Master's Degree in
Micro and Nanotechnologies for Integrated Systems

Master's Degree Thesis

On-Chip Processing for Pixel Imagers in a 28nm technology for High Energy Physics applications

Supervisors

Dr. Davide CERESA

Prof. Guido MASERA

Prof. Adil KOUKAB

Candidate

Jashandeep DHALIWAL

September 2022

Abstract

English version

This thesis explores a novel approach to on-chip data processing and readout in pixel imagers based on embedded processors for High Energy Physics applications. This subject represents a new field of development posing new challenges both at the system and implementation levels.

The first part of the thesis proposes a methodology to evaluate the performance of a Network-on-Chip (NoC) connecting several processing elements. Network latency has been studied taking into consideration several input parameters and different topologies. The maximum sustainable particle rate before the system diverges has also been analyzed. Such a methodology can be used to compare distributed solutions based on NoC to single-processor solutions and help define the readout architecture for the pixel imagers.

The second part studies embedded processors for on-chip data processing and readout. Given the limited power, area, and latency budget available for High Energy Physics applications, general-purpose processors are typically too slow or power and area hungry. For this reason, the concept of Application Specific Instruction-set Processor (ASIP) has been introduced for the first time in the field of High Energy Physics. The objective is to demonstrate a workflow to design a processor from the Instruction Set Architecture (ISA) and micro-architecture to its physical implementation. In addition, functional verification and profiling require the development of an application-specific test code providing measurements of cycle count, instruction, and functional utilization. Finally, a complete RTL-to-GDS implementation flow in a 28 nm CMOS technology provides the relevant figures of merit regarding achievable frequency, area occupation, and power consumption completing the evaluation and enabling further optimization.

This project establishes the first exploratory study for both Network-on-Chip and ASIP applied to on-chip data processing and readout for future experiments in High Energy Physics.

Abstract

Versione italiana

Questa tesi esplora un nuovo approccio all'elaborazione e alla lettura dei dati on-chip nei pixel imager basati su processori integrati per applicazioni di fisica delle alte energie. Questo argomento rappresenta un nuovo campo di sviluppo che pone nuove sfide sia a livello di sistema che di implementazione.

La prima parte della tesi propone una metodologia per valutare le prestazioni di una Network-on-Chip (NoC) che collega diversi elementi di elaborazione. La latenza della rete è stata studiata prendendo in considerazione diversi parametri in ingresso e diverse topologie. È stato analizzato anche il tasso massimo di particelle sostenibile prima che il sistema diverga. Questa metodologia può essere utilizzata per confrontare soluzioni distribuite basate su NoC con soluzioni a singolo processore e contribuire a definire l'architettura di lettura nei pixel imager.

La seconda parte studia i processori integrati per l'elaborazione e la lettura dei dati on-chip. Dato il budget limitato di potenza, area e latenza disponibile per le applicazioni di fisica delle alte energie, i processori per uso generale sono tipicamente troppo lenti o affamati di potenza e area. Per questo motivo, il concetto di Application Specific Instruction-set Processor (ASIP) è stato introdotto per la prima volta nel campo della fisica delle alte energie. L'obiettivo è quello di dimostrare un flusso di lavoro per la progettazione di un processore, dall'architettura del set di istruzioni (ISA) e dalla microarchitettura fino alla sua implementazione fisica. Inoltre, la verifica funzionale e il profiling richiedono lo sviluppo di un algoritmo specifico per l'applicazione che fornisca misurazioni del conteggio dei cicli, delle istruzioni e dell'utilizzo delle funzioni. Infine, un flusso di implementazione completo da RTL a GDS in una tecnologia CMOS a 28 nm fornisce le cifre di merito relative alla frequenza raggiungibile, all'occupazione dell'area e al consumo di energia, completando la valutazione e consentendo ulteriori ottimizzazioni.

Questo progetto stabilisce il primo studio esplorativo di Network-on-Chip e ASIP applicato all'elaborazione e alla lettura dei dati on-chip per i futuri esperimenti di fisica delle alte energie.

