Politecnico di Torino

Department of Electronics and Telecommunications

Master Thesis in Electronic Engineering

Design and Development of a DSP Accelerators Array in a FPGA-based Clustered Architecture for Space Applications

Supervisor
Prof. Maurizio ZAMBONI

Master Candidate
Antonino CATANESE

Academic Year
2021-2022

"What is ***magic*** to one man is **`engineering`** to another."
— Robert A. Heinlein

Abstract

This thesis focuses on the design of an array of accelerators for Compute-Intensive Telecom Applications within the framework of a project founded by ESA and carried out in *Argotec*, an Italian Aerospace Company developer and manufacturer of new solutions for astronaut comfort and the manufacturing of micro and nano satellites for deep space.

In detail, this project aims at providing a novel architectural approach based on a cluster of Field Programmable Gate Arrays (FPGA), which are programmable devices that provide high flexibility and high performance thanks to their integration and re-configurability. This architectural model could enable the usage of non-space-grade Commercial-of-the-Shelf (COTS) devices in long-life missions in harsh radiation environments by leveraging the resource redundancy and distribution enabled by clustering methods, together with the possibility of different mitigation technique application. Commercial-graded devices are made up of the newest and most performing technology on the market and their use in space environment gives advantages both in terms of performances and costs.

In detail, the present thesis focused on the design of a system with high computing capability by leveraging hardwired primitives present in state-of-the-art FPGAs for Digital Signal Processing (DSP). These DSP units are attractive candidates to balance the programmability typical of dedicated arithmetic logic units and the high performance of dedicated hardware thanks to the possibility of implement a variety of processing modes.
Overall, each DSP unit together with a specific management which will provides signals and controls for data computations create a Programmable Functional Unit (PFU) which is the the key element on which this thesis is based.

The main aim of this thesis is the development of a programmable and scalable accelerator structure capable of executing a range of tasks for telecom applications in space environment. Furthermore, the studies were also carried out with the aim of optimising the architecture presented with a view to maximising performance and minimising the resources used.

In order to achieve these goals, different design solutions were investigated, all of which converged on a replicable architecture inside the target device via a versioning strategy that allowed for the most efficient use of resources and, as a result, the array's appropriate implementation. Furthermore, this design flow resulted in the best trade-off in terms of flexibility, granularity and resource utilisation.
Finally, during the versioning process and on the final design, it was verified that the project's performance requirements were satisfied.

Contents

Listing of figures

Listing of tables

Listing of acronyms

ESA ..... European Space Agency

DSP ..... Digital Signal Processing

FPGA ..... Field Programmable Gate Array

COTS ..... Commercial-of-the-Shelf

FSM ..... Finite State Machine

LDPC ..... Low Density Parity Check

SIMD ..... Single Instruction Multiple Data

PFU ..... Programmable Functional Unit

ALU ..... Arithmetic Logic Unit

MAC ..... Multiply and Accumulate

FIR ..... Finite Impulse Response

SEB ..... Single Event Burnout

SEGR ..... Single Event Gate Rupture

SEL ..... Single Event Latch-Up

SEU ..... Single Event Upset

SET ..... Single Event Transient

TMR ..... Triple Modular Redundancy

SDR ..... Software-Defined Radios

ADC ..... Analog to Digital Converter

DAC ..... Digital to Analog Converter

TT&C ..... Telemetry, Tracking and Command

FFT ..... Fast Fourier Transform

SNR ..... Signal-to-Noise Ratio

AWGN ..... Additive White Gaussian Channel

BER ..... Bit Error Rate

HDL ..... Hardware Description Languages

CLB ..... Configurable Logic Block

LUT ..... Look-Up Table

FF ..... Flip-Flop

DDR ..... Double Data Rate

CU ..... Control Unit

MIG ..... Memory Interface Generator

ISA ..... Instruction Set Architecture

DMA ..... Direct Memory Access

GPU ..... Graphics Processing Unit

PE ..... Processing Element

1 Introduction

This chapter provide a brief introduction concerning the state of the art of current methodologies and devices used in telecommunications for space. It evaluates problems in using commercial devices to replace those currently used for space applications known as rad-hard devices [5].

First, section 1.1 sums up the current and future applications approaches that are aimed at implementing more and more computational power through the use of commercial technology, thus reducing costs and maintaining high reliability.

Then, in section 1.2 the FPGA will be presented, devices used as the core of the project and which historically have a great use in the Telecom and Aerospace environment. Finally, it will be briefly explained what COTS devices are, and what advantages are obtained by introducing them in the space applications.

Subsequently, section 1.3 aims to provide an overview of the current state of the art of satellite communication systems by presenting the techniques used for earth to space transmission.

Finally, section 1.4 the main problems that affects devices in space environment are discussed. In detail, a focus is made on the main problems that are high temperatures and radiation, examining their causes and what they entail. Finally, the section ends by presenting some commonly used mitigation and recovery techniques.

1.1 Trends in Space Communication Systems

One of the main points in space application is the continuous increase in the amount of data collected and sent to Earth. This is a direct consequence both of the desire to know what is in space but, also, of the continuous devices shrinking that allowing to integrate more and more sensors on a single chip capable of receiving and accumulating an ever-increasing amount of data. The accumulation of data itself is not a problem but, it is a problem to send an increasing amount of information from a satellite to Earth.
In this view, a key factor in the evolution of telecom system for space is the digital electronics advent and rapid expansion which will be briefly explained in section 1.3. In fact, it has made possible a performance and memory capacity exponential increase.
Additionally, this has also brought the big advantage of reducing satellites size [6]. Thanks to technologies evolution, in fact, it is possible to make very small satellites weighing a few tens of kilograms. This has radically changed the space missions point of view. Furthermore, also from the launches point of view it has allowed many changes as many companies tend to focus on the launch of satellites with masses of a few hundred kilograms allowing, among the others, launching costs reduction.
Thanks to the possibility of introducing COTS-type devices discussed in subsection 1.2.1 in the space environment, the common vision is to move from few large satellites system to a system with many more satellites with smaller dimensions that deal with more specific tasks and also having lower costs [7]. This is also greatly encouraged by the entry of many private entities into the space sector allowing a gradual increase in terms of performance thanks to the competition introduced.

Field Programmable Gate Arrays (FPGAs) discussed in section 1.2 provide high computational density and efficiency for many computing applications by allowing circuit customization for many applications of interest. FPGAs also support programmability allowing to modify the circuit at a later time through reconfiguration, allowing on the one hand to make updates and further improvements but, on the other hand, also to take advantage of the unique ability of FPGAs to customize the system for greater reliability problems discussed in section 1.4. Mainly for these reasons there is a great interest in exploiting these advantages in space and other

radiation environments [8].
Finally, the increased resource availability and performance margin enjoyed by FPGAs enable the inclusion of effective mitigation techniques both at the design level and within its development flow. On the other hand, these improved parameters allow the introduction of overlay infrastructures to increase the programmability and abstraction in the implementation of the target functionality. Therefore, these opportunities open the way for the development of scalable and reusable platforms with high reliability not strictly bounded to target custom designs.

1.2 Field Programmable Gate Array

FPGA stands for Field Programmable Gate Array and is a particular type of device that can be customized after manufacturing to implement different functionalities. Specifically, they are made up of an array of three main blocks, which according to the Xilinx (i.e., one of the main FPGA manufacturers) nomenclature can be referred as: Configurable Logic Block (CLB) units, RAM Memory Blocks (BRAM), and Digital Signal Processor (DSP) as can be noticed in Figure 1.1.
CLB blocks have historically been present in the FPGA, they are the elements that are actually programmable and are in turn composed of digital logic components such as multiplexer, Flip Flop (FF) and look-up table (LUT) which can be suitably configured to implement logic functions. On the other hand, BRAMs and DSPs have been introduced over time to increase the functional capabilities of the device.

FPGAs are usually programmed using hardware description languages (HDL) such as Verilog and VHDL. Then, a file describing their configuration called bitstream is generated through a specific tool and will then be stored in a specific configuration memory location inside the board. One of the main FPGA characteristics is the possibility of loading new configurations (both before and after deployment) and above all downstream of manufacturing [9] (in opposite with ASIC device). In general, the reconfigurability is the main FPGA feature and places it as an intermediate device between ASIC (highly customizable with respect to the required performances but not flexible at all) and general-purpose computation (highly flexible towards any type of application, but not optimized, as it is *generic*).
Recently there has been an increase in the use of FPGAs thanks to both the high computing capacity and the considerably lower energy consumption [10].
Among the most common applications are the medical electronics sector, automo-

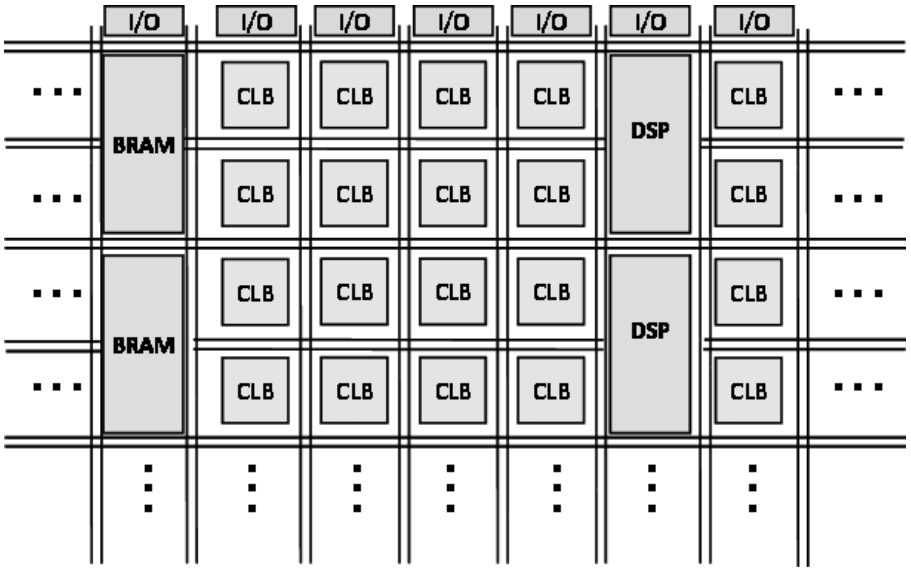

Figure 1.1: Internal FPGA Configuration [1]

tive sector, transmission, consumer electronics and last but not the least, in the aerospace sector.
In particular, the wide use of these devices in the aerospace environment is mainly due to the possibility to perform pre-prototyping, to the fact that often few samples are needed for these applications which would not justify the high costs to be incurred for the design of an ASIC system, and for the possibility of increasing reliability through the use of mitigation techniques which will be discussed in section 1.4 that can be implemented thanks to the large number of resources available.
In particular, for aerospace environment, different types of devices are present on the market, in the past in fact mainly Rad-Hard type devices were used (i.e., devices specifically strengthened to withstand the problems that may arise in the space).
In subsection 1.2.1 will be shown how nowadays there is an increasing use of Commercial Off the Shelf devices which allow to obtain several advantages in terms of performance, price and production times [11].

1.2.1 Commercial of the Shelf

Commercial of the Shelf (COTS) is defined as a device that is [12]:

- A commercial item;

- Sold in substantial quantities in the commercial market; And
- Offered to the State, by virtue of contract or subcontract at any level, without modification, in the same form in which it is sold on the commercial market.

The use of these systems is increasing sharply and ranges in different fields of application including space environment.
In space, COTS devices compete, as anticipated, with conventional radiation-resistant (rad-hard) components by offering the benefits of higher performance, faster/cheaper development, ease of maintenance, ready-to-use development systems [13].
From one point of view, devices of this type in space can be severely damaged due to the extreme conditions discussed in section 1.4. However, through the use of special techniques discussed in subsection 1.4.3 it is possible to manage these problems and obtain a good degree of reliability.
Moreover, the use of these devices has led to several advantages including the possibility of having several devices of this type on board a single satellite thanks to their low cost. Lately, the use of COTS in highly reliable space applications is on the rise and ranges from memory chips in flagship planetary explorers to various pico/nanosat parts [13].

1.3 Satellite Communication Systems

Since the analog technology is fixed and does not allow the user to have any flexibility, the trend over the years is to move towards digital systems allowing to use programmable hardware logic but also software implementations. This technology trend, in the field of space communication find its best example in the shift from analog radio to Software-Defined Radios (SDR) techniques with which the final product results in a compact and flexible communication system.

Applying this concept to small satellites can increase data throughput, add the possibility to perform software updates over-the-air and make it possible to reuse the hardware platform for multiple missions with different requirements [14].

As anticipated, old space telecommunication systems were based on analog systems but have been replaced by digital ones using analog to digital conversions and vice versa, in the following paragraph a brief explanation is provided.

1.3.1 Digital Regenerative Processor

Initially, GEO satellites for telecom applications were based on the ***bent-pipe*** approach. In this configuration, the transponder receives a weak signal which is shifted from the receiving frequency to the transmitting one. In addition, the system performs certain operations such as amplification of the received signal and some purely analogue functions. A scheme of principle is shown in the Figure 1.2.

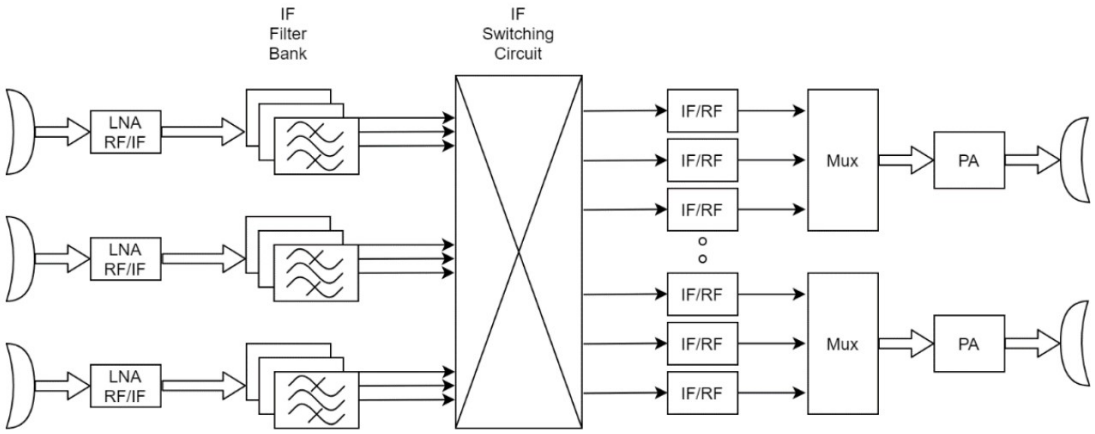

Figure 1.2: Bent-Pipe Architecture

Modern systems maintain all the characteristics of bent-pipe systems, but add front-ends processing and an Analog to Digital Conversion (ADC) and an inverse Digital to Analog Conversion (DAC) to perform digital channelization functions. Moreover, regenerative processors for GEO telecom satellites are mainly based on rad-hard devices, which provide good level of flexibility and in-orbit re-configurability.
Satellites functionalities could be further extended to fully regenerative repeaters adding feature such as multichannel digital re-generations supported by Fast Fourier Transform (FFT), digital filtering, demodulators and modulators, channel encoding/decoding as can be noticed from Figure 1.3 [15].
Finally, high performances are also obtained not only through the new communication systems but also thanks to the newest coding techniques. Channel coding allow to obtain high data throughput with low Bit Error Rate (BER) by sending appropriately coded information which will then be reconstructed at the receiver side.

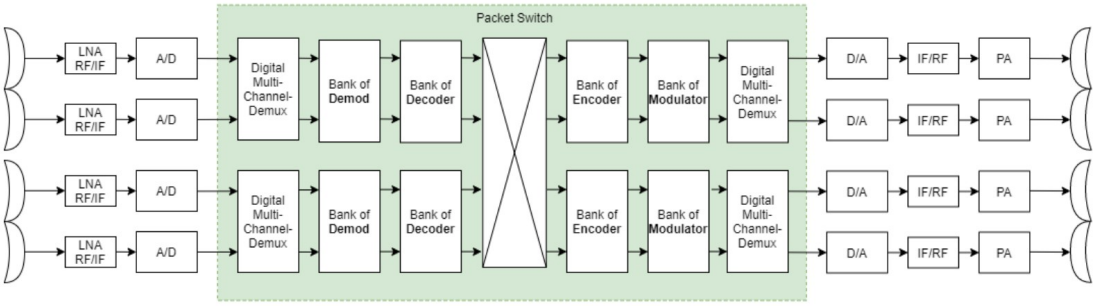

Figure 1.3: Regenerative On-Board Processor Architecture

1.3.2 Channel Coding

Channel coding plays a very important role in modern communication systems. For years, researches have been carried out to find coding schemes allowing to reach that channel capacity. Turbo Codes and LDPC codes have solved the "capacity-approaching challenge" for a point-to-point channel getting very close to channel capacity limits. Typically, Turbo codes are good choices for power-constrained links, instead LDPC codes serve well for higher data rate links when bandwidth is constrained.
The LDPC codes will be explained in more detail in section 3.3, as they will be the benchmark used in the program and in this thesis.

1.4 Deployment in Harsh Environment

The main problems when designing an electronic device for space applications are related to the environmental threats to which devices will be exposed (e.g., vacuum, radiation, extreme temperatures and many others).

Temperature and radiations threats consist of the most challenging aspects to deal with and they will be presented in details in the following.
The effects of environmental problems in space can be very serious as to cause the failure of the device and can range from intermittent and recoverable misbehavior up to the complete non recoverable failure of the system. The devices suitability is usually evaluated firstly through analysis and mathematical models, and in later stages also through thermal and radiation testing on the component.

1.4.1 Temperature

Heat transfer into space can occur mainly due to the effect of radiation or conduction.
Among them the main source of heat is the irradiation which has mainly three sources:

- **Direct solar radiation:** it is the main source of radiation and therefore of heat in space. However, direct solar flow of energy coming from the sun is not constant, for two main aspects.
  First, sun energy flow itself is not constant but depends significantly on the solar cycle. Secondly, the flux impacting an Earth orbiting satellite is strongly influenced by the Earth's orbit around the sun.
- **Reflected solar radiation:** it refers to Earth reflected radiation into space. However, a satellite orbiting the earth will not always be subject to this type of radiation but only when it is in the area where the reflection itself occurs.
  Furthermore, the reflected energy flux is not constant as the flux itself is not and because different reflective properties are present on the earth surface (e.g., clouds, oceans or continent).
- **Earth radiation:** it is a combination of infrared radiation emitted by the atmosphere and the surface of the earth. This aspect is greatly influenced by the climatic conditions of the earth both on the surface and in the clouds.

As in an any electronic device, keeping under control temperatures allows for higher performance, lower power consumption and to prevent system failure. Therefore, preliminary considerations for devices design become more complex since not only internal thermal dissipation is present.

1.4.2 Radiation

Radiation is the main problem encountered in space device deployment and also the one on which the program, presented in chapter 2, is focused.

The typical effect is an energy transfer from the radiation to the atoms, which causes an excitation or ionization of the electrons inside the silicon. If a sufficiently high energy is transferred, the semiconductor characteristics can change leading to transient or permanent misbehavior.
In detail, a long exposure to radiation can lead to an accumulation of effects that can permanently damage the MOS. This effect is considered to be the maximum Total Ionizing Dose (TID) a device can receive before it stops working.
On the other hand, single ionizing particles can instead cause instantaneous problems called Single Event Effects (SEEs), which can be permanent [16] (i.e., that changes the structure of the semiconductor causing an enduring failure) or transient (i.e., which are extinguished after a while) problems. In the following these last ones will be analyzed being the most critical for FPGA devices [17].

Transient Effects

These problems are also called *soft errors* as they can extinguish after a short period of time. The most common are the following [18]:

- **Single Event Upset (SEU):** it is a corruption of the content of a memory element. This typically results from a sudden change in the charges distribution within the memory control logic causing a bit inversion in a specific information bit called bit-flit [19].

- **Single Event Transient (SET):** when an ion interacts with the p-n junction in the active region of one transistor it can temporarily break the barrier of the junction causing a voltage spike [18].
  This type of problem can be present inside a memory and if it is sampled from a memory element, it cause a SEU.

- **Micro Single Event Latch-Up (Micro SEL):** it is a recently studied problem mainly originated due to dimensions and power supply level reduction. This problem occurs mainly near the input and output terminals of a logic gate.

In order to improve electronics system reliability for space applications, different mitigation technique can be exploited with the aim of hiding or solving the problems generated by radiation.

1.4.3 Mitigation Techniques

The techniques used in FPGA solutions are different, among the others can be stressed shielding, derating, redundancy and repairing. In this section a particular focus will be made on redundancy and repairing techniques that are the most common in FPGA hardening.

Redundancy

This approach does not solve radiation effects but enable to mask them. Structural redundancy consists of using several modules working in parallel and computing the same calculation in order to have different results of the same operation and then choose the results comparing them.
Within this class of techniques, the most common is the Triple Modular Redundancy (TMR). It consists of triplicating the designed logic in the programmable resources and then putting a logic voter which will provide the result based on two out of three output. A high-level diagram is shown in the Figure 1.4.

