the PC/104 computing layer, the interfacing between layers 1 and 2 mounted on layer 3, and the cPCI interfacing layer of the fully integrated CDHS architecture. Additionally, a thermal vacuum testing procedure was performed on the PC/104 computing layer to qualify it for future satellite missions. Based on our findings, the selection of a command and data handling standard and communication interface should be determined by the specific requirements of the space mission and the capabilities of available technologies. Building on the successful implementation of modularity in CDHS hardware architectures, as experimented in this thesis, future research could explore the application of modularity to other subsystems in satellites. This could include investigating the feasibility of using modular designs in power systems, communication systems, and propulsion systems. Additionally, there is potential to explore the scalability of modular designs in satellite systems of varying sizes and complexities. Further research could also explore the benefits of modularity in reducing development costs and improving system reliability. This thesis provides a strong foundation for future work in the field of satellite hardware architecture and paves the way for a more efficient and effective approach to designing and building satellite systems.

References

- [1] N. Anand, G. Joseph, S. S. Oommen, and R. Dhanabal. Design and implementation of a high speed serial peripheral interface. In *2014 International Conference on Advances in Electrical Engineering (ICAEE)*, pages 1–3. IEEE, 2014.
- [2] M. Baldwin and V. Di Lello. New techniques to address layout challenges of high-speed signal routing. *Cadence, PCB West, Tech. Rep*, 2016.
- [3] L. Bielich. In-system eye scan of a pci express link with vivado ip integrator and axi4. In *Application Note XAPP 1198 (v1. 1)*. Xilinx Inc., 2014.
- [4] R. Bittner. Bus mastering pci express in an fpga. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 273–276, 2009.
- [5] R. Botelho and A. Xavier. A unified satellite taxonomy proposal based on mass and size. *Advances in Aerospace Science and Technology*, 04:57–73, 10 2019.
- [6] D. Bueno, C. Conger, A. D. George, I. Troxel, and A. Leko. Rapidio for radar processing in advanced space systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(1):1–38, 2007.
- [7] S. Busch, P. Bangert, S. Dombrovski, and K. Schilling. Uwe-3, in-orbit performance and lessons learned of a modular and flexible satellite bus for future pico-satellite formations. *Acta Astronautica*, 117:73–89, 2015.
- [8] S. Busch and K. Schilling. Robust and efficient obdh core module for the flexible picosatellite bus uwe-3. *IFAC Proceedings Volumes*, 46(19):218–223, 2013.
- [9] S. Busch, K. Schilling, P. Bangert, and F. Reichel. Robust satellite engineering in educational cubesat missions at the example of the uwe-3 project. *IFAC Proceedings Volumes*, 46(19):236–241, 2013.
- [10] G. Capovilla, E. Cestino, L. Reyneri, and G. Romeo. Modular multifunctional composite structure for cubesat applications: Preliminary design and structural analysis. *Aerospace*, 7:17, 02 2020.
- [11] M. J. Chong. *A PCI Express to PCIX Bridge optimized for performance and area*. PhD thesis, Massachusetts Institute of Technology, 2003.
- [12] J. H. Christensen, D. B. Anderson, M. E. Greenman, and B. D. Hansen.

- Scalable network approach for the space plug-and-play architecture. In *2012 IEEE Aerospace Conference*, pages 1–10, 2012.
- [13] B. F. Collins. Development of a space universal modular architecture (sumo). In *2013 IEEE Aerospace Conference*, pages 1–9, 2013.
- [14] B. Cook and P. Walker. Ethernet over spacewire—hardware issues. *Acta Astronautica*, 61:243–249, 06 2007.
- [15] G. Dassano and M. Orefice. The picpot satellite antenna systems. In *2007 IEEE Antennas and Propagation Society International Symposium*, pages 3029–3032, 2007.
- [16] D. Del Corso, C. Passerone, L. Reyneri, C. Sansoe, S. Speretta, and M. Tranchero. Design of a university nano-satellite: the picpot case. *IEEE Transactions on Aerospace and Electronic Systems*, 47(3):1985–2007, 2011.
- [17] C. J. Finley, K. Richards, J. C. Cabanillas, D. Gudea, P. Katz, S. Ray, G. Moretti, and D. Tsairides. How ors – modular space vehicle on the t2e mission. In *25th Annual AIAA/USU Conference on Small Satellites*, 2011.
- [18] J. Flores-Arias, M. Ortiz-Lopez, F. J. Quiles-Latorre, V. Pallarés, and A. Chen. Complete hardware and software bench for the can bus. In *2016 IEEE International Conference on Consumer Electronics (ICCE)*, pages 211–212. IEEE, 2016.
- [19] F. I. for Open Communication Systems (FOKUS). A versatile high performance computer for every application. In *OBC-SA Brochure: https://cdn0.scrvt.com/fokus/aee649f1ecde67ea/425e706edda0/OBC-SA_BroschuereEN_Web_280218_01.pdf*, pages 1–2. Fraunhofer IPMS, 2023.
- [20] T. Fountain, A. McCarthy, F. Peng, et al. Pci express: An overview of pci express, cabled pci express and pxi express. In *10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems*, 2005.
- [21] S. Fuller. *RapidIO: The embedded system interconnect*. John Wiley & Sons, 2004.
- [22] G. Furano and A. Menicucci. *Roadmap for On-Board Processing and Data Handling Systems in Space*, pages 253–281. Springer International Publishing, Cham, 2018.
- [23] W. Gasti, A. Senior, O. Emam, T. Jorden, R. Knowelden, and S. Fowell. Modular architecture for robust computation session: Spacewire onboard