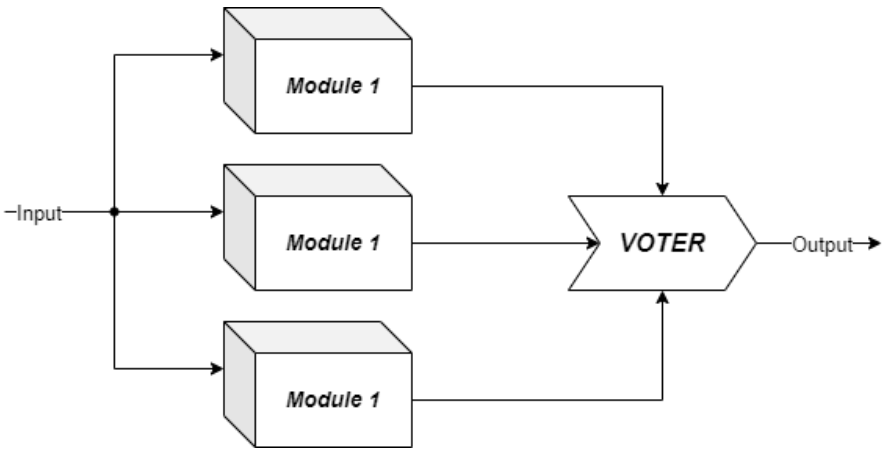

Figure 1.4: Triple Modular Redundancy Mitigation

The use of this technique requires a greater amount of logic (i.e., at least three times greater) and therefore considerations regarding additional resources and power consumption should be made. However, the functional approaches of this technique are much less costly in terms of resources. In fact, very often data replication and instruction memory techniques are used [20] and [21].

Repair

This method consists in correcting any eventual bit-flip that has occurred, this is done by performing a refresh of the correct value inside the affected memory element. One of the most popular and effective methods is scrubbing [22].
There are several methods to perform scrubbing, two of the most common are blind and readback scrubbing. Blind scrubbing is based on memory updating without considering whether they are corrupted or not. The readback scrubbing, instead, is only performed when a problem is detected. Another classification can be made with respect to updating timing: it can be identified periodic scrubbing which periodically occurs or, instead, on-demand scrubbing which happen only upon a request. One of the main advantages of this technique is the ability to correct errors preventing them from accumulating.

1.5 Summary and Highlights

The use of COTS devices enable the possibility to obtain performances above those possible with current Rad-Hard systems. Moreover, state-of-the-art technology allow to meet future Telecom applications demands requiring higher data rates for the uplink, higher distances and improved regeneration capabilities. For these reasons the use of COTS device for space applications is a rowing trend in last years.
On the other hand, as discussed in section 1.4, when deployed in space, COTS devices can be severely harmed by the harsh environmental conditions of high vacuum, extreme temperatures, high levels of ionizing radiation.

In this view, the program presented in chapter 2, aims to provide a new perspective on possible architectures able to leverage the concepts just discussed in previous sections. In fact, the envisioned architecture will include an array of programmable accelerators capable of maximizing the system performance by leveraging a high degree of parallelization, also providing the possibility to efficiently support today satellite systems for communication.
Moreover, taking advantages of the clustering approach it will be possible to have enough resources to obtain a structure able to address with appropriate strategies the reliability problems discussed above.

1.6 Outline

The current section describes the organization of the thesis and gives a short description of the content of each chapter:

Chapter 2 - provides an overview of the program, detailing its goal and highlighting where the work carried out in this thesis takes place. Moreover, it gives an overview of the envisioned structure and briefly explains the different solutions analyzed with respect to different clustering strategies. Finally, it focuses on the selected approach and presents the identified structural strategy.

Chapter 3 - in this chapter the technical concepts relevant for the understanding of the proposed architectural solution are provided. In detail, a focus on the two main blocks on which the accelerator design is based (i.e., BRAM and DSP unit) will be discussed showing their main features. Finally, backgrounds on the decoder algorithm selected as telecom case study are provided.

Chapter 4 - details the architectural solution studied and implemented for the array of accelerators, explaining the evolution of the structure created to then converge on the final design. Then, the strategies used to exchange data between computing cores and the envisioned memory structures will also be briefly described. The chapter ends with a feasibility demonstration done presenting the application of these studies to the selected case study.

Chapter 5 - presents the results, obtained through analysis, simulations and on board tests. In particular, a trade-off analysis will be presented and then an on-board test is discussed.

Chapter 6 - closes the thesis by discussing the main conclusions and possible future developments of the work.

2

Program Framework

The following chapter wants to present the framework of the European Space Agency (ESA) project focused on Advanced Research in Telecommunications Systems and consists of researching and developing state-of-the-art and innovative hardening techniques for Commercial-Off-The-Shelf (COTS) Field Programmable Gate Array (FPGA) devices for digital telecommunications payloads.
In the following the preliminary considerations regarding possible implementations and software/hardware architectural trade-offs will be presented.
The adopted approach is a clustered one consisting of an inter-FPGA network organized into Tiles and Nodes: a Node is a single FPGA device while the Tile is a four nodes cluster structure. Each single FPGA node is used as an hybrid structure which enable both general-purpose computation through the usage of soft microprocessor and compute intensive custom operations. In fact, on the one hand a soft microprocessor is used and, on the other hand, customization is achieved thanks to the atomic functional block that has as its reference the digital signal processing units (DSP) present on the FPGA and managed through appropriate host processors.

Specifically, in section 2.1 a brief analysis will show the main clustering strategy benefits.

Then, in section 2.2 the structure of the single Node is presented and the main two different architectural solutions illustrated. Moreover, how the final structure was obtained and what advantages it brings will be depicted.

Finally, in section 2.3 details concerning the Programmable Functional Units approach will be provided.

2.1 Clustering Strategy Benefit

The opportunity of using more than one device to implement the architecture provides extra resources for the introduction of mitigation techniques.
Furthermore, exploiting parallelism and programmability rather than only dividing the application on a custom collection of samples devoted to a specific functionality, the opportunities for a scalable and the flexible re-purposing are maximized.
The envisioned Tile clustered architecture is reported in Figure 2.1 and it is composed by four nodes where each node is composed by an FPGA.

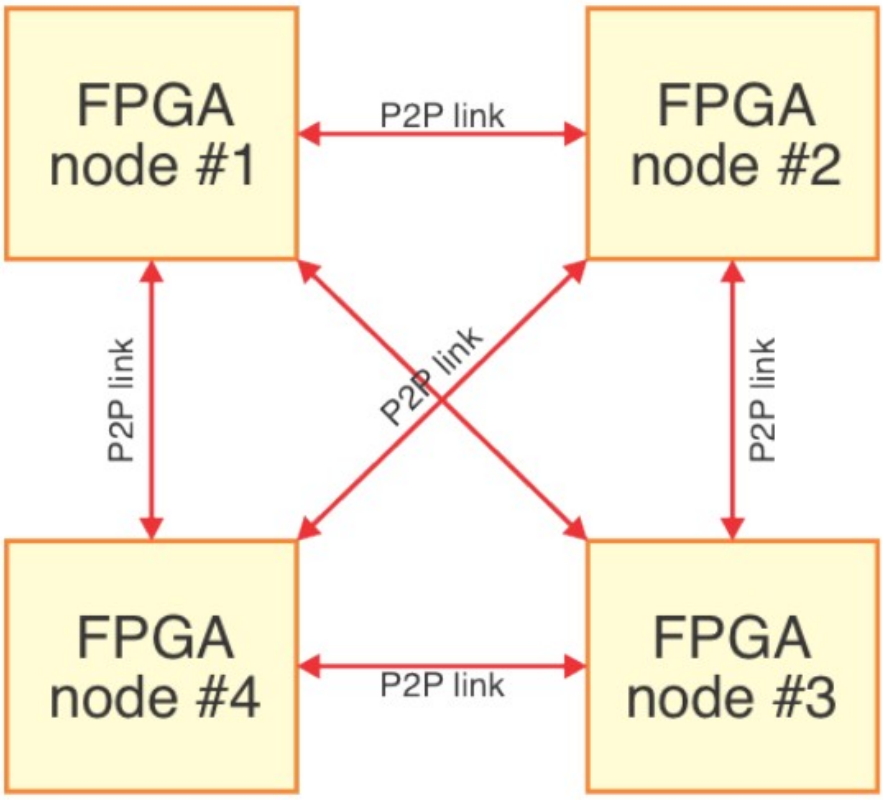

Figure 2.1: Tile full-mesh Interconnections Architecture

In addition to what is reported in Figure 2.1, the final design architecture is envisioned to be composed of at least 4 tiles as an overall clustered structure that allows to increase the available resources once again.
The use of multiple devices (e.g.,four), allows to have high gains in terms of available resources also compared to the rad-hard solutions currently on the market.

Moreover, the price can also be reduced considerably (i.e. by about two orders of magnitude).
The gain in resources and performance provide opportunities to leverage several mitigation techniques, as the one discussed in subsection 1.4.3. Furthermore, with the improved parameters it is possible to implement infrastructures for high programmability in the implementation of the target functionality. These opportunities, therefore, pave the way for the development of scalable and reusable solutions, highly reliable platforms not strictly linked to customized target projects.

2.2 Architecture

To map the functionality on the system different clustering approaches can be adopted.
The key to achieving effective partitioning is determining the best approach for breaking down the work to be distributed across the different nodes.
The main strategy can be related to the heterogeneous or homogeneous one [23].
In the first case, each node is different from the others and it has a dedicated task and can be customized and optimized in order to execute it. In the second approach, each node is architecturally equivalent and the task is homogeneously distributed.

In heterogeneous approach different cores are able to develop and carry out different tasks in a parallel and in independent way. This usually translates in providing better performance.
This strategy optimizes each sub-block in order to implement a specific application. This solution can maximize performance but on the other hand it provide a limited flexibility as each block is ad-hoc developed to execute a specific functionality.

On the other hand, the homogeneous architecture is mainly oriented in increase scalability and reusability. The idea is mostly focused in nodes redundancy that enables tasks decomposition in parallel computation.
In this approach different FPGAs are used but all of them are configured in the same way exploiting programmability since no specific function is envisioned. Moreover, performance of homogeneous approach cannot reach the optimizations opportunities of custom data path of heterogeneous ones.

These considerations motivate the Hybrid Solution selected as baseline for

the project, presented in subsection 2.2.1. The goal is to obtain a trade-off between flexibility and scalability that homogeneous structures allow without losing the performances that can be reached with a dedicated structure such as the heterogeneous one.
To do that, the idea is to use functional units based on re-configurable processors (i.e., DSPs unit) in order to maintain a homogeneous structure but with the possibility of re-configuring and adapting the operations according to the needed. In this way, the execution is no longer requested from generic microprocessors but is attributed to properly instructed Programmable Functional Unit (PFU) which is the main aim of this thesis and will be discussed in detail in the following chapters.

2.2.1 Hybrid Structure

In order to obtain a system able to maximize programmability and granularity, a trade-off between the two main architectural approaches briefly discussed in previous section has been considered.
In this view, the identified architectural strategy will be equipped in a hybrid solution, developed by homogeneous software-oriented approaches, while maintaining a high degree of efficiency in the optimal customization of specific functionalities, as typically allowed by heterogeneous customized hardware solutions.
Through this efficient compromise, it was decided to implement an architectural structure capable of maximizing the aspects of programmability and customization through a clustered DSP-based Functional Units Array.
In each node, the array of DSP-based functional units will be managed through a distribution of tasks and processing data by a programmable processor. This processor will also be in charge of handling communications to the outside and exchanging data with the other nodes of the tile. An high level structure of the single FPGA is reported in Figure 2.2.

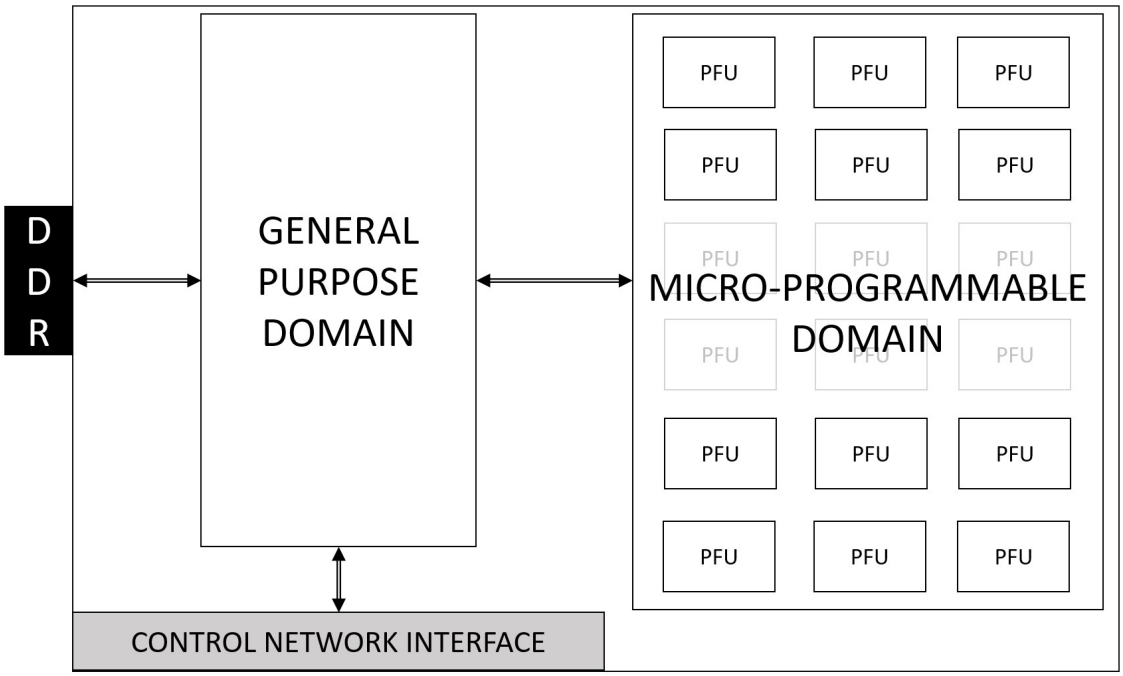

Figure 2.2: Hybrid Node Architecture

Each FPGA includes two main component types: fixed and programmable. The fixed ones mainly refer to the various interfaces between the internal blocks and to the connections to the outside for the other nodes of the tile. The programmable types, instead, refer to the hierarchical structure that can be seen in the figure, through three main levels of programmability:

- The deepest level of programmability is at the DSP units level, which allows to implement a wide range of operations based on specific signals and it is coupled with a BRAM, included into each `PFU` block.
- Subsequently, the second programmable level is constituted by the array of PFU cluster, forming the `Micro-Programmable Domain`.
- Finally, the last layer is the one formed by the soft-processor acting as node manager, identified as `General Purpose Domain`.

2.3 Programmable Functional Unit

The use of different Programmable Functional Units (PFUs) allows to perform many calculations on small data, benefiting from a high parallelism, and at the same time developing the target algorithm. Each PFU will be set with the appropriate opcode and will work on the corresponding data in order to process the current layer.
In other words, each PFU receives a partial subset of values which are stored in the dedicated local memory and which will then be appropriately processed by the DSP cores and by the boundary logic.

Nevertheless, thanks to the configuration in Figure 2.3, it is possible to maximize the utilization of the FPGA by successfully couple DSP and BRAM present in each PFU. As the structure itself of the FPGA, in fact, is organized in columns of BRAMs and DSPs, the exploitable regions increase as much.
Finally, paying attention to the arrangement of the PFUs within the FPGA, it will also be possible to reprogram suitable regions enabling the possibility to reconfigure and restore a certain number of PFUs that will fail by implementing the scrubbing technique mentioned in subsection 1.4.3.

2.3.1 High Level Functional Unit Structure

In order to obtain a system that is as high-performance and flexible as possible at the same time, the wired primitives present in FPGAs for digital signal processing (i.e., DSP) are very attractive.

According to specific control sequences, a DSP unit is able to process various operations ranging from classical arithmetical or logical ones to specific parallel opportunities, more detailed analysis will be provided in following chapters and in [2]. Among these modes, the Single Instruction Multiple Data (SIMD) is extremely attractive configuration targetting the parallel implementation of the selected testbed. In the following chapters it will be examined in depth.

Furthermore, in current FPGAs the DSPs are integrated in order to easily interact with the user memories allowing faster local memory usage [24].
These considerations made possible to base the decoding activities core on the DSP using the described configuration also reported in Figure 2.3.

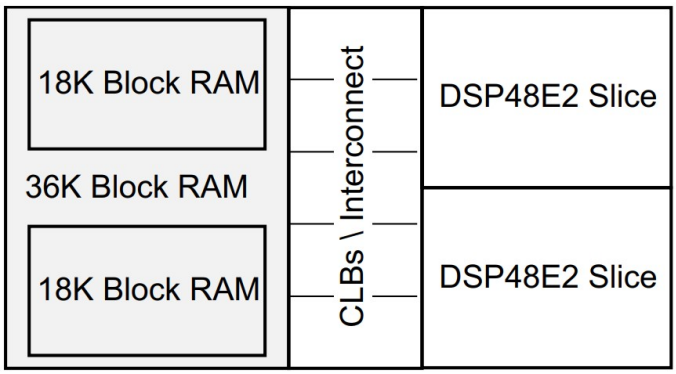

Figure 2.3: DSP Tile

Finally, within each functional unit, a series of registers and components useful for carrying out operations and sorting data will then be made available.

3

Technical Background

This chapter aims to provide a brief description of the main characteristics of the blocks and of the concepts used in the design of Programmable Functional Units.

As anticipated in previous chapters, DSP Unit and BRAM are the two fundamental blocks on which the realization of the Programmable Functional Unit is based.
Therefore, they will be described in detail and the studies made to reach a level of knowledge such as to be able to put programmability into practice shown.

Finally, the chapter closes with some details regarding the selected functional testbed aimed at demonstrating its validity in terms of programmability and scalability, also providing some useful details for the project understanding.

3.1 Digital Signal Processing Unit

A Digital Signal Processing unit is a specialized component device present on FPGAs able to execute arithmetic and logic operations with a very huge fields of applications. In detail, the selected development board, uses the DSP48E2 primitive [2].
The purpose is to dynamically manage it in order to run-time manipulate operations types.

The auto inference by the synthesizer is the common approach for instantiating a component like the DSP unit. It is an instance made directly in the HDL code that allows to have full access to all the configuration and control signals of the unit. However, using such an instance can be very complicated as it requires very thorough and detailed knowledge of each individual control signal.

In [25] an interesting solution is proposed. The idea is to use a `wrapper` file that incorporate the primitive instance and setting all the generics and port map signals as a default value. The wrapper, in fact, has exactly the same generics and ports as the primitive. In this way the instance becomes much more easy and only the interested signals will be changed. These features allow to have a very compact instance almost as much as a behavioral one, while still having access to all the possible modes of the DSP.

DSP usage makes possible to achieve the scalability and flexibility without sacrificing performance. The goal, in fact, is to be able to dynamically manage the DSP, as anticipated, so as to be able to adapt its feature to the operations to be performed. Some of the most common uses for the DSP are for example the MAC (Multiply and Accumulate) functions in which the multiplier and the adder cooperate. Other uses are for example FIR (Finite Impulse Response) [26] filters exploiting cascade capabilities of the DSP.
The following thesis has the main aim of demonstrate DSP potentiality even outside the common uses mentioned above.

3.1.1 Architecture Analysis

As the DSP usage is one of the main points of the PFUs it is important to manage the `wrapper` described in the correct way.
To do this, the main steps are as following:

1. Correctly understand how it works and evaluate constraints or limits;
2. Figure out micro-controlled possibilities of control;
3. Understand how different operations can be dynamically selected to the same DSP;

The structure of the DSP can be found in [2] and is reported below in Figure 3.1.

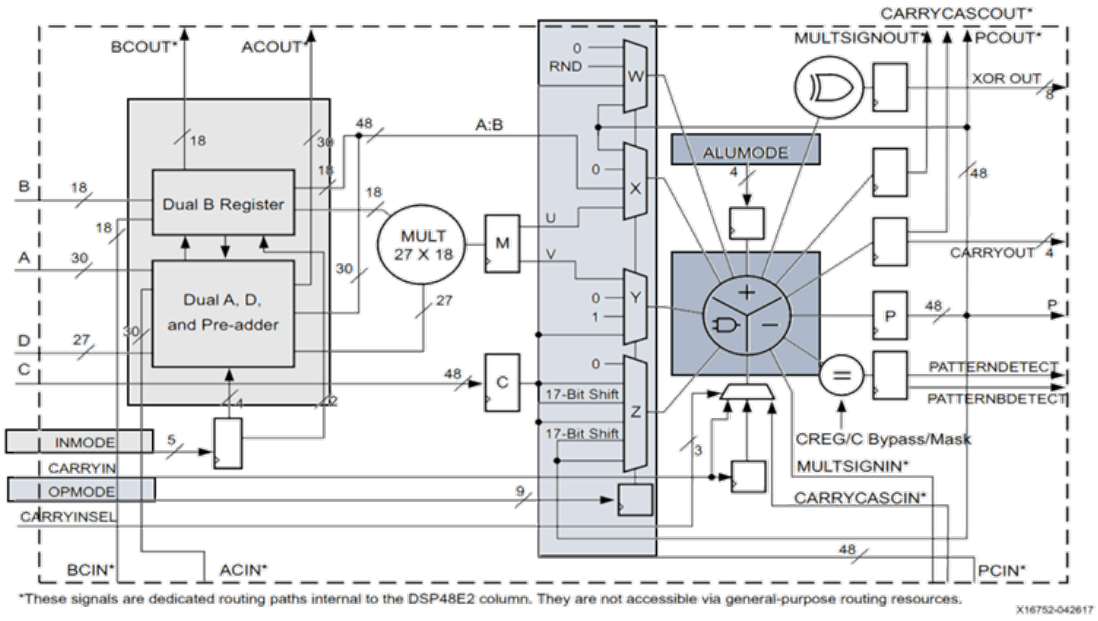

Figure 3.1: DSP Architecture [2]

From Figure 3.1 different control signals can be noticed. All of them together allow to control the DSP behavior. However, three among them deserve to be mentioned as they are the ones with which the actual arithmetic operation and data combinations are selected.

- **OPMODE:** it is a 9-bits control signal and it is responsible for X, Y, Z and W multiplexer output arrangement. It is divided into 4 section that are respectively of 2 bits for each X, Y and W multiplexers and 3 bits for the Z one.

- **INMODE:** this signal drives *Dual B Register* and *Pre-Adder* block configurations. It is the trickiest among them. It is composed by 5 bits that combined with three static controls (namely *amultsel*, *bmultsel* and *preaddinsel*) can vary the possible combinations of input signals.

- **ALUMODE:** it consists of 4 bits and is responsible for selecting the arithmetic or logic ALU operations.

3.1.2 SIMD features

Among the main modes of the DSP the SIMD (Single Instruction Multiple Data) surely deserve to be mentioned. This feature allows to work in parallel with multiple data sets sharing the same instruction commands.
In Figure 3.2 a scheme which represents the possible configurations is reported.

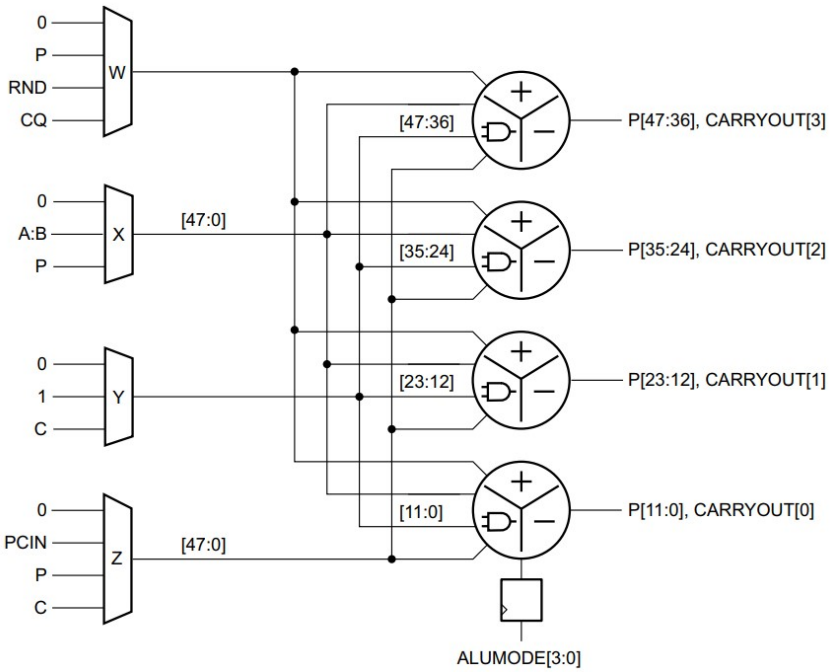

Figure 3.2: DSP Architecture in SIMD Configuration [2]

The DSP allows to use this mode in two different configurations:

- **Two 24-bits:** the ALU will work separately on two data sets.
- **Four 12-bits:** the ALU will work on four different data sets.

Splitting into more sections the parallelism it is possible to exploit higher performance. By using the four 12-bits configuration, for example, it is possible to improve the throughput up to 4 times.

On the other hand, the SIMD mode does not allow to use all DSP features, but it is limited to the Arithmetic Logic Unit (ALU) section and only some data input configurations are allowed.

3.2 Block RAM

The BRAMs are another types of primitives present on the selected board [24] are organized in 36 Kb storage area cells. In detail, the 36 Kb memories can be used as a single memory block or be divided into two equivalent 18 Kb memories.

As previously discussed, their organization is designed in order to optimize access especially when working with DSPs unit through structures such as the one in Figure 2.3 replicated several times on the board as reported in Figure 1.1.
Furthermore, it can be configured to work in three main modes:

- **Single Port:** in this case there is a single port for both reading and writing operations and therefore only one of them can be performed at a time.
- **Simple Dual Port:** in this case there are two independent ports, one for writing and one for reading, which can be used simultaneously.
- **True Dual Port:** in this case there are two independent ports, both of which can be used for reading and writing simultaneously.

Both write and read operations are synchronous and require a clock cycle.

3.3 Selected Testbed

The functional testbed identification is mainly based on demonstrating the implementation opportunities trying at the same time to cover a relevant scenario for space communications. Channel decoding is one of the most relevant; notably, the Low Density Parity Check Codes algorithm [27] has been chosen thanks to its high error correction capabilities.
Furthermore, the parallelization opportunities provided by the LPDC algorithm allows to exploit the clustering concepts presented, allowing to perform operations with several PFUs that cooperate in an optimal way for channel decoding.
Parallel approaches [28] have ranged from hardware implementations to software ones. Particularly, either hardware implementations, multi-core processors implementations and purely software techniques to maximize multiple data computations with a single instruction have been exploited.

From one side, implementations with high levels of parallelization maximize performance but produce an explosion of area and number of resources used, also reducing the flexibility of the structure. On the other hand, a serial structure allows greater flexibility but reduces the possible performance to a minimum.
Therefore, solutions that use a partial parallelism only on specific decoding sections have been exploited obtaining an overall trade off between serial and parallel computation capable of maximizing the gains in terms of performance and flexibility.

3.3.1 Algorithm

An interesting algorithm is the LDPC extended telemetry codeword encoding algorithm involving an iterative technique known as Belief Propagation. This algorithm propagates messages (i.e., probabilities) through a graph that can be represented with the so called code parity check matrix H. The main characteristic of the H matrix is the low density of ones. The general structure is reported in Figure 3.3. More detailed analysis can be founded in [3] and [29].

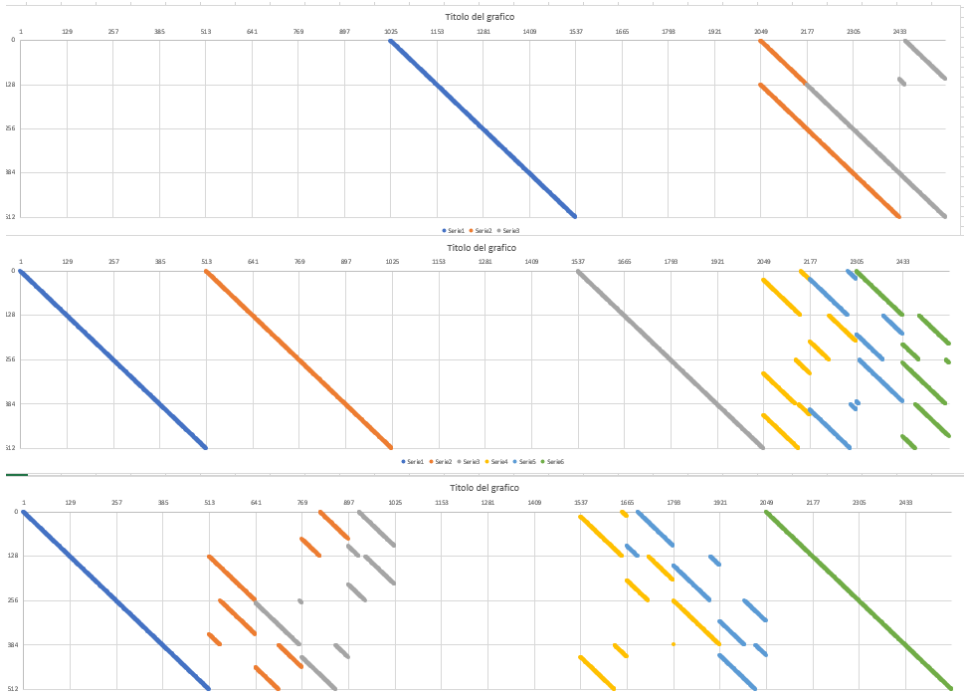

Figure 3.3: H matrix [3]

Analyzing the matrix of the New Generation Uplink CCSDS [3] can be noticed how it can be decomposed into 128 x 128 sub-matrices. Each of these sub-matrices can be equal to zero or have a single element per row and column equal to one.
These non-empty matrices are called **circulats** and are formed by Identity matrices shifted by a certain amount. Finally, it can be noted that in a single row there are only 3 or 6 circulating.
In [4] an extensive explanation about the algorithm is discussed.

In general, one of the most popular layered approach consist of the Min-Sum-Algorithm (MSA) where the decoded words are obtained through a certain number of iterations converging to the final results with a good degree of certainty.
Assuming to have:

y - received codeword.

m - rows of H matrix.

n - columns of H matrix.

The parallelizable and compute-intensive part of the algorithm is reported in Figure 3.4.

Iteration Loop
  for all $\ell = 1, \ldots, L$ **do** ▷ Loop over horizontal layers
    for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **VN-processing**
      $\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$;
    for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**
      $\beta_{m,n} = \prod_{n' \in \mathcal{H}(m) \setminus n} \operatorname{sgn}(\alpha_{m,n'}) \cdot \min_{n' \in \mathcal{H}(m) \setminus n} |\alpha_{m,n'}|$;
    for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **AP-update**
      $\tilde{\gamma}_n = \alpha_{m,n} + \beta_{m,n}$;
  end (horizontal layers loop)

Figure 3.4: Layered MS Decoding Algorithm [4]

The layered algorithm in Figure 3.4 is one of the most advantageous and that is why it was chosen. This makes out that the layers must be calculated serially, since in many cases the output of the previous layer is the input of the next one. On the other hand, all the messages related to a single circulant are independent. Therefore, at least 128 elements of the 1024 H-Matrix reported in Figure 3.3 (i.e., samples per circulant) can be computed independently.

From Figure 3.4 can also be identified three main sub-processing cycle:

1. **VN processing:** through a sequence of subtractions between the γ and β values of the preceding iteration, this phase creates intermediate values called α. Where β are the outcomes of the previous CN processing iteration (or 0 in the case of the first iteration), and γ are the results of the previous AP update iteration (or the inputs of the matrix in Figure 3.3 in the case of the first iteration). It is a vertical processing so no type of dependency exist between samples in the circulant. Therefore, depending on the number of circulant in one layer (i.e., 3 or 6) all messages can be processed in parallel.

2. **CN processing:** provides the computation of new β values by concatenating the product of the signs of the other circulating α values with the minimum of their absolute values. Also in this case the processing within a circulant can be done concurrently. However, in this case dependency exists between each circulant, so at most 128 messages can be processed independently.

3. **AP update:** produces the new γ values by summing the α and β values that have just been computed. This final processing cycle also can be computed in parallel within the layer since no dependencies between circulant exists.

These steps are repeated for all layers of the matrix H, and for each iteration it is finally executed the hard-decision and the syndrome check.
These VN, CN and AP cycles are the core of the algorithm and will be mapped on the Programmable Functional Units that will exploit the SIMD approach proposed in section subsection 3.1.2.

4

Design Overview

In this chapter the Programmable Functional Unit design is presented. It represents the main computing resource to accelerate Digital Signal Processing tasks involved in telecommunications and case study of this thesis. Moreover, it will be shown how the PFUs will be able to communicate with the entire system to perform required operations (e.g., the LDPC decoding selected as testbed).

First, a high level block diagram showing the Node structure will be displayed and a brief introduction to how data flow is implemented within the system done, then it will be presented a focus on the envisioned structure for PFUs data exchange.

Secondly, the preliminary design will then be presented; it was developed with the aim of gradually approaching the envisioned architecture. In detail, the preliminary architecture is proposed to highlight the the process carried out from the baseline assumptions to the final solution.

Successively, a more accurate description of the final PFU is reported, also showing the main differences and similarities between the final version and the preliminary version previously mentioned.

The high-level architecture of the PFU is explained, providing details also regarding the PFUs cluster architecture and the main components. Subsequently, the key features of the architecture will be highlighted such as the scalability opportunities.
Subsequently, the main features related to system programmability will be explained through the preliminary envisioned Instruction Set Architecture (ISA).

The chapter ends with a focus on the implementation of the selected testbed, demonstrating the validity and feasibility of the desired scalability concepts.

4.1 Node Overview

The accelerator array designed in this thesis, as anticipated, is part of a more complex structure called Node.
In detail, each Node is composed of several blocks whose main purposes are:

i. Communicate information about the node status to the spacecraft and other nodes within the tile.

ii. Exchange data useful for the computations of the node (inbound and outbound).

iii. Monitor the system's health and, if required, react with proper recovery mechanisms.

iv. Manage the transfer of data to and from the main memory for processing.

In Figure 4.1 the high level block diagram showing the design main blocks is reported. From this picture, the following building blocks can be identified:

- The main Processing System which is charge of two main activities: to run the cluster management services, and to run the specific computations and data-flow management operations.
- A dedicated health status monitoring IP core which has the main task of monitor the health status of the Node and the Tile.
- The data link manager which is composed by two main blocks in charge of managing the data flow across Nodes and within the Node itself.
- The *DDR4 Memory Controller*, which enable the access to the on-chip external memory.
- The Data Link connections. In particular, a high-data-rate communication protocol is required for data transfer and consequently the Aurora protocol is envisioned to be chosen, since it allows to reach up to 16 Gbps of throughput on each lane.
- The Robust Bus for Health Status information exchange.
- The Control Link connection which is necessary for the observability of the Node status. In detail, the connection for observability does not require high-data-rate communications and it is therefore envisioned to be designed with an I2C or CAN protocol.

- At least one high-data-rate access and transfer device that allows data to be transferred to and from the memory based on Direct Memory Access (DMA) transfers.
- A dedicated interface infrastructure to feed and harvest data from the Array of PFU-based Accelerators.
- The *Array of PFU-based Accelerators*, consisting of Clusters of Programmable Functional units in charge of accelerating applications parallel and compute intensive tasks.

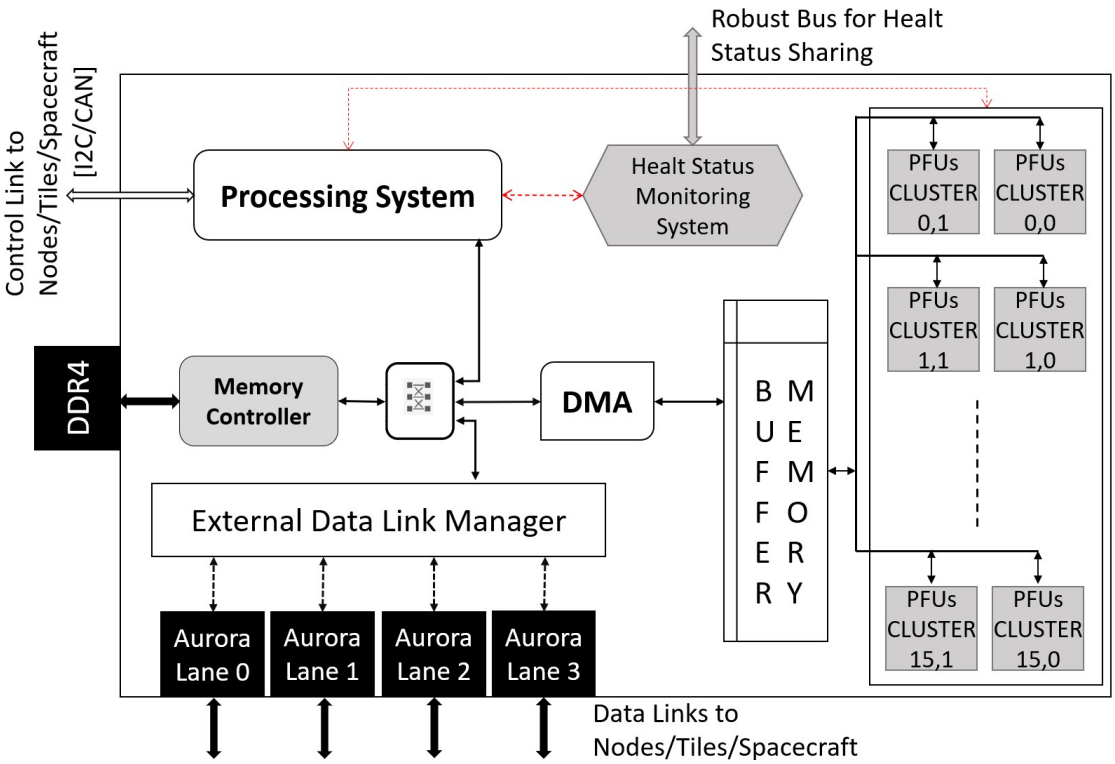

Figure 4.1: High Level FPGA Block Diagram

4.1.1 Design Flow

The identification of the architecture for the accelerator array has been guided by two main drivers. On one hand, to satisfy system requirements, a framework has been developed to achieve flexibility through scalability and programmability. On the other hand, the design has been optimized in order to correctly address to the requested execution of the selected testbed.
The project was designed using standards known as technical specifications, which are the basic criteria that the design must fulfil. In detail, the main technical specifications referred to the accelerator array design are:

i. The Node shall include an array of accelerators.

ii. The Node accelerators shall be programmable.

iii. Each Node accelerator shall include an addressable memory space and a Digital Signal Processing unit.

iv. Each Node accelerator shall support at least the following SIMD instructions on operands of at least 8-bit or less than 16-bits:

- Sum;
- Subtraction;
- Absolute Value;
- Minimum among two values.

v. The array of accelerators shall support the instructions required to perform the CN Processing, VN Processing, and AP Update of a Layered Quantized Normalized Min-Sum LDPC decoding algorithm.

vi. The array of accelerators shall be composed by at least 128 processing elements.

vii. Each 5% of available accelerators in the array shall provide an average contribution of at least 1 Mbps while performing the computation for the Layered Quantized Normalized Min-Sum LDPC decoding algorithm.

viii. The Node shall be able to receive and elaborate information for codewords decoding with a gross rate of 20 Mbps.

In this view, all possible techniques and methods to be employed to properly fulfil the above requirements were investigated using the study of various clustering techniques, feasible strategies to minimise area and maximise performance while fulfilling programmability.
In Figure 4.2 a flow diagram show the methodology used to for the identification of the high-level system architecture.

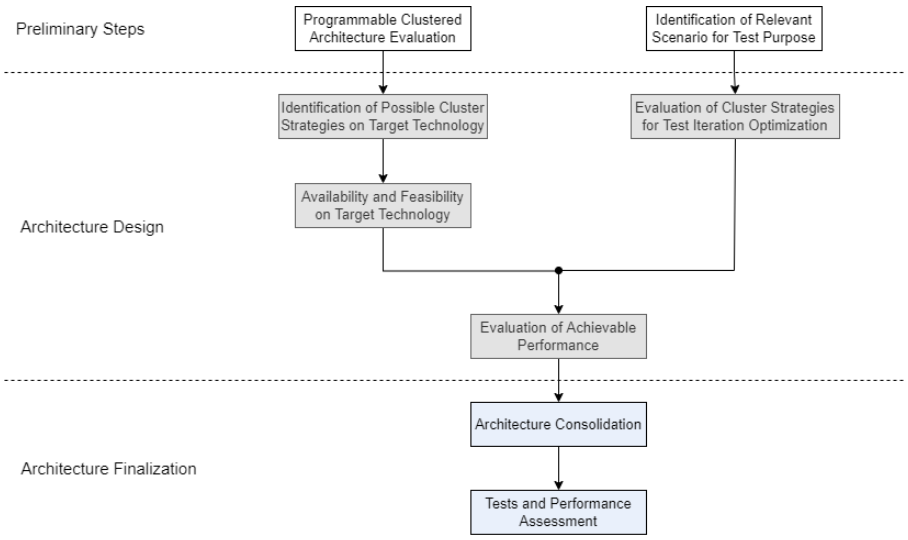

Figure 4.2: High Level Methodology Flow

This assessment highlights the analysis performed according to the clustering approach that will be described in detail in the following sections.

In particular, the preliminary steps, as shown in the flow diagram in Figure 4.2, were to establish a clustered structure that would fulfil the system requirements. In addition, a potential scenario for performing a relevant test appropriate for verifying the design was identified. In detail, the current structure has been developed mainly to support the decoding of the LDPC for the CCSDS (1024; 1/2)[3].

Subsequently, the potential clustering strategies on the target device were examined in the architectural design, taking into consideration the feasibility in terms of resources, as will be explained in further detail in subsection 5.2.1. Parallel to that, the algorithm's capabilities were chosen in order to allow for the right iterations on the intended design. At the end of this step, a preliminary baseline of the achievable performance of this preliminary design was carried out.

Finally, during the architectural finalization stage, the design was validated via functional verification of the system's behaviour, as well as an assessment of the achievable performance has been performed.

More in detail, the PFU design has followed and incremental approach, in order to iteratively validate, test, and improve the envisioned functionality.
The main aspects tackled in this phase have consisted in:

i. Evaluate the DSP hard macro flexibility.

ii. Identify how to effectively couple the DSP hard macro with the BRAM-based PFU Local Memory.

iii. Identify basic functional requirements and which of them shall be implemented in DSP versus custom logic.

iv. Consolidate the preliminary strategy to cluster and manage groups of PFUs presented in subsection 4.2.1.

v. Identify an effective strategy for feeding and harvesting data to/from these clusters.

4.1.2 Data Flow

Data-flow mechanism has to be evaluated with respect to data transfer through memory structures in the design. The envisioned memory units for data storing in the design are mainly two: one is the main memory, the Double Data Rate (DDR4), which is located off-chips (i.e., out of the FPGA) and contains both data and instructions and it is located off chip, while the other large portion of memory is made up of local memories organized as will be described in the following sections. From Figure 4.1 the following data flows can be distinguished:

1. Incoming data from the outside are sent to the device through a high data rate communication link. They are therefore stored inside the DDR which is located off-chip.