- equipment and software. *Proceedings of the International Spacewire Conference 2007*, 2008.
- [24] M. Haque, J. Arifin, et al. *Eshtablishment of" BRAC Onnesha" nano sattelite ground station and comparative analysis of different types of antenna as a portable low earth orbit sattelite ground station receiving antenna*. PhD thesis, BRAC University, 2017.
- [25] H. J. Herpel, A. Schuettauf, G. Willich, S. Tverdyshev, S. Pletner, F. Schoen, B. Kiewe, C. Fidi, M. Maeke-Kail, and K. Eckstein. Open modular computing platforms in space—learning from other industrial domains. In *2016 IEEE Aerospace Conference*, pages 1–11. IEEE, 2016.
- [26] R. F. Hodson, W. A. Powell, A. H. Lanham, A. D. Geist, P. Collier, D. I. Nakamura, and H. J. Yim. Spacevpx interoperability assessment. Technical report, National Aeronautics and Space Administration (NASA), 2022.
- [27] L. O. Hoeft, J. L. Knighten, and M. Ahmad. Measured surface transfer impedance of multi-pin micro-d subminiature and lfh/sup tm/connector assemblies at frequencies up to 1 ghz. In *1999 IEEE International Symposium on Electromagnetic Compatability. Symposium Record (Cat. No. 99CH36261)*, volume 2, pages 577–582. IEEE, 1999.
- [28] A. Hossain. Brac onnesha launches into space. *The daily sun*, 2017.
- [29] JLCPCB. Multilayer high precision pcb's with impedance control. In *JLCPCB Website: <https://jlcpcb.com/quote/pcbOrderFaq/PCB%20Stackup>*, page 1. JLCPCB, 2022.
- [30] S. Johl, E. Glenn Lightsey, S. M. Horton, and G. R. Anandayavaraj. A reusable command and data handling system for university cubesat missions. In *2014 IEEE Aerospace Conference*, pages 1–13, 2014.
- [31] C. J. Kief, B. Zufelt, S. R. Cannon, J. Lyke, and J. K. Mee. The advent of the pnp cube satellite. In *2012 IEEE Aerospace Conference*, pages 1–5, 2012.
- [32] M. Kortman, S. Ruhl, J. Weise, J. Kreisel, T. Schervan, H. Schmidt, and A. Dafnis. Building block based iboss approach: fully modular systems with standard interface to enhance future satellites. In *66th International Astronautical Congress (Jerusalem)*, pages 1–11, 2015.
- [33] V. Krishnan. Towards an integrated io and clustering solution using pci

- express. In *2007 IEEE International Conference on Cluster Computing*, pages 259–266, 2007.
- [34] U. Kulau, J. Herpel, R. Qedar, P. Rosenthal, J. Krieger, F. Schoen, and I. Masar. Towards modular and scalable on-board computer architecture. *it - Information Technology*, 63, 07 2021.
- [35] T. A. F. R. Laboratory, 2021.
- [36] Z. Lee. Cubesat constellation implementation and management using differential drag. Master’s thesis, Massachusetts Institute of Technology. Department of Aeronautics and Astronautics, 2017.
- [37] R. Lipsett, C. F. Schaefer, and C. Ussery. *VHDL: Hardware description and design*. Springer Science & Business Media, 2012.
- [38] B. Lucia, B. Denby, Z. Manchester, H. Desai, E. Ruppel, and A. Colin. Computational nanosatellite constellations: Opportunities and challenges. *GetMobile: Mobile Comp. and Comm.*, 25(1):16–23, jun 2021.
- [39] M. Martin, D. Fronterhouse, and J. Lyke. The implementation of a plug-and-play satellite bus. In *22nd AIAA / USU Conference on Small Satellites*, 2008.
- [40] M. Martin, J. Summers, and J. Lyke. Afri plug-and-play spacecraft avionics experiment (sae). In *2012 IEEE Aerospace Conference*, pages 1–6, 2012.
- [41] S. S. Math and R. Manjula. Design of amba axi4 protocol for system-on-chip communication. *International Journal of Communication Network and Security*, 1:38–42, 2012.
- [42] S. O. Nambiar, Y. Abhyankar, and S. Chandrababu. Migrating fpga based pci express geni design to gen2. In *2010 International Conference on Computer and Communication Technology (ICCCT)*, pages 617–620. IEEE, 2010.
- [43] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore. Understanding pcie performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 327–341, 2018.
- [44] T. M. Nguyen. Future satellite system architectures and practical design issues: An overview. *Satellite Systems - Design, Modeling, Simulation and Analysis*, 2020.