2. Then, data has to be transferred in the FPGA through a specific block that allows to have a proper interface with the external memory. As DDR includes both computations data and processor instructions, it is important to manage data transfer in an appropriated manner with the microprocessors.

3. As soon as data have been transferred within the FPGA, they will be transferred to the accelerators array through a specific structure implementing the clustering architecture discussed in subsection 4.2.1.

4. Finally, accelerators array computations results will follow the same path in the opposite direction reaching the DDR memory.

One of the most critical points for data distribution the bottleneck due to the data retrieve from the DDR as it is equipped with a single access point.
For this reason, a specific structure has been designed allowing to maximize the distribution of data as much as possible by minimizing the storing latency but above all allowing to have a reasonable fanout (i.e., number of inputs that can be connected to the output of a only logical gate). In section subsection 4.2.1 this point will be deepened.

4.2 Preliminary Design Overview

The goal of this section is to explain the design flow used to finalize the PFU implementation/realization. In fact, this incremental approach allows to obtain preliminary information in terms on resource usage and power consumption for the design implementation and an initial estimation of the required time for the decoding process.
In subsection 4.2.1, an high level overview of the clustered architecture will be provided, in order to present the envisioned design.
Subsequently, in subsection 4.2.2 a general outlook on the preliminary version of the functional unit will be provided, examining in more detail the main blocks that make up the composition.
Finally, in subsection 4.2.3 it will be described which points can be consolidated and which ones will have to be re-evaluated for the next design release.

4.2.1 Accelerators Array Infrastructure

The main reason for clustering consist of optimizing the data feeding and harvesting management, avoid explosion of fanout and routing, and produce a replicable structure that can scale with required computational effort.

The envisioned structure has been designed in order to meet the requirements of the selected testbed. In fact, in order to maintain performance requirements presented in subsection 4.1.1, a number of PFUs have been estimated to process different H-matrices decoding the LDPC code described in section 3.3 while maintaining performance requirements.
In this view, the structure will allow to process multiple matrices as expected or to re-execute a certain task in a specific region in order to recover from a hypothetical failure of a section of the accelerators owing to the issues stated in section 1.4.
Since not all PFUs can be connected at the same time to the output bus coming from the DDR, it was necessary to think of a structure that would allow to obtain a feasible fanout. Typically, to have a low fanout it is possible to use a serial structure that has a single access point connected to one block after another.
The PFUs organization wants to obtain a trade-off between these two aspects by obtaining a structure with a low fanout (i.e., two) combined with a serial structure that sends data to one PFU after another within the various clusters.
The envisioned clustering approach is shown in Figure 4.3, with the highlights of the sub-portion of this structure.
Considering Figure 4.3, the maximum data width available for the exchange with external memory elements, and the FPGA fabric layout, it has been obtained as optimal clustering approach the following one:

- PFUs to be grouped in cluster of 8.
- Clusters of PFU to be grouped in branch of 2.
- Each cluster branch to be feed sequentially by a sub-portion of 32 bits of the 512 available at the input of the cluster array.
- A total of 16 branches are required to execute support all the required PFU cluster groups.

In detail, at the current stage, each cluster of a branch can be fed serially with 32-bit inputs and starts its computation in a pipelined manner. Conversely, clusters of different groups can process their input in parallel, as they can be fed at the same time.
In fact, according to the current view, PFUs that belong to the same `"Collection"` can make their processing concurrently, while PFUs belonging to the same Cluster can process their data one after the other, as will be explained in detail in following sections.

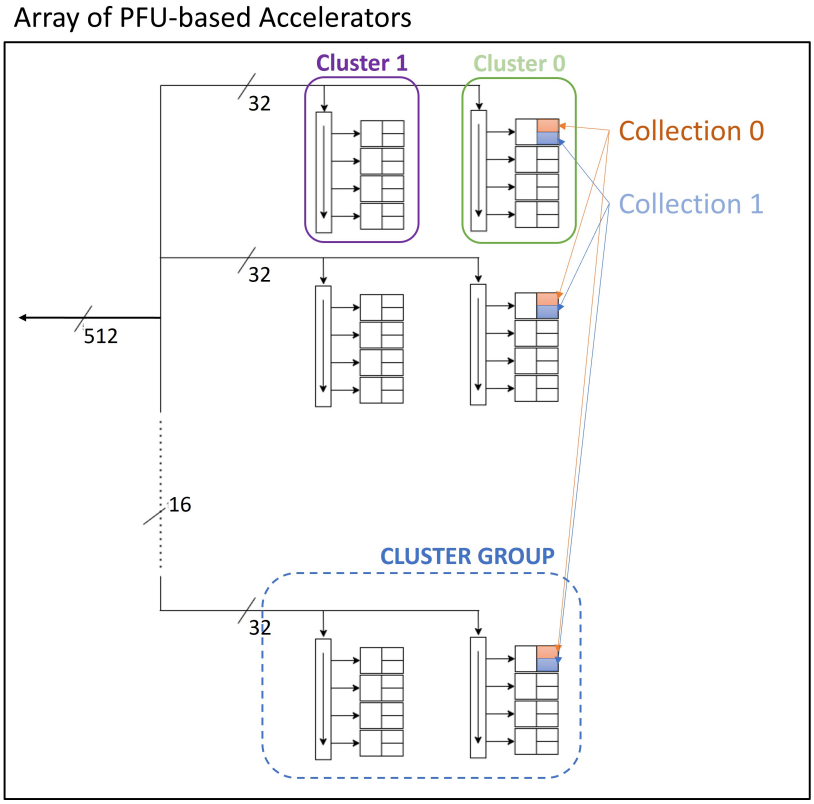

Figure 4.3: Array of Clustered PFU-based Accelerators with the Highlights of "Cluster" and "Collection".

4.2.2 Initial PFU Design

As anticipated, the algorithm (CCSDS Layered Decoding, 1024-1/2) [3] is taken as reference and mapped on the DSP-based functional units through a dedicated state machine which evolves accordingly and provide a simpler set of programming word to the PFU.

In this first implemented version, the DSP core is controlled at the low level by a small FSM which manages the operations to be executed by the DSP according to the task control sequence (which in the following will be addressed as OPCODE) received, which will be optimized in a later stage.
In Figure 4.4 a high-level representation of the described PFU architecture is reported.

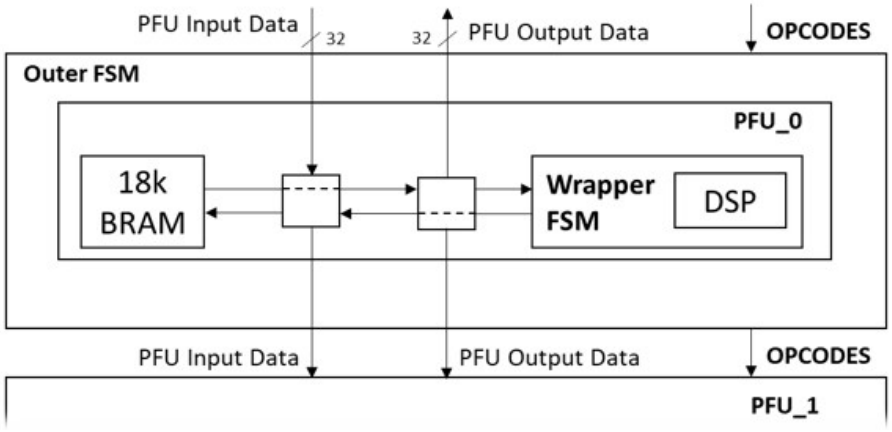

Figure 4.4: High Level PFU Preliminary Architecture

In detail, the PFU receives 32-bit wide inputs from outside and produces an output of the same width. The PFU inputs and output signals can be acquired form the PFU itself or "forwarded" to the next one. In fact, the preliminary PFU design has been realized to enable the interconnection and cascading of several PFUs to form a cluster.

The main effort has been focused in obtaining a baseline structure to analyze and validate the architectural approach. Once such baseline has been consolidated, it has been used as foundation to perform further and focused optimizations to increase the programmable unit flexibility, while minting resource usage controlled.

Concerning the PFU structure, five main sub components can be considered: they will be discussed in detail below.

Finite State Machine

Each PFU is equipped with two FSMs: a parent FSM that handles high-level contacts with the outside world by sorting data to and from the BRAM or the next PFU, and a smaller FSM identified in Figure 4.4 as the `Wrapper FSM` that is responsible for executing the algorithm. The FSM is not programmable at this stage because it is specifically made for the DSP and operation management. This is one of the most important components that will be developed and presented in the final version in subsection 4.3.1.

Because all PFUs in the cluster conduct the same functions, in addition to controlling programmability, it is also important to consider limiting the number of FSMs within the same cluster in order to reduce resource utilization as much as possible.

Computing Core

The computing core present in each PFU is the DSP unit described in detail in section 3.1. It was used in the SIMD configuration as can also be seen from the high-level diagram shown in Figure 4.8. In this way the maximum computing capability is guaranteed, allowing to process 4 data sets in parallel exploiting the low parallelism of the data being worked on.
As the description of the SIMD mode in subsection 3.1.2 has shown, the use of this mode compromises the use of one of the input ports of the DSP (i.e., port *D*). However, this does not introduce performance degradation for the execution of required operations for the selected testbed as it does not require calculations between more than two operating per time. On the other hand, by taking advantage of the programmability of the envisioned architecture, all ports can be re-titled if needed for other applications.
The remaining input ports of the DSP have been suitably mapped to receive the necessary input data coming from the BRAM or from the temporary support registers inside the PFU using suitable multiplexers. Similarly, the output port is managed in order to allow the storage of the results in the local memory or in the respective internal registers.
Finally, the DSP operations useful for the realization of the testbed are sums and subtractions suitably mapped according to the algorithm described in section 3.3 that meet the technical specifications in subsection 4.1.1.

Local Memory

Each PFU includes a dedicated local memory, which similarly to the DPS unit is implemented relying on a ultrascale primitive hard macro. In detail, the RAMB18E2 is used [24].
The high regularity and organization provided by the FPGA fabric between DSP Slices and BRAM Slices enable to couple to each DSP one BRAM instantiated as 18 kbit RAM as discussed in subsection 2.3.1.
The RAMB18E2 is explicitly instantiated in the Hardware Description Language

(HDL) as a Simple Dual Port RAM supporting 32-bit data width and up to 512 storage depth. In this way, each PFU local memory has a pair of ports allowing independent reading and writing operations.
In fact, with two independent ports it will be possible to optimize the order of execution, starting data processing while completing last data storage. Since only one port for each operation is present, the possibility of writing/reading from the outside or inside the PFU must be managed, a task that is devoted to the FSM which controls the operations.
To efficiently manage this data transfer, both the interfaces of the memory have been customised rather than depending on the supplier's memory controllers, with the goal of maximizing performance as much as possible. The standard interfaces for accessing the BRAM, in fact, are AXI-type interfaces (i.e., a standardised communication protocol for data transfer), which are very useful since they allow several blocks to be interfaced with each other, but they have significant latencies, so it was decided to work with a custom interface to maximise communication delays.

Ancillary Logic Components

As previously stated, each PFU inside is equipped with custom logic in order to maximize performance. By using appropriate support registers, in fact, it is possible to minimize writing and reading latencies in the BRAM, managing to use the DSP. However, a trade-off have been considered to achieve lower resource usage for support registers and lower latencies.
Furthermore, through the use of sorting components (i.e., multiplexers) it is possible to supply the computing units with the necessary data. As anticipated, in fact, each PFU has a single DSP that have to perform different operations, consequently it is necessary to provide it with the necessary data for each iteration.
All custom components inside the PFU are managed by the cluster FSM. This means that the enable of the registers and selectors of the multiplexers are suitably controlled by an additional custom logic available for the whole cluster capable of carrying out the operations required by the application.

Input & Output PFU Interfaces

Each PFU shall enable the communication with the outside. In particular, the PFU shall be capable or receiving input data as well as providing the output of its computation. Furthermore, it will be necessary to provide to each cluster the configuration words to implement the desired programmability as depicted in subsection 4.3.3. Finally, it will be necessary to provide the clock and reset respectively to synchronize the operations and to provide system initialization as depicted in the previous subsection paragraph. The main input and output interface are shown in Figure 4.5.

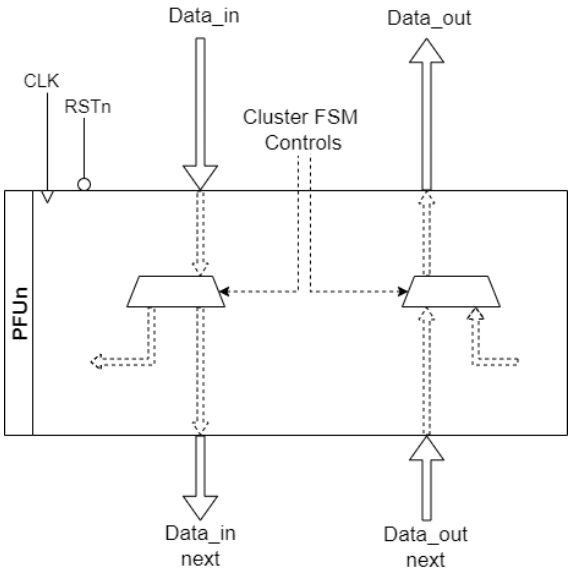

Figure 4.5: PFU Input & Output Interfaces

The data transmission management and storage protocols are strictly linked to the cluster organization described at the beginning of subsection 4.3.1. In detail, input data arrive within a serial data protocol which requires longer times for data sending but allows to reduce the parallelism of the communication buses.
Once memory have been loaded with input serial data, the Cluster FSM will connect the input bus indicated in Figure 4.5 as `Data_in` with the input bus of the next PFU named `Data_in_next`.

4.2.3 Preliminary Design Evaluation

This first benchmark allows to verify the key concepts presented in subsection 4.1.1. In particular, different architectural strategies were investigated allowing to identify limitations and potentiality that the initial design presents.
The main ones are analyzed in detail below:

i. It has been possible to identify and investigate the DSP potentiality mainly devoted to testbed operations. Moreover, it has been extended the know-how to other operations as well, since system programmability is one of the key aspects of the project.
   Possible operations have been evaluated to be carried out in SIMD mode, evaluating whether and which ones were sufficient for the realization of the testbed and how to dynamically manage the control signals of the DSP to exploit programmability. In detail, the main operations for the realization of the testbed are additions, subtractions, absolute value and minimum value. It has been seen that all these operations can be carried out through appropriate configurations of the DSP.
   However, a first limitation is the impossibility of carrying out operations beyond simple additions and subtractions when working in SIMD unless re-configuring the DSP. Furthermore, for the execution of the testbed it is necessary to calculate the minimum value between five data and this is not possible, the strategy is to calculate iteratively the minimum value in pairs of two, paying higher latency.
   Finally, for absolute value calculations it is necessary to perform two operations with the same data in order to estimate the sign between the two difference (i.e., performing `0-A` and `A-0` and then evaluating the sign the absolute value is founded). This means that even working in SIMD 4x12 it is possible to calculate the absolute value of only two data at a time. However, also in this case it has been considered that the latency introduced with respect to the increase of resources necessary in the case of implementation in sparse logic is a good compromise.

ii. It has been possible to couple the two key blocks (i.e., DSP and BRAM) as will be deepened in subsection 4.3.2 by identifying the potentiality and bottlenecks that this combination presents and how exploit the interaction between these two blocks maximizing performance and minimizing the use of resources.
   In particular, the coupling alternatives between DSP and BRAM were eval-

uated considering the latencies introduced with a intensive use of memory or, on the other hand, the high use of resources in the case of intensive use of temporary resources such as registers.
In detail, the use of temporary registers makes it possible to optimize processing times while the intensive use of memory requires times for read and write accesses, increasing processing times. The envisioned approach aims to obtain a trade-off between these two aspects using the minimum amount of resources sufficient to execute the operations with the minimum increase in terms of execution time.

iii. It has been possible to identify the possibilities of implementing operations with the use of custom logic and which through the use of the DSP, in detail it was assessed that the use of custom logic for arithmetic operations did not allow to obtain higher performance while introducing a considerable overhead of resources. For this reason the architectural choices have moved towards the intensive use of the DSP for arithmetic operations leaving the custom logic only for the boundary operations such as multiplexer and registers.
In detail, as previously anticipated, it has been opted for the use of the DSP by exploiting the elementary operations to carry out all the steps of the algorithm selected as testbed.

iv. It has been possible to consolidate the key concepts of the clustered approach identifying blocking points of the strategy. In this view, it has been noted that it was necessary to use a grouping not only in terms of the arrangement of the PFUs but also at a functional level, allowing to have a single FSM for the cluster in order to minimize the resources utilization.
More in detail, the use of a shared FSM between the eight PFUs of the cluster allows to share the operations limiting the programmability of the system but allowing to have an optimization of the consumption of resources. In this view, the penalty to pay for programmability is not very relevant as it would also be difficult to manage a large number of PFUs with such detailed granularity.

v. Finally, it has been possible to identify the reading and writing strategies for input and output data in order to correctly synchronize the execution of the operations.
In particular, it was tested the envisioned feeding system and the relative output data reading through the serial interface which will be analyzed

more in detail in subsection 4.3.1 allowing to have a continuous flow of input and output data and so that the PFUs work in pipeline.
Furthermore, through the first implementations it was possible to test the switch mechanism of the input bus and the relative output bus which are suitably connected to the corresponding PFU according to the data being received.

These considerations allowed to continue by incremental steps towards the final structure, refining the envisioned architecture step by step.

4.3 Final Design Overview

The primary principles that have been applied in the design of the final PFU will be outlined in this section, as it will presents the changes from the previous version and the opportunities.
As anticipate, the Programmable Functional Units (PFUs) represent the main computing resources to accelerate Digital Signal Processing tasks and it is envisioned to be wrapped in a high-level programmable shell, capable to efficiently provide the settings for the inner DSP, BRAM and ancillary components.
This control shell is envisioned to be used to manage a cluster of PFUs as will be described in subsection 4.3.1 in opposite with the preliminary design described in section 4.2 where each Functional Unit has its own hard-coded Finite State Machine.

Subsequently, in subsection 4.3.3, the proposed approach for exploiting programmability will be detailed.
In this view, a trade off has been considered to obtain a sufficient granularity without introducing an overhead in terms of minimum programmable and reconfigurable area. In the proposed structure this minimum area corresponds to a cluster, composed of 8 PFUs. This choice was made considering that a too large (i.e., all the accelerators array) reprogrammable structure could be disadvantageous because it would not allow for sufficient flexibility in terms of programmability and reconfiguration time, on the other hand a structure with an excessively fine granularity (i.e., a single PFU) could instead introduce a very large overhead in terms of management.

More in detail, the updated clustered structure will be described in subsection 4.3.1, namely, the control unit will be shared for the cluster's functional units, as anticipated.

The internal structure of the PFU will next be described in detail in subsection 4.3.2, showing which main resources are employed.

The primary configurations and structure of the instruction supplied to the PFU to accomplish programmability will thereafter be presented in subsection 4.3.3.

Finally, in section 4.4, the algorithm's mapping onto the implemented structure will be explained.

4.3.1 Clustering Approach

The cluster architecture has been realized in order to maximize computation throughput while minimizing latencies. The involved latencies are mainly related to the reading and writing of the input data. In fact, using the proposed structure, it is possible to reduce data transfer latency and ensure that performance mostly depends on data processing rather than transfer operation.
As an example, for data transfer of one of the tasks for the execution of the algorithm selected as testbed, e.g. for the 3-circulants task, there will be 6 clock cycles of latency and approximately 50 clock cycles for computation. Consequently, every 6 clock cycles a new PFU will start processing and every 6 cycles thereafter will provide its output results.
The envisioned structure is reported in Figure 4.6.

In this structure, the data arrive on a 32-bit bus named `DATA_in` and are stored in one PFU after another.
In detail, the incoming data are the gamma and beta values for the LDPC algorithm decoding that must be processed, the first twelve or six values (depending on the number of circulants within the processed layer) are stored in the BRAM within the first PFU and subsequently through the block `A` will switch connecting the input bus with the second PFU, and so on. In this way it is possible to reach all the PFUs with a limited fanout and avoiding having a high latency. As a large number of PFUs is present, in fact, it would not have been possible to reach each of them individually as, assuming 128 PFUs and having to transfer 6 data each (e.g. in the case of task 2 for the testbed algorithm), the last functional unit would have had a latency of $128 \cdot 6$ clock cycles.

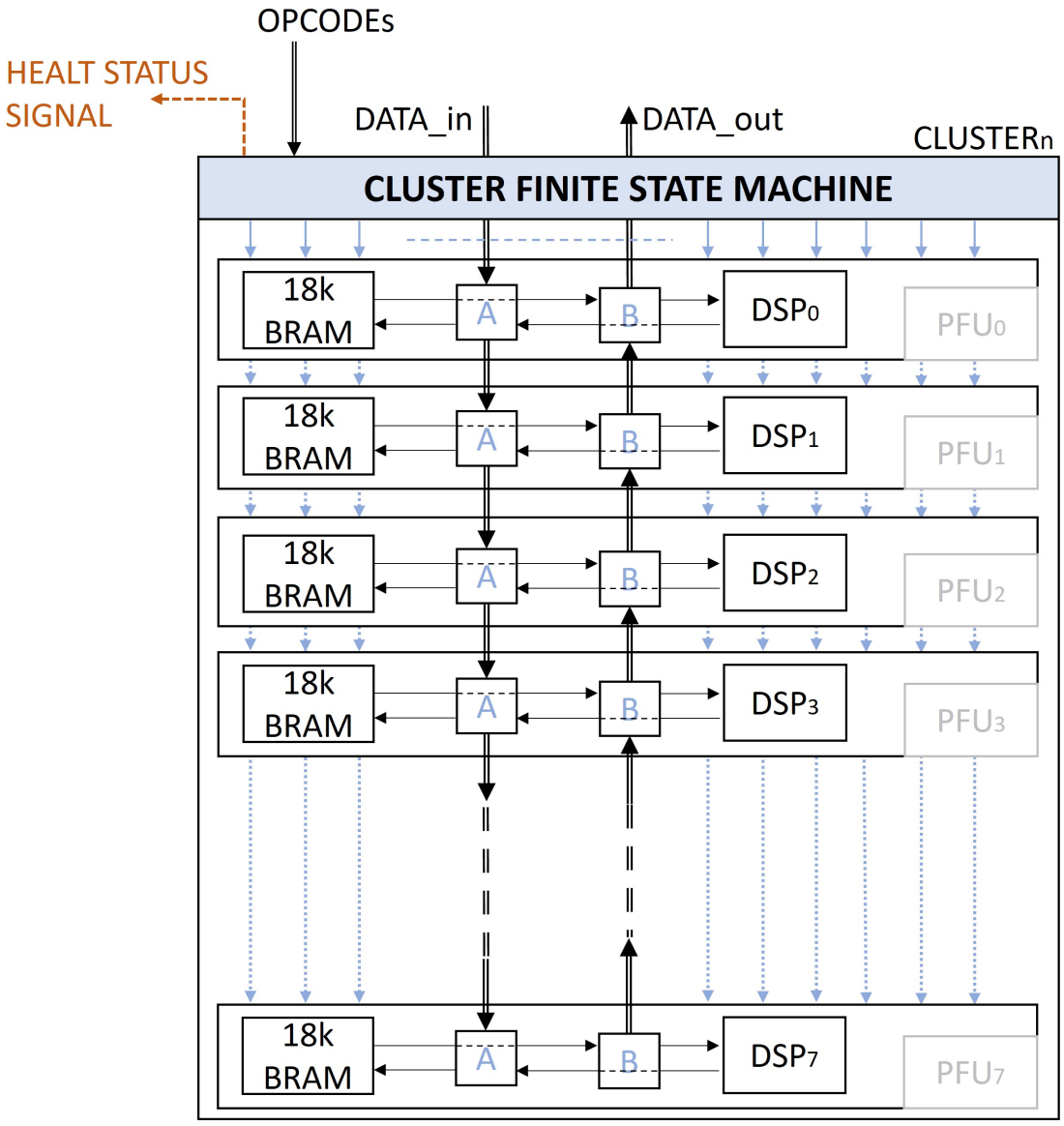

Figure 4.6: High Level Cluster Block Diagram

After having communicated all the real data to the current cluster, data will be sent to the second cluster which are part of the block called `CLUSTER_GROUP` in Figure 4.3.
In particular, the sixteen groups of clusters will be filled in a serial way, one after the other, and each cluster will be managed as just described. On the other hand, the 16 groups will work in parallel and therefore the overall latency is equal to that

of the single group.
Similarly, block `B` is used for reading the data produced by the PFUs and one after the other connects the output of the PFUs to the output bus of the cluster named as `DATA_out`.
The control signals produced by the cluster's Finite State Machine are suitably multiplexed to all the Functional Units of the cluster with the aim of performing the operations with the correct timing. In detail, as data is delayed by a number of clock cycles equal to the number of gamma values (i.e., 6 or 3 depending on the number of circulants within the processed layer), the instructions will also be delayed the same clock cycles. In this way, the processing of each PFU occurs concurrently with the start of data transmission.

Scalability Assessment

The main goal of the analysis presented in the following is aimed at identifying an architectural strategy capable of providing a solution able to maximize the portability and scalability. In fact, talking more specifically of scalability, the proposed architecture embeds all the necessary components and computing feature to be used as overlay concept. Differently from monolithic RTL approaches, this allows to potentially extend the proposed solution both in terms of devices and number of programmable accelerators to deploy different and bigger applications without the needs to rethink the whole architecture, with the relative mitigation approaches and providing opportunities for portability.
In particular, the cluster described at the beginning of subsection 4.3.1 is the elementary block on which the system is based. More in detail, groups of 2 clusters form what is called `CLUSTER GROUP` and which is indicated in Figure 4.7. Subsequently, the group described is replicated several times (16 times for the testbed) to obtain the final structure composed of a certain number of PFUs (256 PFUs for the testbed purpose). Thus, the architecture obtained allows to process a layer of the H matrix of the testbed in SIMD mode as will be depicted in section 4.4.
The key point that allow to have such a scalable structure is the use of a 512-bit data bus which is divided into 16 sections of 32 as depicted in Figure 4.7.
From Figure 4.7 can be noticed how expanding the input feeding data bus it is possible to replicate different times the same structure obtaining different subsystems working on a different data bus portion being completely parallel and independent. As a result, the potential computing performance is `n` times higher then a single branch, where `n` in the shown scenario is 16 since the 512-bit input bus was designed

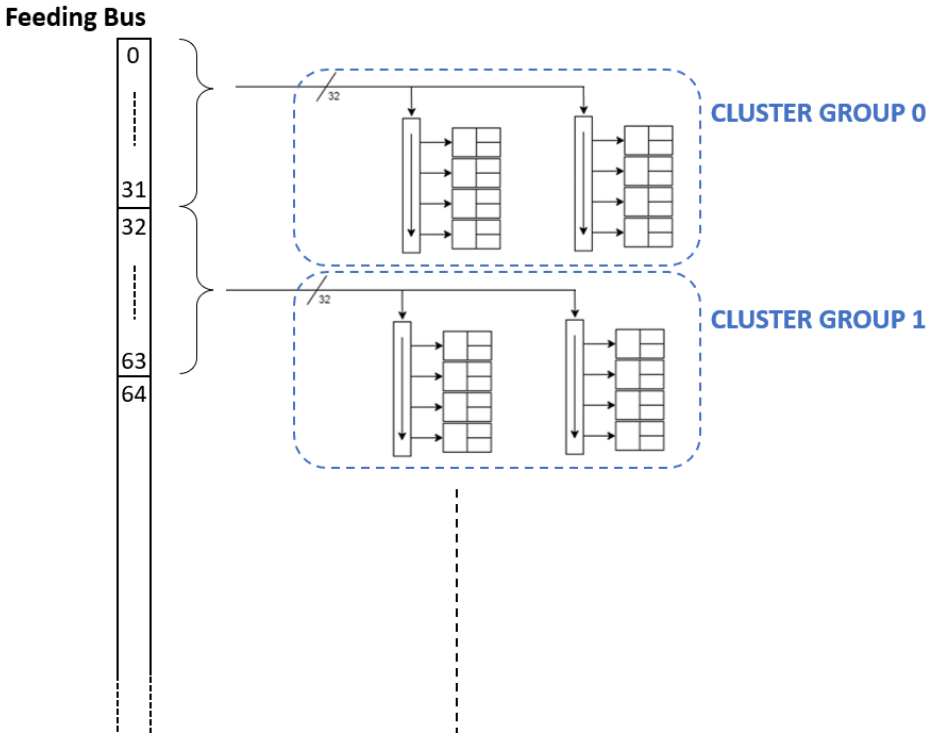

Figure 4.7: Clusters Scalability

to support 16 clusters.
For the system output, everything works in a specular way, the only difference is due to the fact that while at the input, as seen from Figure 4.7, the clusters are connected to the same bus obtaining a fanout 2, at the output this is not possible and shall be used a different mechanism. In detail, the output data of the clusters will be sent to a multiplexer and its output will then be sent to a data bus organized exactly like the feeding bus that can be seen in the previous figure.

Commercial Architectures Comparison

The higher-level aspect of the clustered structure of several nodes anticipated in section 2.1 is another key element that may help to estimate the system scalability. The proposed architecture, in particular, is very similar to the conventional structure employed by current Graphics Processing Units (GPUs), which are built on a

system of accelerators also known as Processing Elements (PEs) that are responsible for supporting the compute intensive part of the calculation. GPUs, in fact, are primarily intended to achieve fast throughput and contain a huge number of computing units, making them ideal for compute-intensive applications that require high speed [30].
Due to its array structure, FPGAs, on the other hand, provide a great level of flexibility and programmability. In addiction, the architecture proposed in this thesis makes it feasible to take use of the accelerator array's programmability and flexibility due to the specific characteristics that have been anticipated and are detailed in depth in the subsequent sections.
In Table 4.1, a high-level comparison that allows an initial assessment in terms of common characteristics such as operating frequency, operations performed per second, power consumption, etc... This also allow a primary estimation of the main feature comparing the XCKU040 FPGA [31][32] chosen as reference for the architecture testbed, and the GeForce-RTX-3070 [33], which is one among the latest GPUs on the market.

	RTX 3070	**Current Architecture**
Working Frequency [MHz]	High (≈ 1500)	Medium (≈ 250)
Number of PEs	High (5888)	Medium-Low (128-256)
Peak Throughput	High (≈ 20 TOPps)	Medium (≈ 123 GOPps - 246 GOPps)
Power Consumption[W]	High (≈ 220)	Low (≈ 13)
Memory Size [GB]	Medium (≈ 8)	Medium (≈ 2)
Radiation Robustness	No	Yes
Updates Availability	No	Yes
Cost [$]	Medium-Low (≈ 600)	Medium-High (≈ 2-3 K)

Table 4.1: Comparison Between Current Design Architecture and Commercial GPU

It should be observed that, in order to offer a fair comparison, the suggested architecture's performance has been expressed in operations per second. To accomplish so, the activities of *storing and reading, adding, subtracting, calculating the minimum and absolute value* described in section 3.3 have been taken into consideration; moreover, all arithmetic operations have been multiplied by 4 owing to the usage of the SIMD mode described in subsection 3.1.2. This resulted in a total of 192 operations, which were completed in 50 clock cycles. With a working frequency of 250 MHz, 192 operations will be completed in around 200 ns. The amount of operations per second of each individual PFU may be calculated using a simple proportion, the obtained value has been then multiplied by the node's PFUs and reported in Table 4.1.

As can be seen in this preliminary comparison, the structure performs less well in in the case of a single node, but it supports applications in space field and allows the FPGA structure's high flexibility to be leveraged.
Furthermore, if the study is extended to a set of four nodes, as anticipated, a more balanced comparison may be produced.

	RTX 3070	**Current Architecture**
Working Frequency [MHz]	High (≈ 1500)	Medium (≈ 250)
Number of PEs	High (5888)	Medium (512-1024)
Peak Throughput	High (≈ 20 TOPps)	Medium-High (≈ 492 GOPps - 984 GOPps)
Power Consumption[W]	High (≈ 220)	Medium (≈ 52)
Memory Size [GB]	Medium (≈ 8)	Medium (≈ 8)
Radiation Robustness	No	Yes
Updates Availability	No	Yes
Cost [$]	Medium-Low (≈ 600)	High (≈ 8-12 K)

Table 4.2: Comparison Between Tile Architecture and Commercial GPU

As can be observed in Table 4.2, the suggested design's high scalability capabilities allow performance to be extended to equal the order of magnitude of the calculation capabilities of standard graphics computation devices. This result is clearly preliminary, and all data reported must be considered as order-of-magnitude estimations; however, it is clear that it is feasible to create a structure that is as large and high performing as needed by increasing the number of nodes as needed. While the number of GPU accelerators is significant but constant, in this architecture it may be increased or decreased by taking use of the idea of scalability.
However, the development of a system of this type requires a higher cost due to the components used and the techniques needed to allow the system deployment in space application. However, another very important aspect for space applications can be noted, namely power consumption. One of key aspects in space is in fact keeping power consumption under control, and this is also one of the aspects for which common devices such as GPUs are not widely used in space environment.

4.3.2 Final PFU Design

The Programmable Functional Units (PFUs) envisioned architecture is based on two main elements: a DSP unit and a BRAM. Furthermore, custom logic will also be included allowing from one hand to have temporary registers or multiplexers to increase the functional unit flexibility and, on the other hand, to obtain the logic connections useful to implement a dedicated Finite State Machine (FSM) able to manage the PFU operations.
However, the FSM will not be included in each PFU as this would drastically increase the resources usage, thus it will be shared among a cluster, as discussed in subsection 4.3.1.
In fact, different considerations were taken into account in designing the PFU architecture:

i. First, the Node contains different blocks (e.g., microprocessors, the health checking system, the block for external communication) which will use a certain amount of resources on the device. The first analysis was therefore to estimate how many resources are available for the accelerator array net of those used for system management.

ii. Secondly, a trade-off between the use of custom logic (e.g., support temporary registers or multiplexers) and a BRAM based structure (i.e., less use of temporary registers at the expense of greater latency) were analyzed.

From one side, an higher customization increase computation flexibility and efficacy at the cost of an increased amount of resources. On the other hand, it produces a lower programmability feature than just using memory as storing point, also requiring a greater amount of resources. These analyzes made possible to obtain a structure that reflects the programmability project requirements without compromising computational performances.
The high level diagram representing the PFU structure which operate complying with the cluster FSM is shown in Figure 4.8 with the highlight of the SIMD mode anticipated in previous sections.

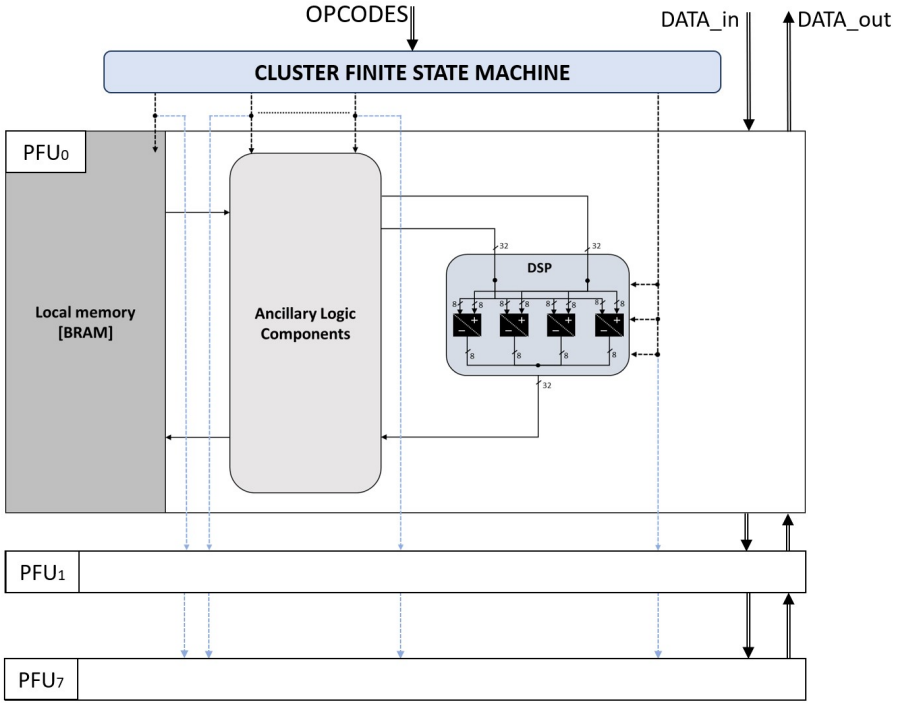

Figure 4.8: PFU High Level Block Diagram

As can be seen from Figure 4.8, the two macro-blocks mentioned and also the custom logic such as registers or multiplexers are present allowing to perform required operations. The FSM, in turn, instruct the PFUs accordingly to the received OPCODE sequence ordering corresponding actions to the computing cores. The OPCODE is a configuration word designed to allow the functional unit programmability. Through this sequence of bits, in fact, it will be possible to request the execution of different operations and to enable the needed resources behaviour. It will be further discussed in subsection 4.3.3.

4.3.3 Instruction Set

The current section provides an overview of the instruction set that allows PFUs to perform a specific set of operations. In fact, the PFU either as single element or relying on cascading properties, is envisioned to support a set of operations consisting of the foundation for Telecom Functionalities, and more in details for Digital Signal Processing.
In Figure 4.9 an high level representation of the Instruction format is reported. In general, the format of the instruction set is contrived to support the configuration of:

- Input/Output data management controls, referred as to `IO Ctrl`
- DSP primitive management, referred as to `"DSP Ctrl"`;
- BRAM primitive management, referred as to `"BRAM Ctrl"`;
- Ancillary Wrapper components (e.g., registers, Mux/Demux, etc.), referred as to `"A Ctrl"`.

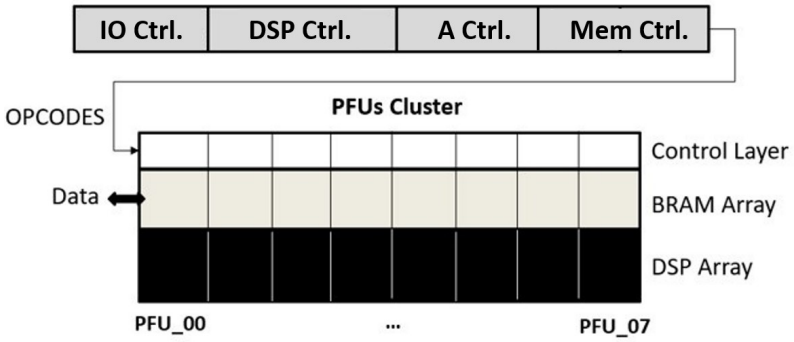

Figure 4.9: Instruction Set Format

The envisioned strategy is based on the development of different tasks that will be defined in advance and appropriately programmed through the selection by the application. For the management of the algorithm selected as testbed there will be mainly two tasks necessary to carry out the operations for three or six circulants in accordance with what has been described in section 1.4.
More in detail, a library has been created containing a package where tasks are defined and, subsequently, depending on the task selected and the current iteration managed from the Cluster FSM, the correct sequence will be supplied to the PFU.

In Appendix A it is possible to see an example of a sequence of instructions for the implementation of the task related to the operations for carrying out a layer of the LDPC algorithm with 3 circulants.
The fundamental operations as well as the combinations thereof that can be used for more complex tasks (i.e., the calculation of the minimum or of the absolute value) can be suitably coded and inserted in the package at the time of programming.
In this way, it is possible to exploit the programmability of the system and it is also possible to program each cluster of 8 PFUs in such a way to perform a different task, expanding the potential of the system as much as possible.
In the next section the main operations required for the realization of one of the two tasks for the realization of the LDPC algorithm will be presented.

4.4 LDPC Min-Sum Algorithm Mapping

In this section it will be shown how it is possible to use the structure created to map the algorithm chosen as a testbed, in detail the main steps will be shown and the chosen configurations explained.
The operations required to perform the Layered LDPC Min-Sum Algorithm decoding kernel (i.e., VN Processing, CN Processing and AP Updated) to process one row of the H-matrix are the ones explained in subsection 3.3.1.
The processing of each row of the H-matrix can be performed in parallel, using the SIMD DSP feature presented in subsection 3.1.2 and the array of accelerators depicted in subsection 4.2.1.
Therefore, to deploy the target telecom testbed, each PFU will perform four sets of operands concurrently.
Nominally, the operation to perform can be dived into the following sets of operations:

- **VN-Processing:** a series of three or six subtractions (depending on the number of circulants within the layer) to compute the α values.
  In detail, this operations are carried out on the basis of gamma values coming from outside and beta ones stored within the local memory. Consequently, for the calculation of alpha it is necessary to perform 2 accesses in memory, one to read the gamma value and one for the beta value which will be both stored on two temporary registers. Subsequently, the alpha results are stored both in local memory and in temporary registers. This

is because on the one hand it is useful to keep them in support registers to avoid additional latencies for the following gamma calculation. On the other hand, for the calculation of the absolute value it may be useful to have the values stored in memory avoiding higher combination of registers selection and consequently a lower use of multiplexers.
As an example, in Table 4.3 can be noticed the steps for three circulants computations related to the alpha values. .

Clock Cycle	**Store Operations (from outside)**	**Store Operations (from PFU)**	**Read Operations**	**Computations**
1	Storing $\gamma_{0\div3}$		Reading $\beta_{0\div3}$	
2	Storing $\gamma_{4\div7}$		Reading $\gamma_{0\div3}$	
3	Storing $\gamma_{8\div11}$		Reading $\beta_{4\div7}$	Computing $\alpha_{0\div3}$
4		Storing $\alpha_{0\div3}$	Reading $\gamma_{4\div7}$	
5			Reading $\beta_{8\div11}$	Computing $\alpha_{4\div7}$
6		Storing $\alpha_{4\div7}$	Reading $\gamma_{8\div11}$	
7				Computing $\alpha_{8\div11}$
8		Storing $\alpha_{8\div11}$		

Table 4.3: Computing α Sequences

- **CN-Processing:** a series of three or six β computations (depending on the number of circulants within the layer), which are obtained from the concatenation between the product of the signs and the absolute minimum of the involved α.
  In detail, the beta computations are mainly divided into two phases, first alpha absolute values are evaluated, then the minimum among the absolute is calculated. More specifically, as anticipated in subsection 4.2.3 for both these operations it was decided to use the DSP which, despite increasing the latency due to the computation of the different operations, allows to save considerably in terms of resources.
  The evaluated absolute value will be stored in memory to minimize the use of temporary registers. Subsequently, the minimum value will be evaluated, in detail as has been anticipated, in the case of six circulants machines the latency increases drastically as the minimum must be computed in pairs and therefore for each beta value it is necessary to perform 4 iterations. Finally, the minimum value is concatenated with the signs product and

also stored in locale memory, in particular the beta value will be kept locally for the next iteration.
As an example, in Table 4.4 can be noticed the steps for three circulants computations related to the beta values. .