- [45] J. Norhuzaimin and H. Maimun. The design of high speed uart. In *2005 Asia-Pacific Conference on Applied Electromagnetics*, pages 5–pp. IEEE, 2005.
- [46] V. Olenev, I. Lavrovskaya, I. Korobkov, and Y. Sheynin. Design and simulation of onboard spacewire networks. In *2019 24th Conference of Open Innovations Association (FRUCT)*, pages 291–299, 2019.
- [47] S. Parkes and P. Armbruster. Spacewire: a spacecraft onboard network for real-time communications. In *14th IEEE-NPSS Real Time Conference, 2005.*, pages 6–10. IEEE, 2005.
- [48] S. Parkes, A. F. Florit, A. G. Villafranca, C. McClements, and A. Srivastava. A prototype spacevx lite (vita 78.1) system using spacefibre for data and control planes. In *2017 IEEE Aerospace Conference*, pages 1–9. IEEE, 2017.
- [49] P. Prisaznuk. Integrated modular avionics. *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference*, 1992.
- [50] P. J. Prisaznuk. Arinc 653 role in integrated modular avionics (ima). In *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pages 1–E. IEEE, 2008.
- [51] L. M. Reyneri, C. Sansoè, C. Passerone, S. Speretta, M. Tranchero, M. Borri, and D. D. Corso. Design solutions for modular satellite architectures. In *Aerospace Technologies Advancements*, 2010.
- [52] S. Rietz, B. Schneider, S. Bender, W.-J. Fischer, H. Grätz, and A. Heinig. Multisensor signal processing with can bus interface. *Sensors and Actuators A: Physical*, 62(1):729–733, 1997.
- [53] L. Rota, M. Caselle, S. Chilingaryan, A. Kopmann, and M. Weber. A pcie dma architecture for multi-gigabyte per second data transmission. *IEEE Transactions on Nuclear Science*, 62(3):972–976, 2015.
- [54] T. Schervan, J. Kreisel, K. Schroeder, and D. R. Wingo. New horizons for exploration via flexible concepts based on building blocks using the standardized issi (intelligent space system interface) modular coupling kit by iboss. In *Global Space Exploration Conference (GLEX 2021), St Petersburg, Russian Federation*, pages 14–18, 2021.
- [55] Z. Schoenborn. Board design guidelines for pci express™ architecture. In *PCI-SIG APAC Developers Conference*, page 142, 2004.
- [56] L. Semiconductors. Sub-lvds serial to cmos parallel sensor bridge. Technical Documentation, 2012.