Clock Cycle	**Store Operations (from outside)**	**Store Operations (from PFU)**	**Read Operations**	**Computations**
3	Storing $\gamma_{8\div11}$		Reading $\text{beta}_{2\div3}$	Computing $\alpha_{0\div3}$
4		Storing $\alpha_{0\div3}$	Reading $\gamma_{4\div7}$	Computing $\text{absolute}_{0\div1}$
5			Reading $\beta_{8\div11}$	Computing $\alpha_{4\div7}$
6		Storing $\alpha_{4\div7}$	Reading $\gamma_{8\div11}$	Computing $\text{absolute}_{2\div3}$
7		Storing $\text{absolute}_{0\div1}$		Computing $\alpha_{8\div11}$
8		Storing $\alpha_{8\div11}$		Computing $\text{absolute}_{4\div5}$
9				Computing $\text{absolute}_{6\div7}$
10		Storing $\text{absolute}_{2\div3}$		Computing $\text{absolute}_{8\div9}$
11				Computing $\text{absolute}_{10\div11}$
12		Storing $\text{absolute}_{4\div5}$		Computing $\beta_{0\div3}$
13		Storing $\beta_{0\div3}$		Computing $\beta_{4\div7}$
14		Storing $\beta_{4\div7}$		Computing $\beta_{8\div11}$
15		Storing $\beta_{8\div11}$		

Table 4.4: Computing β Sequences

- **AP-Update:** a series of sum between the computed α and β to evaluate the new γ values.
  In detail, the operations required for the gamma calculation are simple sums between beta and alpha. Therefore, each step requires to read previously calculated beta value from the memory and add it to the alpha value which is stored in the temporary registers. Finally, the gamma value just calculated is stored in memory and subsequently sent out through the dedicated bus.
  As an example, in Table 4.5 can be noticed the steps for three circulants computations related to the beta values.

Clock Cycle	**Store Operations (from outside)**	**Store Operations (from PFU)**	**Read Operations**	**Computations**
15		Storing $\beta_{8\div11}$		Computing $\gamma_{0\div3}$
16		Storing $\gamma_{0\div3}$		Computing $\gamma_{4\div7}$
17		Storing $\gamma_{4\div7}$		Computing $\gamma_{8\div11}$
18		Storing $\gamma_{8\div11}$		

Table 4.5: Computing γ Sequences

From the operations shown in these tables it can be noticed how thanks to the use of the DSP unit it is possible to perform the decoding operations required by the testbed. Furthermore, through an efficient timing scheduling of the operations it can be seen how the decoding sequences have been minimized as much as possible. In fact, a first assessment was also carried out on the design achievable performance, and will be discussed in detail in subsection 5.2.3.

5
Results

This chapter will show the main analyzes and tests carried out to demonstrate the correct functioning of the design and to verify the performance and potentiality of the system.

The Xilinx Kintex Ultrascale XCKU040 [31][32] has been chosen for the tests as it will also be the board used for the radiation tests once the design will be finalized. In particular, the XCKU040 component was chosen owing to the availability of the development board, making it a valid solution for the deployment of the intended testbed implementation. It's important to note, though, that the COTS XCKU060 [32] device is also available. It uses the same technology and core design as the XCKU040, allowing for easy comparison and mobility. When comparing them, it is easy to see that the resource count is somewhat lower. Following a preliminary investigation, it was determined that the resources provided in the 040 version were sufficient for an initial assessment of the design, which is why it was chosen.
The tools used for this development are provided by Xilinx and are:

- Vivado 2021.1
- Vitis 2021.1
- Xilinx Power Estimator

In detail, the flow followed for the verification and for the realization of the setup created for the deployment of the tests will is firstly reported.

Secondly, the analysis carried out regarding the resources utilization will be presented with the relative comparison between the two designs discussed in chapter chapter 4.

Subsequently, an analysis will be shown comparing the present proposal performing the algorithm discussed in section 3.3 used as a testbed with a commercial IP-CORE that performs the same function in terms mainly of resources and performance.

Finally, the actual functioning of the structure will be demonstrated through simulations and tests carried out on board tests.

It is important to note that in most of the analyses that follow, a comparison will be done between the two designs outlined in chapter 4. Because the initial version of the PFU was not small enough to be replicated the required number of times, as will be addressed in detail in subsection 5.2.1, a limited number of PFUs were evaluated in both situations to create a fair comparison (i.e. 128 PFUs for a total of 16 clusters).

5.1 Methodology

The flow of analysis and verification followed consists of verifying the feasibility possibilities of the design and subsequently an implementation on board for functional and performance verification.
The methods used to validate the design are depicted in the flowchart assessment in Figure 5.1. As can be seen, after having finished the architecture's design, the capability to integrate the array of accelerators on the board was verified through the resource occupation analysis. Various portions of the design had to be reconsidered during this phase in order to accommodate the required number of accelerators within each board.

At the same time, the functionality of the model was validated using testbench and simulations.

Subsequently, power and performance analysis was carried out in order to verify the system's capabilities, both in terms of power consumption and bandwidth. Concluding the analysis point of view with a comparison was made between the solution described in this thesis and a commercial device, which will be addressed later, to verify the capabilities of the envisioned architecture and confirm the tradeoffs between programmability and growing occupancy.

Parallel to this, necessary data for on-board testing was obtained, and an excel model was developed to provide a gold standard for measuring the accelerators functionalities. Specifically, a collection of input data representing the input samples

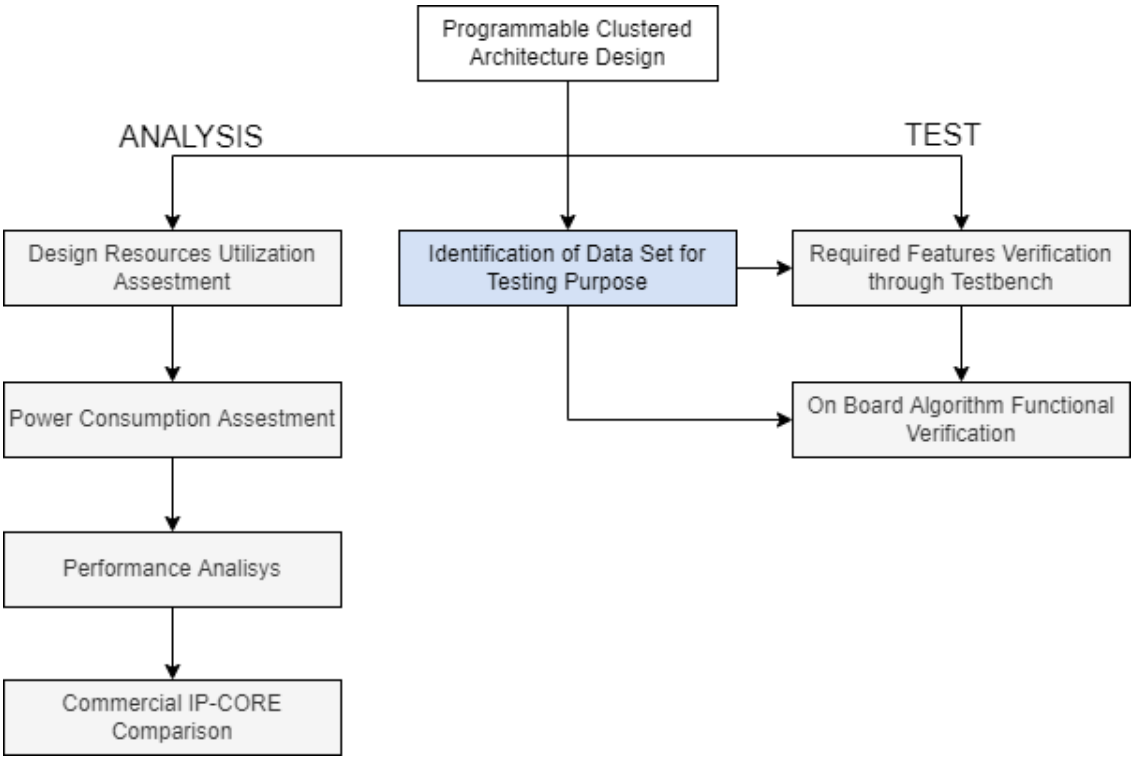

Figure 5.1: Results Analysis and Tests Methodology

for the LDPC decoding method defined in subsection 3.3.1 during this phase has been generated.

Following that, the hardware design was functionally tested to check on-board behaviour and validate the design; the output bits were generated starting from input samples, and the architecture was thus consolidated.

5.2 Analysis

Some of the main analyses carried out to evaluate the intended structure will be described in this section, with the goal of reporting the factors that have allowed to fine-tune the project.
The resource analysis will be presented in depth first, with a comparison between the two primary variants of the accelerator array discussed in the previous chapter. Following that, a comparison with a commercial IP-CORE will be done in order to

demonstrate the project's capabilities and how they might be achieved.
The power consumption study will then be explained, with the help of the Xilinx Power Estimator (XPE) tool, which allows an estimate of the static and dynamic power that a section of the FPGA will require based on the resources utilization.
Finally, a performance study will be provided in two primary scenarios: absolute performance and performance per Mb.

5.2.1 Resources Utilization

This subsection will present the results of the resource utilization analysis. In fact, an assessment of the resources utilised for accelerators array was done based on the preliminary partial designs acquired.
This analysis was originally performed in numerical form, resulting in the deployment of a cluster and then assuming a linear trend. Then, a confidence margin has been added to the obtained results due to the preliminary nature of the utilised design. Considered resources for the analysis are:

- CLB LUTs
- CLB Registers
- BRAM
- DSP

The analysis was conducted employing Vivado 2021.1, which was deployed to develop and implement the design as well as collect resource use data using the `"report utilization"` function. The current study was conducted by comparing the two PFU variants reported in the previous chapter. The analysis findings of the structure referred to as the first design, in which each PFU has its own FSM, are reported in Table 5.1. In detail, the use in absolute terms and the percentage occupancy on the target board will be shown for each component reported.
Because the computing structure is replicated many times within the cluster, the PFU structure will be visibly more sophisticated in this situation, resulting in a higher resource usage.

	CLB LUTs	CLB Registers	BRAMs	DSPs
1 Cluster	≈ 5%	≈ 1%	≈ 1%	≈ 1%
16 Clusters	≈ 85%	≈ 25%	≈ 10%	≈ 5%
16 Clusters x1.25 margin	≈110 %	≈ 30%	≈ 15%	≈ 10%

Table 5.1: Resources Utilization First Design

Similarly, data from the analysis performed with the second design has been provided in Table 5.2, resulting in a common FSM for each PFU within a cluster. The duplicated structure in this scenario will just be that of the PFU without the computational component; as a result, each PFU will process distinct data, but the controls will not be replicated; as a consequence, a lower resource usage is expected than in the prior example.

	CLB LUTs	CLB Registers	BRAMs	DSPs
1 Cluster	≈ 1%	≈ 1%	≈ 1%	≈ 1%
16 Clusters	≈ 30%	≈ 20%	≈ 10%	≈ 5%
16 Clusters x1.25 margin	≈ **40%**	≈ **25%**	≈ **15%**	≈ **10%**

Table 5.2: Resources Utilization Final Design

As anticipated, a multiplier factor of 1.25 is used to the final resource consumption, although this structure ensures that the number of resources used is sufficient for implementation on a board, it remains a conceptual analysis since it is linear extended to the entire PFU array.
Furthermore, as predicted, the improvements outlined in the preceding chapter about the paradigm change with which the FSM is employed result in a significant improvement in terms of resources, which in the case of LUTs, for example, results to be less than half.
A bar graph has been provided that displays the trend of the key resources under

investigation in order to offer a visual representation of the data presented in Table 5.1 and Table 5.2. Only the resources relating to LUTs and Registers have been reported for the sake of simplicity because, as can be seen from previous tables, the data relating to BRAMs and PFUs do not change in the two designs because the internal logic of the PFU has not changed in terms of storage memory (i.e., the BRAM) and computing units (i.e., the DSP).

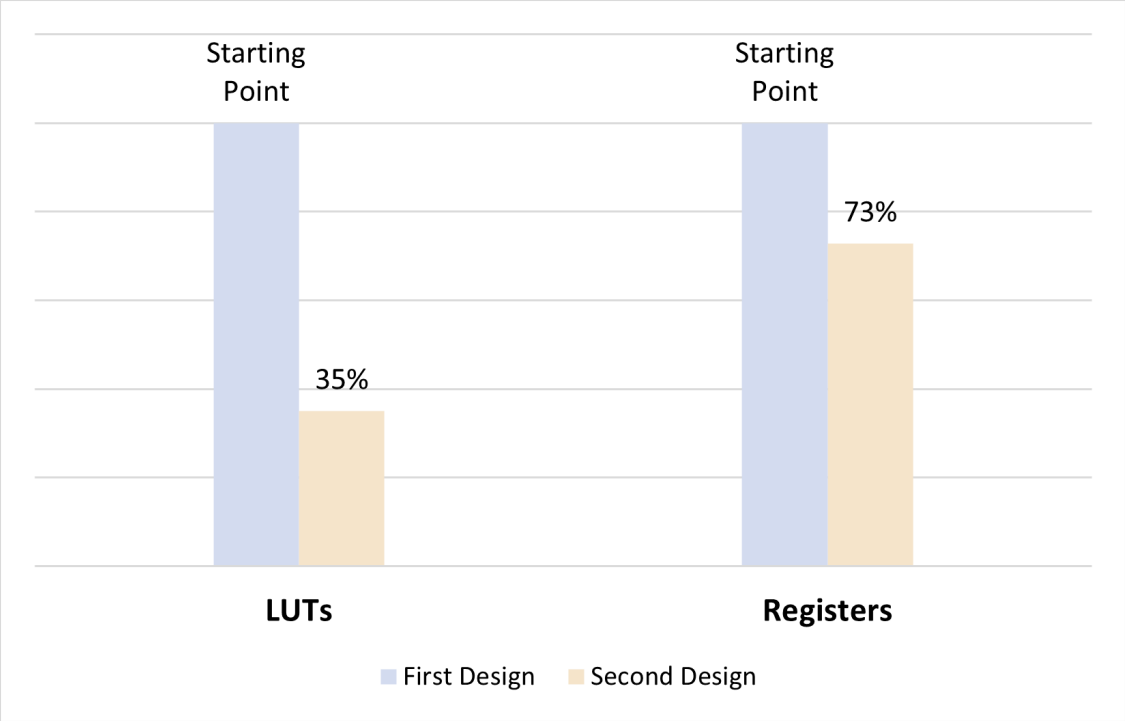

Figure 5.2: LUTs and FFs Resources Utilization Improvement per Cluster

5.2.2 Power Consumption

This paragraph describes the power analysis that has been carried out, in particular the analysis that was carried out using the resource usage stated in subsection 5.2.1. In fact, the Xilinx Power Estimator (XPE) can be used to assess dynamic and static power consumption in relation to resource utilization.
In this case, the same tool (Vivado 2021.1) has been used for resource utilization to extract the data in tables Table 5.1 and Table 5.2, and the XPE has been employed

as a consequence. More specifically, the Xilinx XPE tool allows to choose the temperature, the physical interface components, and lastly the resources themselves. The data for the two designs are shown in Figure 5.3. In this analysis, the power consumption was estimated exclusively for the accelerator array; no physical interface with data and control link was taken into account, a toggle rate of 12.5% and a 250 MHz working frequency have been considered, and finally the ambient temperature was employed (i.e., 25°C).

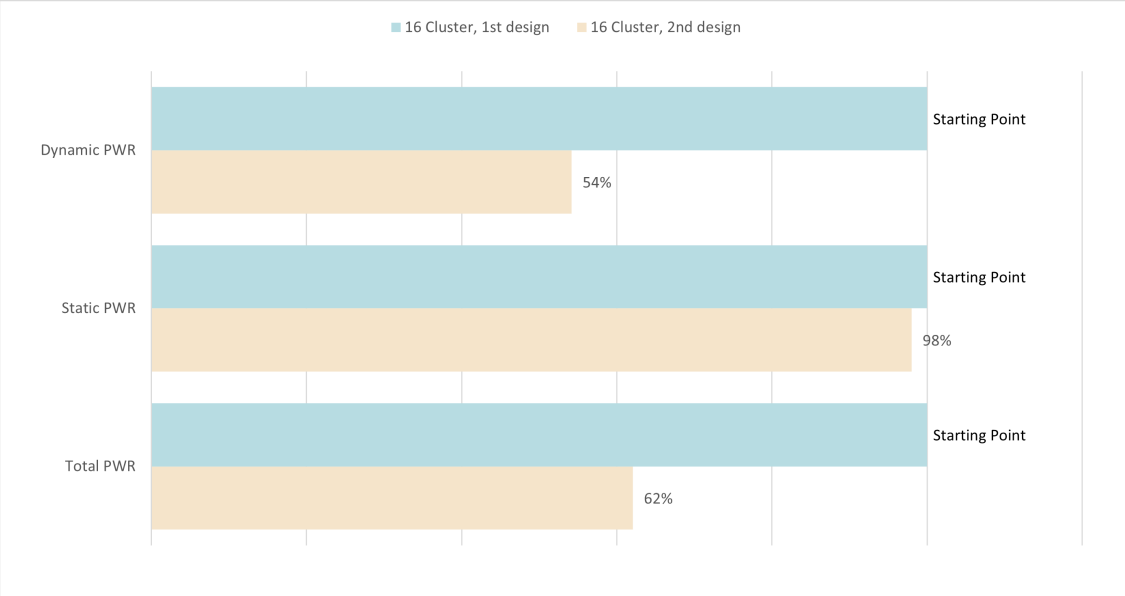

Figure 5.3: Improvement from First to Second Design Power Consumption

As can be observed from Figure 5.3, the power consumption varies as expected, exactly as the resources have been drastically lowered from the first to the second design, also the power do. In further detail.
An other aspect to observe is the clear distinction between static and dynamic power variation; as can be seen, the static power consumption increases relatively little, whereas the dynamic power contributes the majority of the power consumption.

5.2.3 Achievable Performance

The performance that the described design may achieve will be presented in this paragraph. In particular, the results that will be discussed are closely linked to the processing capabilities of a single cluster and then extended to the cluster array. Moreover, the overhead associated with loading and reading the data will not be addressed in this phase while only the clusters array performance will be estimated. The first stage was to estimate the amount of clock cycles needed to complete a computation, which was reported in Table 5.3.

	Three Circulant	**Six Circulant**
Clock Cycle per Task	85	169
Number of Required Tasks per Layer	4	4
Clock Cycle per 1 Layer Computations	$85 \cdot 4 = 340$	$169 \cdot 4 = 676$

Table 5.3: Clock Cycle Evaluation for LDPC Decoding

Since each PFU processes four data in SIMD, each cluster will process $4 \cdot 8 = 32$ data. Because each layer has 128 data, it is essential to repeat four times on the same cluster to complete a layer's processing. This is why the last row in Table 5.3 has been computed by multiplying by four.
The H-Matrix, as specified in subsection 3.3.1, is made up of four layers of three circulants and eight layers of six circulants. As a result, the total clock cycles required to decode a matrix are:

$$CLK\_CYCLEs = 4 \cdot LAYER_{3CIRC} + 8 \cdot LAYER_{6CIRC} = 4 \cdot 340 + 8 \cdot 676 = 6,768 \tag{5.1}$$

Once the latency in terms of clock cycles has been evaluated, it can be estimated the clock cycle required for a certain number of iterations required to obtain a reasonable decoding performance. Supposing to work with 20 iterations the total clock cycle will become:

$$CLK\_CYCLE = 6,768 \cdot 20 = 135,360 \tag{5.2}$$

Once the overall latency has been estimated, with a certain working frequency available, the overall performance can be evaluated. A reasonable assumption would

be to use a frequency of 250 MHz, hence a period of 4 ns. With these parameters, the computation time `T` required to process a matrix in a single cluster is:

$$T = \frac{1}{250 \cdot 10^6} \cdot 135,360 = 540\ \mu s \tag{5.3}$$

This last calculation shows that it takes 540 μs to process a matrix, which according to the CCSDS standard (1024,1/2) is made up of around 2560 bytes when considering the rate 1/2 plus a little overhead for the first transmission packets.
As a result, the throughput of a single cluster, or the amount of bits processed per second, may be calculated using a simple proportion:

$$540\ \mu s : 2,560 \cdot 8 = 1\ s : x \Longrightarrow Th = x = \frac{2,560 \cdot 8}{540 \cdot 10^{-6}} \approx 38\ Mbps \tag{5.4}$$

Considering that the throughput previously calculated is specific to a single cluster and that the overall array has 16 clusters, it may be finally possible to estimate the throughput by considering that the clusters operate independently, resulting in the following estimation:

$$Th_{array} = Th_{cluster} \cdot 16 \approx 600\ Mbps \tag{5.5}$$

This last outcome exceeds the overall system's performance requirement, which demand a minimum throughput of 20 MSps over 8 bits, meaning 160 Mbps. With a throughput of 600 Mbps on each node, the system has the required minimum performance even assuming a margin of 100% and therefore reducing it to 300 Mbps, accordingly to the latencies that can be introduced by data transfers.
Furthermore, some consideration may be given to the number of PFUs required to process 1 Mbps, which can be calculated using the following proportion:

$$128\ PFU : 600\ Mbps = x : 1\ Mbps \quad \Longrightarrow \quad x = \frac{128 \cdot 10^6}{600 \cdot 10^6} \approx 1\ PFU \tag{5.6}$$

Then, by applying a confidence margin of 100% considering the non-definitive nature of the project, it can be obtained a number of PFUs equal to: $\#PFU = 2$.
As a result, with a minimum structure of 128 PFUs, the accelerator array's resources will be utilised as:

$$PFU_{\%} = \frac{2}{128} \cdot 100 = 2\% \tag{5.7}$$

While processing the target LDPC algorithm, this fulfils the 5% array usage requirements mentioned above for 1 Mbps processing.

5.2.4 Commercial IP-CORE Comparison

The study performed to evaluate the overhead of resources introduced by the solution described in this thesis in comparison to a commercial and monolithic custom solution decoding the same matrix with the same standard [34] (CCSDS (1024, 1/2) LDPC decoding will be described in this paragraph. The goal of such study is to quantify what the trade-off is between the benefits of the current architecture flexibility and programmability and the benefits of currently existing solutions. In addition, the resources overhead has been quantified by comparing the described Node to the commercial solution delivering the desired functionality.
To strengthen the trust in this comparison, it was validated that the commercial reference used in [34] is compliant with the resource use of other solutions [35] with similar resource usage and performance, even if they are based on different standards and algorithms. In particular, the XCKU040 FPGA has been considered for the resources utilization of the designed structure, as it has been designated as the testbed target device. The usage of a commercial IP-CORE, on the other hand, was examined for use on a rad-hard device, notably the Virtex-5 [36] (XQR5VFX130), which was chosen for its consolidated heritage in the space area. The resource utilization in these two scenarios is provided in Table 5.4, with the ones used for the designed accelerator arrays (essentially in charge of the calculation) and the one required by the implementation of the commercial solution in presented.
In order to estimate the resource usage on the Virtex 5 device, the following considerations have been considered in relation to IP Core resource utilization indicated in [34], which has been mapped on the Zynq Ultrascale+ FPGA:

- **LUTs:** Because both FPGA families have the same number of LUTs per CLB, their amounts have been assumed to be equal.
- **Registers:** the two FPGA families have a 1/2 Registers per CLB ratio. Each CLB in the Ultrascale and Ultrascale+ comprises 16 Registers, but the Virtex 5 only has 8 Registers. In the worst-case scenario, the number of CLB required is double that of the Ultrascale scenario. In this view, an x2 factor has been applied for Registers.
- **BRAMs and DSPs:** Because the primitives for memory and DSP are comparable in both FPGA generations, no changes have been made.

However, given the Virtex 5 older technology and the unavailability of sophisticated synthesis and Place & Route tools, the above statistics must be considered as optimistic for rad-hard device implementation.

	Current Resources Utilization per Clusters Array [%]	Resources Utilization per Commercial IP-CORE [%]
Mapping Device	XCKU040	XQR5VFX130
LUTs	$\approx 40\%$	$\approx 15\%$
Registers	$\approx 25\%$	$\approx 25\%$
BRAMs	$\approx 15\%$	$\approx 5\%$
DSPs	$\approx 10\%$	0%

Table 5.4: Comparison Between the Described Architecture and Commercial IP-CORE Resources Utilization

As expected, the current solution has a higher averaged resource utilisation as seen from Table 5.1. On the other hand, the comparison can be improved considering that only about 85% of clusters array resources are used to achieve the performance of the commercial IP-CORE, comparing them to the clusters array performance detailed in subsection 5.2.3.
In Table 5.5 data have been derived using these assumptions, again considering the implementation of the commercial IP-CORE on a rad-hard device.

	Current Resources Utilization per Clusters Array [%]	Resources Utilization for Commercial IP-CORE[%]
Mapping Device	XCKU040	XQR5VFX130
LUTs	25%	15%
Registers	15%	25%
BRAMs	10%	5%
DSPs	5%	0%

Table 5.5: Comparison Between the Described Architecture and Commercial IP-CORE Resources Utilization with same Perfomance

As can be seen in Table 5.5, there is still a resource overhead for the current solution on COTS devices. However, the possibilities that this architecture brings through software programmability to perform different tasks allow the overhead of resources to be eclipsed by the significant advantage in terms of flexibility acquired.
Moreover, it should also be noted that the use of a COTS device is much less expensive than using a rad-hard board (i.e., about 5 times lower).

5.3 Functional Verification

The results of the tests performed to validate the system's functionality will be displayed in this section. In detail, the tests has been performed as anticipated implementing the algorithmic testbed presented in subsection 3.3.1.
The operation of a single cluster's calculation will be depicted in particular for the sake of simplicity and to prevent complicating the representation. The computing of an individual clusters, on the other hand, is meant to be parallel, so each cluster operates independently from the others.

Extracts from the simulation used to compute a 3-circulants task will be shown in chapter subsection 5.3.1.

Section subsection 5.3.2 on the other hand, will display the outcome of the 6-circulants task's computation realized on board.

5.3.1 Testbench Simulation

Highlights from the simulation will be given in this subsection, with the aim of demonstrating proper operation execution. First, the predicted outcomes of each PFU will be reported in the next paragraph, which were estimated using an excel spreadsheet to provide a baseline against which to compare the data that will be presented.

Expected Results

A test on random data has been conducted to check the system's behaviour. The input data and the expected partial and final results are shown in Table 5.6.
In detail, purely random values for the gamma values were selected while assuming that just a single loop would be done, and therefore βs were set to 0. Values are represented in hexadecimal on a 32-bit.
First, the whole simulation is presented in Figure 5.4, where the general functioning of the cluster can be observed.

Input γ Values	Input β Values	Partial α Results (VN-PROCESSING)	Partial Absolute Value Results	β Results (CN-PROCESSING)	γ Results (AP-UPDATE)
01020304	00000000	01020304	01020304	85868A88	86888D8C
05FAF608	00000000	05FAF608	05060A08	81020384	86FCF98C
F70A0BF4	00000000	F70A0BF4	090A0B0C	01828304	F88C8EF8

Table 5.6: Expected Results

From Figure 5.4 can be noticed how as soon as the start signal is received, the cluster begins execution by recording the data that comes, which is confirmed by the appropriate signal of validation. The results are sent to the output after a certain latency, and they are also confirmed by an appropriate bit of validation. Finally, the DONE signal is pulled up after all of the PFUs in the cluster have produced the result, indicating that the cluster has completed processing.

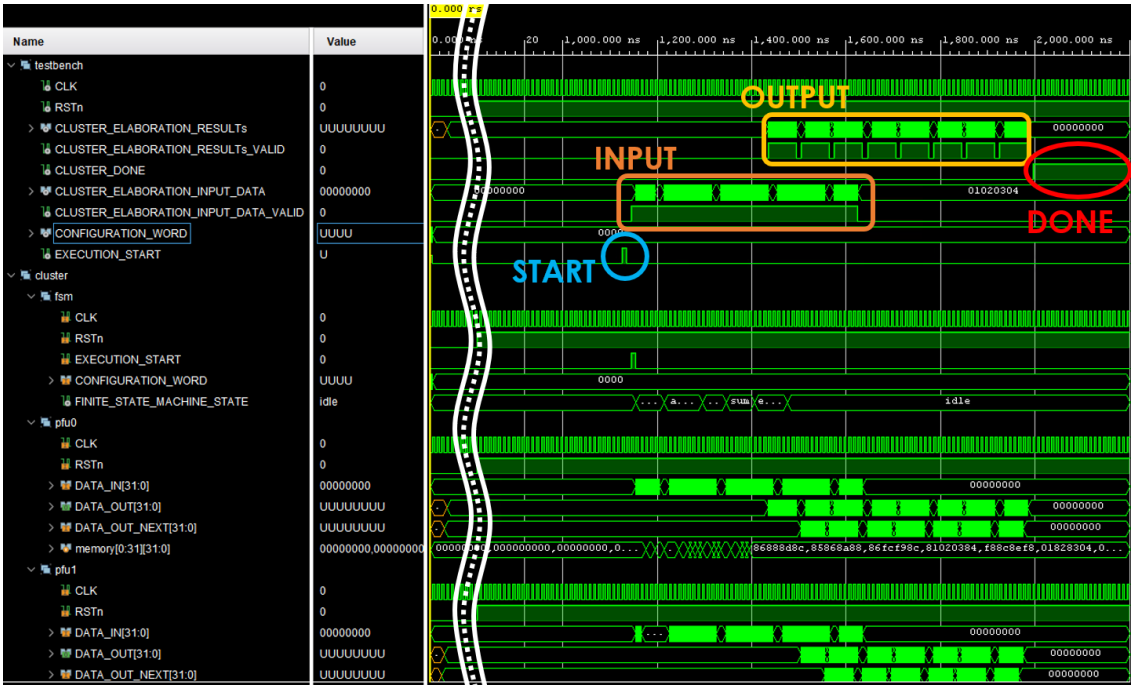

Figure 5.4: Overall Simulation Behaviour

Subsequently, the behaviour of the FSM and individual PFUs may be studied in depth. Figure 5.5 depicts the progression of the FSM states. The machine evolves from the idle state to the `st_g` state, in which the data is stored in the memory, as soon as the start signal comes.

The machine then progresses into the `alpha_abs_compute` stage, which calculates the α values and their absolute values. `The beta_compute` and `sum` states are then used to derive the new β and γ values, respectively. Finally, the new derived data are transferred to the output bus in the `ex_end` state. The FSM then goes back to idle and waits for a new start pulse.

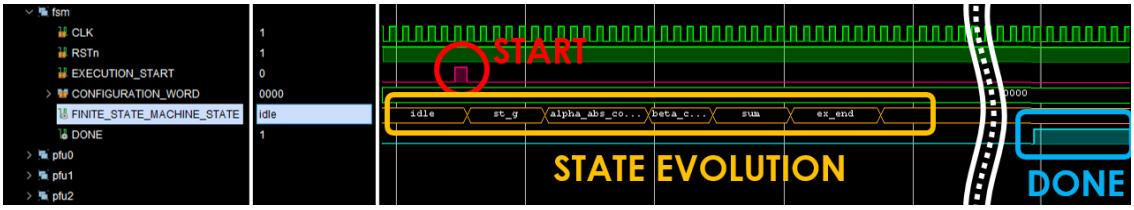

Figure 5.5: Finite State Machine Evolution

It is important to note that, while the first PFU execution ends when the FSM returns to the idle state, the done signal is raised up later, since it denotes that all PFUs in the cluster have been processed.
Next, the data transmission processes for the input and output data were then provided in Figure 5.6 and Figure 5.7, respectively. The dynamics displayed in Figure 4.5 can be seen in detail, with each PFU being linked to the input bus only after the previous PFU has stored the relevant input data. Similarly, each

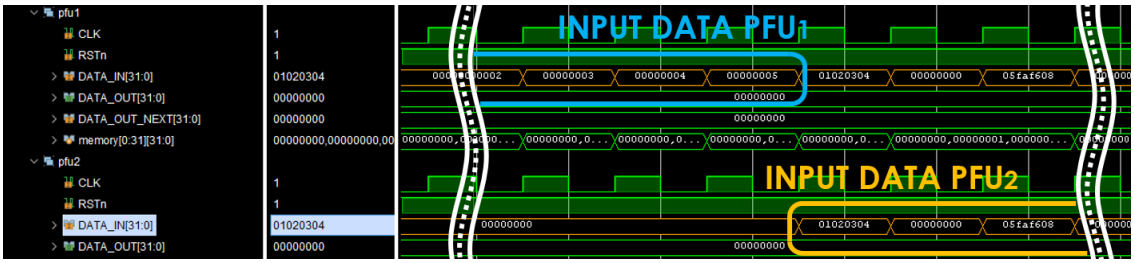

Figure 5.6: Input Data Transfer through PFUs

PFU data outputs are transferred to the preceding PFU in order to be sent on the cluster main bus.

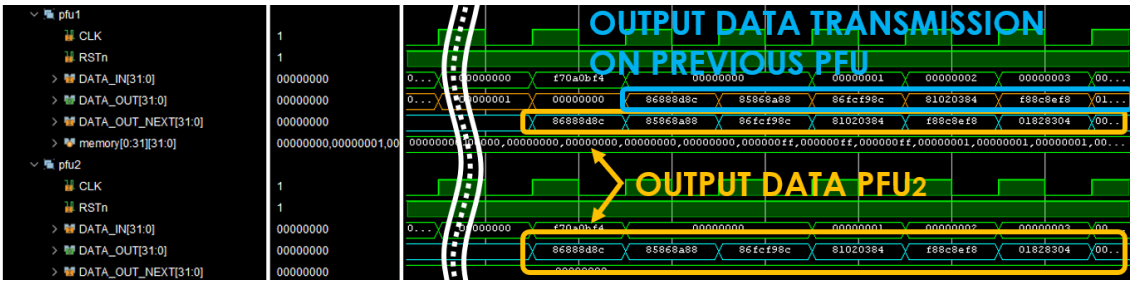

Figure 5.7: Output Data Transfer through PFUs

Finally, in Figure 5.8, it can be observed the decoding process for the identified task in progress. An example of memory usage is illustrated, in which, as mentioned in chapter 4, all results (partial and final) are stored.
In detail, referring to the algorithm described in subsection 3.3.1, it can be noticed the input data storage corresponding to the γ and β values. The partial results linked to the computation of α and its absolute values are then also stored in spare location of the memory. Finally, the β values saving is executed, followed by the new γ values.

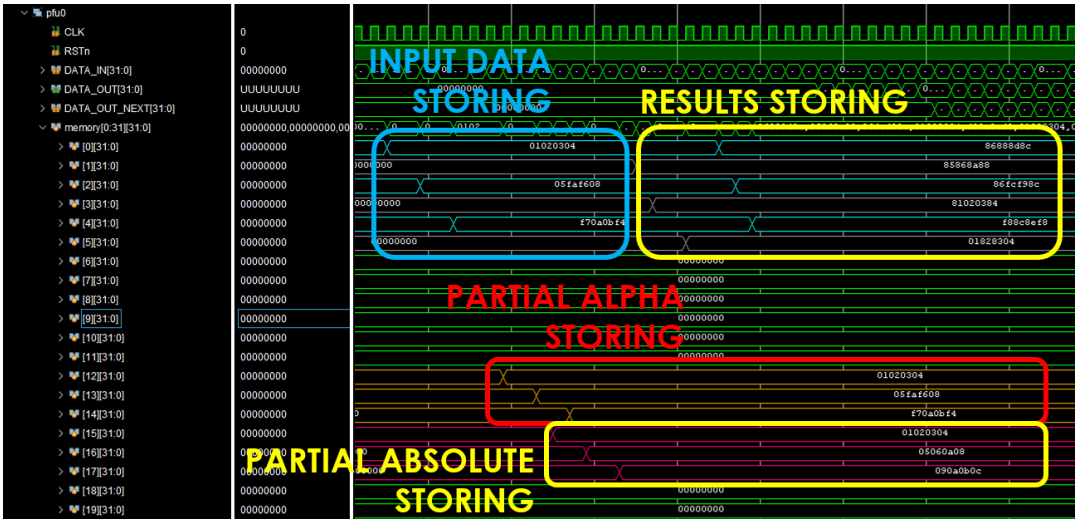

Figure 5.8: Partial and Final Results Storing

As can be observed, the results produced are equivalent to those computed using the excel spreadsheet presented in Table 5.6, indicating the successful tasks execution.

5.3.2 On Board Test

The results of the tests performed on the board will be shown in this chapter. To verify the structure behaviour, Vivado was used to create a block diagram that allows the employment of a processor with a stream and a memory mapping interfaces to transfer data and instructing tasks execution, respectively.

Vitis was then utilised to create a software that validated the board's functioning. It is responsible for conveying the input data and verify that the outputs are consistent with the desired outcomes. The functional verification setup is shown in Figure 5.9, where it can be seen the board in use and the application executing correctly, as well as the outputs that will be explored in further detail later.

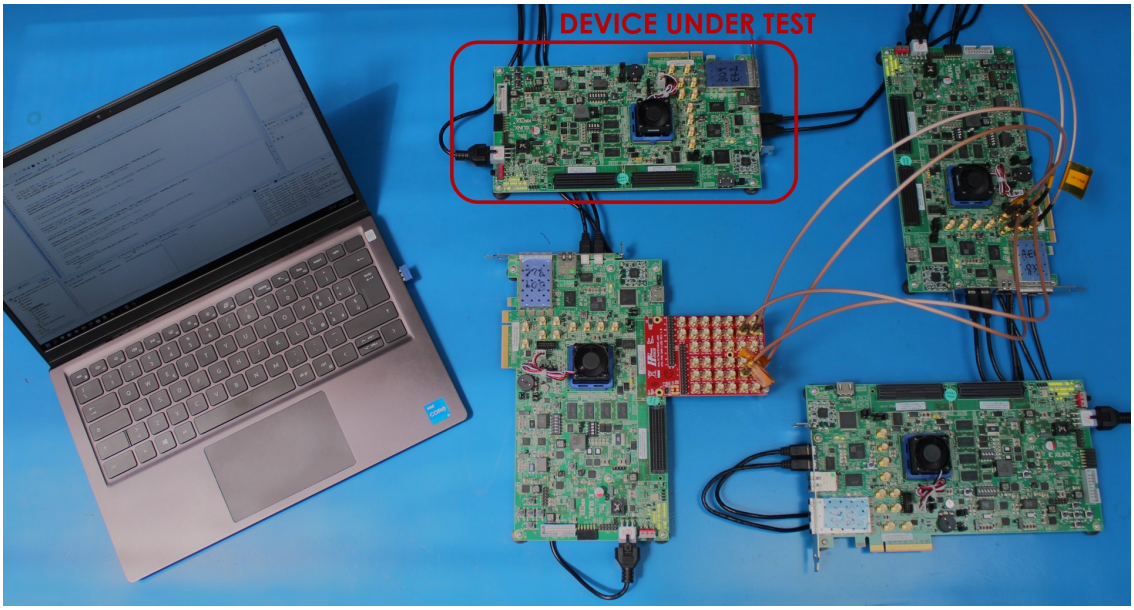

Figure 5.9: Setup for On Board Verification

In particular, the cluster functioning was tested as in subsection 5.3.1, but this time for the task of executing the algorithm stated in subsection 3.3.1 with 6 circulants.

In Figure 5.10 is shown the block diagram realized in order to obtain a design capable of interacting with the IP-CORE realized for the cluster.

In detail, an IP CORE was developed that operates as a bridge between the AXI protocol used by the CPU and the memory mapped access used within cluster. It includes two FIFOs which allow both input and output data to be downloaded and read out using proper validation signals.
This bridge allows input data to be transferred via the AXI-stream protocol, which is one of the standard protocols for sending a stream of data in a high-performance way.
Moreover, an AXI-lite protocol has been added allowing the processor to access several registers on the bridge in both read and write mode to perform the following main operations:

- The output of the FIFO was linked to one register, where the cluster's calculated results were stored. It was suitably designed such that each time that register was visited, a new result is saved there, allowing the cluster's findings to be retrieved by carrying out a series of accesses to this register.
- The configuration that permits the task to be selected was transmitted to another register.
- Another register was used to store the done bit, which allows information about the conclusion of the activities to be stored.
- A fourth register was used to store the execution time, which was estimated in terms of clock cycles from the moment the start signal was asserted until the done signal was communicated, reflecting the cluster's computing delay. This number was calculated and found to be 171 clock-cycles for the 6-circulating task and 88 clock-cycles for the 3-circulating task. This result confirms the system's desired on-board behavior since it matches exactly what was measured in simulation.

Finally, a generic processor and the UART blocks were also added, allowing data to be transferred to the cluster and the results to be provided through UART.

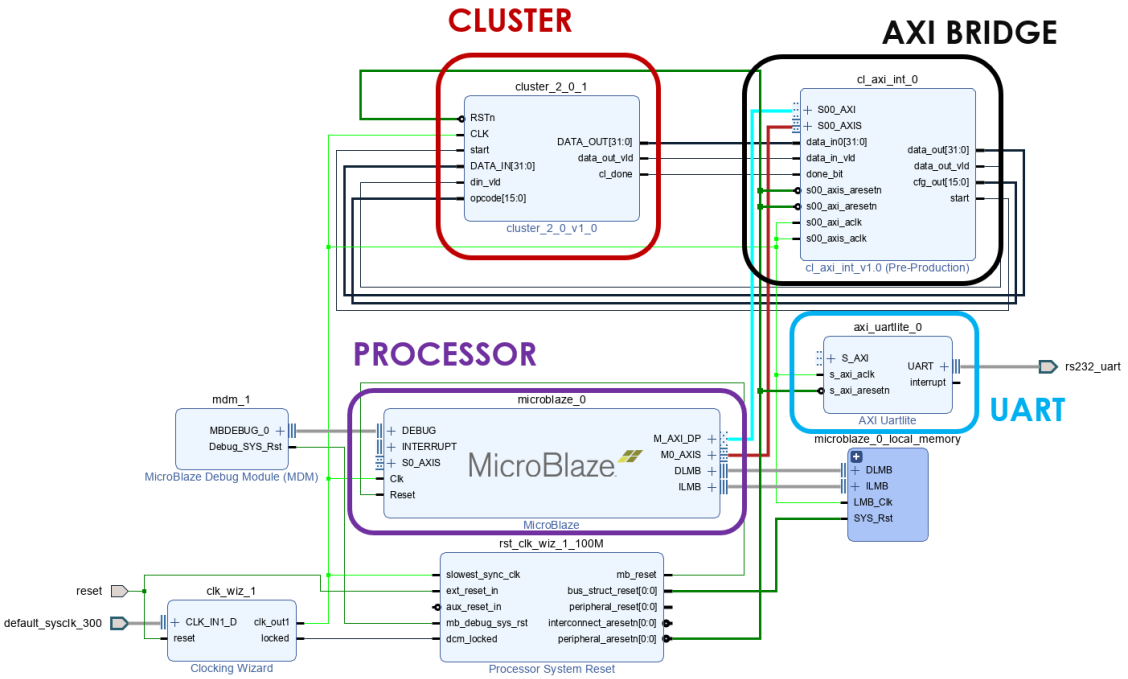

Figure 5.10: High Level Block Diagram

Subsequently, the bitstream was generated after validating the correct execution of synthesis and implementation, providing the information necessary to be downloaded within the board, as detailed in section 1.2.
In particular, it was confirmed that the Vivado tool's synthesis and implementation of the design were both consistent with what was expected in terms of resource usage and time constraints. In fact, it can happens that the design would not be placed while keeping the given frequency. This might happen when there is high occupancy or while performing an extremely dense routing that necessitates connecting specific blocks through very long pathways, preventing the parameters provided from being respected.
As a result of the successful verification of these elements, it has been possible to move on to the functional verification of the design.