- [57] A. Senior, O. Emam, R. Ward, B. Greene, S. Fowell, and P. Ireland. *MODULAR ARCHITECTURE FOR ROBUST COMPUTING (MARC)*. European Space Agency, 2008.
- [58] S. Shanthipriya and S. Lakshmi. Design and verification of low speed peripheral subsystem supporting protocols like spi, i2c and uart. *ARPJ Journal of Engineering and Applied Sciences*, 12:7386–7391, Dec 2017.
- [59] C. Shim, R. Shinde, and M. Choi. Compatibility enhancement and performance measurement for socket interface with pcie interconnections. *Human-centric Computing and Information Sciences*, 9(1), 2019.
- [60] C. Silva, J. Cristovao, and T. Schoofs. An I/O Building Block for the IMA Space Reference Architecture. In *DASIA 2012 - DATA Systems In Aerospace*, volume 701 of *ESA Special Publication*, page 5, Aug 2012.
- [61] S. Speretta, L. M. Reyneri, C. Sansoe, M. Tranchero, C. Passerone, and D. Del Corso. Modular architecture for satellites. In *58th International Astronautical Congress*, 2007.
- [62] A. Subero and A. Subero. Usart, spi, and i2c: serial communication protocols. *Programming PIC Microcontrollers with XC8*, pages 209–276, 2018.
- [63] Thesheetzweetz. The space industry is on its way to reach \$1 trillion in revenue by 2040, citi says, May 2022.
- [64] N. Tidala. High performance network on chip using axi4 protocol interface on an fpga. In *2018 second international conference on electronics, communication and aerospace technology (ICECA)*, pages 1647–1651. IEEE, 2018.
- [65] M. D. Trottier. *Accurate dynamic response predictions of pnp-sat I*. PhD thesis, Air Force Institute of Technology, 2010.
- [66] C. Underwood, G. Richardson, and J. Savignol. Snap-1: A low cost modular cots-based nano-satellite—design, construction, launch and early operations phase. In *15th AIAA / USU Conference on Small Satellites*, 2001.
- [67] C. I. Underwood, G. Richardson, and J. Savignol. In-orbit results from the snap-1 nanosatellite and its future potential. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1802):199–203, 2003.

- [68] A. Vinogradov, E. Yablokov, and V. Yachnaya. Upgrade of ethernet-spacewire protocol. In *2019 24th Conference of Open Innovations Association (FRUCT)*, pages 486–492, 2019.
- [69] Y. Yang and C. Yan. Pcie device drivers in linux : Design and implementation of a high-speed pci express bridge. In *2022 3rd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, pages 430–437, 2022.
- [70] Q. Young. Modular platform architecture for small satellites: Evaluating applicability and strategic issues. In *19th AIAA / USU Conference on Small Satellites*, 2005.
- [71] M. Zibayiwa and T. Chigwagwa. A Review on The Inter-Processor Communication: I2C, UART, and SPI interfacing techniques, Nov 2021.

List of Publications

1. **Abdullah Alsalmani**, Mohammed Almehezi, and Abdul-Halim Jallad "A Modular Command and Data Handling System Concept for Small Satellites", 36th small satellite conference, UTAH, USA, 2022.
2. **Abdullah Alsalmani**, Teana Alniwairi, Alia Alneyadi, and Abdul-Halim Jallad "A Modular Command and Data Handling System for ALAINSAT-1", *The International Geoscience and Remote Sensing Symposium (IGARSS)*, 2023.
3. Abdul-Halim Jallad, Adriano Camps, Prashanth Marpu, Mai AlMazrouei, Ahmed Ba-Layth, Shamma Aleissae, **Abdullah Alsalmani**, Mohamed Okasha, Adrian Perez-Portero, Amadeu Gongga, Juan Ramos-Castro, Shindi Oktaviani, Edwar Edwar, Yasir Abbas, and Mark Angelo Purio "Overview of ALAINSAT-1 Mission: A Remote Sensing Student Nanosatellite", *The International Geoscience and Remote Sensing Symposium (IGARSS)*, 2023.
4. M. Abduljawad and **A. Alsalmani**, "Towards Creating Exotic Remote Sensing Datasets using Image Generating AI," *2022 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, Ras Al Khaimah, United Arab Emirates, pp. 84-88, 2022, DOI: 10.1109/ICECTA57148.2022.9990245.
5. Anas Al Tarabsheh, Ahmed M. Abughali, **Abdullah M. AlSalmani**, Ahmed J. AlSoufi, and Leen B. Ba'ba' (2023) Programmable photovoltaic submodules for hotspot mitigation, *International Journal of Sustainable Engineering*, 16:1, 1-13, 2023, DOI: 10.1080/19397038.2023.2206418

UAEU

جامعة الإمارات العربية المتحدة
United Arab Emirates University



UAE UNIVERSITY MASTER THESIS NO. 2023: 61

Modular command and data handling systems are essential for satellites as they provide flexibility, adaptability, and scalability. These systems allow for easy integration, component swapping, and upgrades, reducing development time and enhancing performance. Modular systems ensure longevity, efficiency, and optimal resource utilization, making them crucial for the success of satellite missions.

Abdullah Alsalmari received his MSc from the Physics Department, College of Science, UAE University, UAE. He received his BSc in Electrical Engineering from the Electrical and Computer Engineering Department, College of Engineering, Abu Dhabi University, UAE.

www.uaeu.ac.ae

Online publication of thesis:
<https://scholarworks.uaeu.ac.ae/etds/>

UAEU عمادة المكتبات
Libraries Deanship

جامعة الإمارات العربية المتحدة
United Arab Emirates University



قسم الخدمات المكتبية الرقمية - Digital Library Services Section