As anticipated, a C code was then created and delivered to the microprocessor using Vitis. The code perform the following operations for task management:

- Loads the data to be processed by sending it through the stream interface of the processor.
- Sends the task selection sequence to the memory mapped interface.
- Polls the done signal of the cluster.
- Reads the results when the execution is finished.

Finally, in the same code, a function that compare the expected data and confirms (or not) the output results correctness was added.
The results of the automatic computation of c software and the execution of operations in hardware are shown in Appendix B, they will be compared by the code but have also been shown in the table for visual feedback.
Finally, the outputs were validated in an automatic way using a few lines of code in the microprocessor, which will return a set of information about the clusters and PFUs verified, as well as the mismatches obtained, as an output on the UART.
Figure 5.11 reports the results of the verification of the cluster under evaluation.

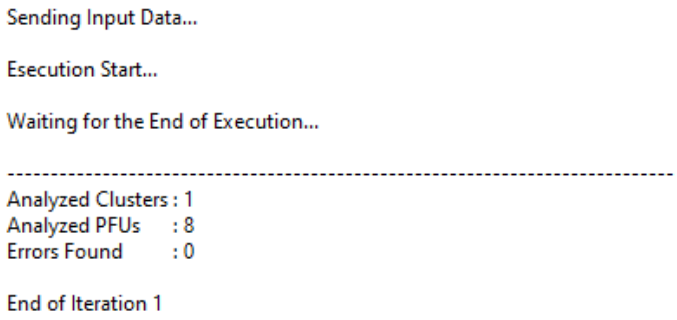

Figure 5.11: Architectural Validation Test Results

As can be seen in both Table B.1 and Figure 5.11, the expected results are the same as those achieved once the task is executed in hardware.
At the current stage, the overall Node architecture cannot be verified since a management system for transmitting and receiving input and output data is necessary for the iterative execution of tasks within the clusters.

6 Final Remarks

The design of a hardware accelerators array was reported in this thesis as part of an ESA program, and it aims to enable the leveraging of COTS devices into the space environment.
The following sections present the main conclusions findings and possible future developments to further develop the described design.

6.1 Conclusion

The essential aspects of the thesis, as well as several technical concepts gained over the course of the thesis and required for understanding it, were first introduced. The accelerator array's design phase was then presented.
In particular, the characteristics required of the accelerator array are mainly programmability and scalability. As far as programmability is concerned, tasks were introduced to create a set of instructions that can be used to perform specific operations such as those required by the LDPC algorithm used as a testbed. On the other hand, for what concern scalability, the structure was designed to reflect as far as possible the idea of replicating small fundamental structures called clusters that can be appropriately programmed and work independently.
Several clusters might be employed to conduct different activities or the same operation on different data to gain maximum flexibility. This also allows for the prospective re-allocation of resources for tasks that may not be performed due to a malfunction.

To deal with the high number of PFUs that would be put into a structure, it was discovered that the structure needed to be redesigned even if in this way no longer each single PFU can be reprogrammed but only the whole cluster (i.e. 8 PFUs) could be reprogrammed. From a programmability standpoint, this lowers flexibility, but it ensures that the array can be implemented on the board. Given the necessity for a high number of resources to control data flow and node health conditions, in fact, the clusters resources and space are limited, thus a downsizing phase of the design was done.
Finally, the system requirements were verified to ensure the design correctness. In detail, as anticipated, the system requirements described guided the design of the structure and defined the minimum characteristics and performance to be achieved by the architecture. A full comparison of the program's aims and the achieved results can be seen below.

- **The Node shall include an array of programmable accelerators.**
  This requirement have been full filled thanks to the introduction of the described architecture in chapter 4 with and the current instruction set described in subsection 4.3.3 and the capacity to perform two tasks with the ability to expand programmability to a wide variety of activities.

- **Each Node accelerators shall include an addressable memory space and a Digital Signal Processing unit.**
  Thanks to the in dept study of the DSP unit, memory and a number of scattered logic components such as multiplexers and registers described in section 4.2 and subsection 4.3.2 allows this requirement to be validated.

- **In order to support the instructions required to perform the CN Processing, VN Processing, and AP Update of a Layered Quantized Normalized Min-Sum LDPC decoding algorithm, each Node accelerator shall support at least the following SIMD instructions on operands of at least 8-bit: *Sum, Subtraction, Absolute Value, Minimum among two values.***
  The analysis of the applicability of the algorithm's steps in subsection 3.3.1, as well as the simulation of the computation in subsection 5.3.1 and the board test in subsection 5.3.2, allow this requirement to be successfully fulfilled thanks to the performing of the main operations described for the three sections of the algorithm.

- **The array of accelerators shall be composed by at least 128 pro-**

cessing elements.
As can be seen from the description of the architecture in Figure 4.3.1 and the analysis of the resource occupancy in subsection 5.2.1, the implementation of the required minimum number of PFUs is verified, and it can also be seen that there is room for improvement both with the current structure and with possible future upgrades of the array architecture that could allow an even higher number of PFUs to be included. In fact, it can be noticed from subsection 4.2.1 and Figure 4.3.1 how the architecture was envisioned to handle a hypothetical increase to 256 PFUs; i.e., twice the present amount.

- **Each 5% of available accelerators in the array shall provide an average contribution of at least 1 Mbps while performing the computation for the Layered Quantized Normalized Min-Sum LDPC decoding algorithm.**
  The performance analysis shown in subsection 5.2.3 together with the conclusions reported regarding the PFUs utilization for the required decoding rate show that it is possible to achieve 1 Mbps with a PFU usage of about 2% of the total array, thus verifying the system requirement.

- **The Node shall be able to receive and elaborate information for codewords decoding with a gross rate of 20 Mbps.**
  It has been possible to verify also this requirement through subsection 5.2.3 the possibility of achieving the required performance through an estimated throughput of about 80 Mbps against the required minimum of 20 Mbps.

As can be seen, all the requirements that guided the design of the architecture have been validated, proving that the project's goal is accomplished and that expectations are satisfied, primarily in terms of structural flexibility, without compromising calculation performance.

6.2 Future Development

Because the research and design process is still ongoing, further adjustments and improvements may be made to both the structure depicted in this thesis, which would be an early version of the array structure, and the connectivity with the rest of the design.
Beyond that, more tasks will also have to be added to the currently generated set to allow for further programmability between different sorts of algorithms and/or processes.

The interface with the node's health controller must be improved and deepened to allow for the detection of any faults and the position of the malfunctions within the cluster, allowing for intervention and reconfiguration of the area of the array where the failure occurred.

In addition, the board version specified for the flight model is the `XCKU060`, which is a bigger version of the currently used `XCKU040`. In particular, the two boards have the same functionality but differ in size, indicating that the 060 board has greater resources available. This implies that in the flight version, the accelerators array might be expanded to a larger number in order to handle even more tasks simultaneously.

Finally, some further investigations that can be carried out on design must be taken into account, in addition to reconfiguration, in fact, other techniques can be investigated such as redundancy (both structural and temporal) or even design.

A

The key properties of the PFU in terms of programmability presented in subsection 4.3.3 will be described in this appendix. The structure of the instructions that manage the elements within the PFU will be explained in more detail, as well as an extract of microcode for executing one of the two jobs needed to operate the testbed.

Regarding the presentation of the instruction set in the following, it is important to emphasise how despite the different instruction optimization techniques that prevent the length of the configuration word for the PFU from exploding [37][38], it was decided to focus the studies on other aspects of the architecture, and as a result, an almost `1:1` control mapping was used to simplify the instruction decoding phase, leaving room for improvement in terms of possible future minimization.

In Figure A.1, it can be seen the general structure of how the controls are sent to the PFUs.

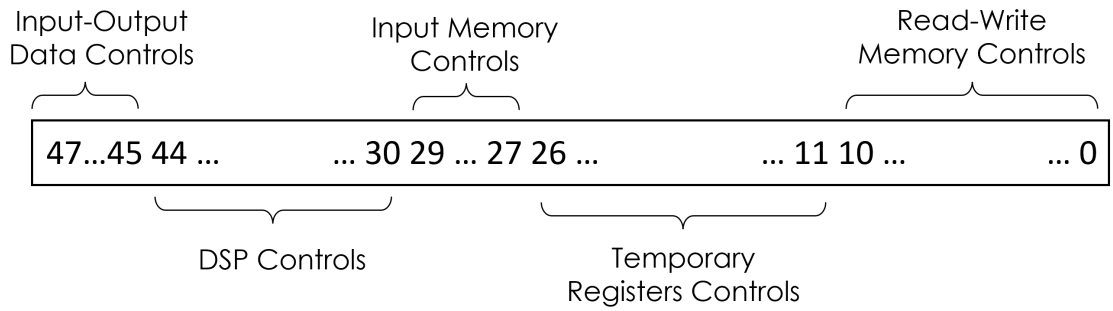

Figure A.1: Instruction Format

As can be seen, the fields are mainly divided into four categories:

- The input and output data management controls, which are used to manage the multiplexers that handle the input and output data streams described in subsection 4.3.1.
- The controls for managing the DSP unit including the three main control sequences OPMODE, INMODE and ALUMODE described in section 3.1, as well as a control for managing a multiplexer to selecting the data on which operations should be performed.
- Memory management controls, comprising read and write addresses and write enable, as well as a control for selecting data to be transmitted as input for storage via a multiplexer divided into two portions called `Input Memory Controls` and `Read-Write Memory Controls`.
- Several control signals used to enable the temporary registers.

Furthermore, as anticipated in subsection 4.3.3, a library of potential operations has been realized with the goal of producing a collection of operations, or rather tasks, that may be executed to carry out particular processes. There are two tasks in the LDPC algorithm, for example, and they are distinguished by the scenario with three or six cycles.
The order of instructions changes depending on the operations to be conducted and is appropriately adjusted for the format of the tasks to be completed. An extract of the circulating 3 task's sequence of operations is provided in Figure A.2.

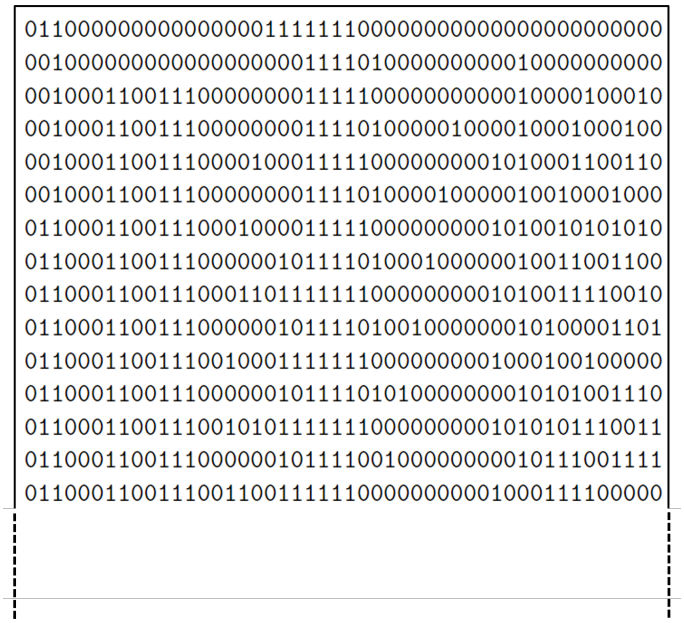

Figure A.2: Extract of Instructions Sequence from Task for 3 Circulating LDPC Decoding

In fact, the Xilinx DSP Units are capable of supporting all of the primary low-level computations required by the examined telecom features, as anticipated in section 3.1. Moreover, multiplications and additions may be easily implemented in addition to the combinations enabling logic and arithmetic operations among different signals (either with a wide bit width or in SIMD, for smaller operands).

B

The following appendix shows the results obtained from performing the LDPC encoding operations described in Figure 3.4 of a cluster, i.e. 8 PFUs.
In particular, in order to test the system's behavior, eight sets of data, randomly generated, were put into the system to confirm that each PFU worked independently of the preceding one.
Table B.1 shows the expected results and those obtained from the hardware execution in order to have a direct comparison.

PFU	**Expected β Results**	**Expected γ Results**	**β Results**	**γ Results**
PFU$_0$	05868A08	06888D0C	05868A08	06888D0C
	01020304	06FCF90C	01020304	06FCF90C
	81828384	788C8E78	81828384	788C8E78
	01020304	0EF4F414	01020304	0EF4F414
	81828384	70949670	81828384	70949670
	01020304	16ECEC1C	01020304	16ECEC1C
PFU$_1$	00000081	00000080	00000081	00000080
	00000081	00000080	00000081	00000080
	00000081	00000080	00000081	00000080
	00000081	00000080	00000081	00000080
	00000081	00000080	00000081	00000080
	00000081	00000080	00000081	00000080
PFU$_2$	02808F82	F57AA49E	02808F82	F57AA49E
	0D020F8C	0B02FB8E	0D020F8C	0B02FB8E
	82808F02	AD75DAB8	82808F02	AD75DAB8
	82809402	B77EA3F6	82809402	B77EA3F6
	82800F02	B679EDF0	82800F02	B679EDF0

	02000F82	E74DC2C9	02000F82	E74DC2C9
$\mathbf{PFU}_3$	13060189	FA4FFEA7	13060189	FA4FFEA7
	1306010F	FF25E406	1306010F	FF25E406
	13068309	DF1A84FA	13068309	DF1A84FA
	13940109	008EF5F6	13940109	008EF5F6
	13060189	001EFAD4	13060189	001EFAD4
	93860189	B14EFD98	93860189	B14EFD98
$\mathbf{PFU}_4$	19020881	47291A48	19020881	47291A48
	9C820881	837B1754	9C820881	837B1754
	99078801	47096B05	99078801	47096B05
	99020801	63262A0B	99020801	63262A0B
	99820881	7D31523F	99820881	7D31523F
	19028F84	354C8783	19028F84	354C8783
$\mathbf{PFU}_5$	01030301	EAEEF0F0	01030301	EAEEF0F0
	01830301	F290E7F8	01830301	F290E7F8
	01030507	FAFE0206	01030507	FAFE0206
	87858381	88888888	87858381	88888888
	81838381	8A8E9090	81838381	8A8E9090
	81038381	92E79898	81038381	92E79898
$\mathbf{PFU}_6$	02828195	CEA54F88	02828195	CEA54F88
	83021B0D	85D81C22	83021B0D	85D81C22
	0282010D	E4D31C49	0282010D	E4D31C49
	0282018D	FFA32F73	0282018D	FFA32F73
	0202018D	FFEC2A5A	0202018D	FFEC2A5A
	0216818D	D514563E	0216818D	D514563E
$\mathbf{PFU}_7$	810B0B8A	851F2932	810B0B8A	851F2932
	840B8B0A	851F7B5C	840B8B0A	851F7B5C
	010B8B0A	F131712F	010B8B0A	F131712F
	010B8BA5	E323389B	010B8BA5	E323389B
	81140B0A	8D1F163D	81140B0A	8D1F163D
	010B8B0A	E3608042	010B8B0A	E3608042

Table B.1: On Board Results Comparing

Bibliography

[1] K. N. et al., *An RSA encryption hardware algorithm using a single DSP block and a single block RAM on the FPGA*. International Journal of Networking and Computing, 2011.

[2] *UltraScale Architecture DSP Slice*. XILINX, 2021.

[3] T. C. C. for Space Data Systems, *TM Synchronization and Channel Coding*. CCSDS, 2017.

[4] T. T. N. Ly, *Efficient Hardware Implementations of LDPC Decoders, through Exploiting Impreciseness in Message-Passing Decoding Algorithms*, 2018.

[5] G.-L. et al., *High-Performance Embedded Computing in Space: Evaluation of Platforms for Vision-Based Navigation*, 2018.

[6] J. Alvarez and B. Walls, *Constellations, clusters, and communication technology: Expanding small satellite access to space*. IEEE, 2016.

[7] H. M. W. Zhang, T. Wu and G. Li, *Hybrid GEO and IGSO Satellite Constellation Design with Ground Supporting Constraint for Space Information Networks*. IEEE, 2018.

[8] M. Wirthlin, *High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond*. IEEE, 2015.

[9] Xilinx, *UG909 Partial Reconfiguration User Guide (v2018.1)*, 2018.

[10] M. R. F. B. Muslim, L. Ma and L. Lavagno, *Efficient FPGA Implementation of OpenCL High-Performance Computing Applications via High-Level Synthesis*. IEEE, 2017.

[11] A. P. ed al., *Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation*, 2020.

[12] U. F. A. Regulations, *2.101 Definitions*, 2017.

[13] G. L. et al., *High-Performance Embedded Computing in Space: Evaluation of Platforms for Vision-Based Navigation*, 2018.

[14] R. B. Gara Quintana-Diaz, *Software-Defined Radios in Satellite Communications*. Norwegian University of Technology and Science, 2018.

[15] D. A. et al., *Benchmarking the future of RF in space missions: From low earth orbit to deep space*. IEEE, 2017.

[16] L. Scheick, *Testing Guideline for Single Event Gate Rupture (SEGR) of Power MOSFETs*. Jet Propulsion Laboratory, 2008.

[17] M. D. et al., *Validation of a tool for estimating the effects of soft-errors on modern SRAM-based FPGAs*. IEEE, 2014.

[18] R. C. Baumann, *Radiation-induced soft errors in advanced semiconductor technologies*. IEEE, 2005.

[19] T. L. et al., *Design and Characterization of SEU Hardened Circuits for SRAM-Based FPGA*. IEEE, 2019.

[20] W. Torres-Pomales, *Software Fault Tolerance: A Tutorial*. Langley Research Center, 2000.

[21] M. R. et al., *Soft-error detection through software fault-tolerance techniques*. IEEE, 1999.

[22] Y. L. J. Zhang, T. Han and J. Li, *Real-Time Redundant Scrubbing (RRS) System for Radiation Protection on SRAM-Based FPGA*. IEEE, 2020.

[23] K. S. et al., *Mapping on Multi/Many-core Systems: Survey of Current and Emerging Trends*. Association for Computing Machinery, 2013.

[24] *UltraScale Architecture Memory Resources*. XILINX, 2021.

[25] fpgaguru, *The DSP48 Primitive*, 2018.

[26] D. P. et al., *Versatile Channelizer with DSP Builder for Intel FPGAs*. Intel, 2020.

[27] P.Hailes, *Design and implementation of flexible FPGA-based LDPC decoders*, 2018.

[28] G.F.P.Fernandes, *Parallel algorithms and architectures for LDPC Decoding*, 2010.

[29] T. C. C. for Space Data System, *Next Generation Uplink*. CCSDS, 2014.

[30] N. X. Chong Xiong, *Performance Comparison of BLAS on CPU, GPU and FPGA*. IEEE, 2020.

[31] Xilinx, *KCU105 Board - User Guide*, 2019.

[32] *UltraScale Architecture and Product Data Sheet*. Xilinx, 2020.

[33] *https : / / www . nvidia . com / it-it / geforce / graphics-cards / 30-series / rtx-3070-3070ti/*.

[34] ComBlock, *CCSDS LDPC AR4JA codes encoder/decoder*, 2022.

[35] V. P. et al., *Flexible High Throughput QC-LDPC Decoder With Perfect Pipeline Conflicts Resolution and Efficient Utilization*. IEEE, 2020.

[36] Xilinx, *Radiation-Hardened, Space-Grade Virtex-5QV Data Sheet*, 2018.

[37] M. Liu and Q. Cai, *A research for the optimization of MIPS instruction set simulation*. International Conference on Computer Science & Education, 2009.

[38] K. Geetha and N. A. Gounden, *Compressed Instruction Set Coding (CISC) for Performance Optimization of Hand Held Devices*. International Conference on Advanced Computing and Communications, 2008.

Acknowledgments

I would like to take this opportunity to express my gratefulness to everyone who has helped me achieve this important objective.

First of all, I would want to express my gratitude to Emilio Fazzoletto, my company supervisor, and all of my Argotec colleagues for allowing me to participate in this significant project. I also want to express my gratitude to Ludovica and Eugenio, who have been incredibly supportive towards me during this journey and have transmitted so much to me both professionally and emotionally.

I would also want to express my gratitude to my Turin friends and colleagues, with whom I experienced the highs and lows of university life as a student away from home.

My family deserves special recognition for their unconditional support and confidence in me during this long journey.

Last but not least, I want to express my gratitude to Federica for her constant support and for always being there for me, both in times of triumph and in times of adversity, day by day.