Abstract

Version française

Cette thèse explore une nouvelle approche pour le traitement et la lecture des données sur puce dans les imageur à pixel basés sur des processeurs embarqués pour les applications de la physique des hautes énergies. Ce sujet représente un nouveau domaine de développement posant de nouveaux défis tant au niveau du système que de l'implémentation.

La première partie de la thèse propose une méthodologie pour évaluer les performances d'un Network-on-Chip (NoC) connectant plusieurs éléments de traitement. La latence du réseau a été étudiée en prenant en compte plusieurs paramètres d'entrée et différentes topologies. Le taux de particules maximum soutenable avant que le système ne diverge a également été analysé. Cette méthodologie peut être utilisée pour comparer les solutions distribuées basées sur le NoC aux solutions à processeur unique et aider à définir l'architecture de lecture pour les imageurs à pixels.

La deuxième partie étudie les processeurs embarqués pour le traitement et la lecture des données sur la puce. Compte tenu du budget limité en termes de puissance, d'espace et de latence disponible pour les applications de physique des hautes énergies, les processeurs polyvalents sont généralement trop lents ou trop gourmands en énergie et en espace. C'est pourquoi le concept de Application Specific Instruction-set Processor (ASIP) a été introduit pour la première fois dans le domaine de la physique des hautes énergies. L'objectif est de démontrer un flux de travail pour concevoir un processeur à partir de l'Instruction Set Architecture (ISA) et de la micro-architecture jusqu'à son implémentation physique. En outre, la vérification fonctionnelle et le profilage nécessitent le développement d'un algorithme spécifique à l'application fournissant des mesures du nombre de cycles, des instructions et de l'utilisation des fonctions. Enfin, un flux d'implémentation RTL-to-GDS complet dans une technologie CMOS 28 nm fournit les facteurs de mérite pertinents concernant la fréquence réalisable, l'occupation de la surface et la consommation d'énergie, ce qui complète l'évaluation et permet une optimisation ultérieure.

Ce projet constitue la première étude exploratoire des Network-on-Chip et des ASIP appliquée au traitement et à la lecture des données sur puce pour les futures expériences de physique des hautes énergies.

Acknowledgements

I would like to express my deepest gratitude to my supervisor at CERN, Doctor Davide Ceresa, for his invaluable guidance and patience throughout this thesis. I would also like to thank my colleagues at the microelectronics section of CERN for the support and friendly environment.

I would like to acknowledge Professor Guido Masera as my supervisor at Politecnico di Torino and Professor Adil Koukab as my supervisor at EPFL for their valuable comments and precious advises on this thesis.

Finally, the completion of the master's thesis marks the culmination of an extraordinary journey, and I want to thank everyone who helped me along the way. I am immensely grateful to my parents for their love and sacrifices to give me a better life. Special thanks to my brother for his continuous support and encouragement. I would like to extend my sincere gratitude to Sukh and Riccardo for the unforgettable laughs, tears, and memories. Last but not least, a heartfelt thanks to Matilde for being my love and constant emotional support for a long time.

Table of Contents

List of Tables VIII

List of Figures IX

1 Introduction 1
- 1.1 CERN . . . 1
  - 1.1.1 High Luminosity-LHC upgrade . . . 3
- 1.2 R&D Programme on Technologies for Future Experiments . . . 4
  - 1.2.1 Technology transition . . . 5
- 1.3 Data readout and processing in HEP . . . 6
  - 1.3.1 Literature review of HEP detector electronics . . . 8
- 1.4 Objective of the thesis . . . 10
  - 1.4.1 On-chip data processing for future experiments . . . 11
  - 1.4.2 Thesis organization . . . 14

2 On-chip communication architecture for data processing 15
- 2.1 On-chip communication networks . . . 15
- 2.2 Network-on-Chip . . . 16
  - 2.2.1 Network-on-Chip for data processing . . . 17
  - 2.2.2 Packet switching technique . . . 17
- 2.3 Network-on-Chip analysis . . . 18
- 2.4 Network-on-Chip results . . . 27
- 2.5 Final considerations . . . 31

3 On-chip Processor for data processing 33
- 3.1 Application Specific Instruction-set Processor . . . 35
  - 3.1.1 ASIP optimization space . . . 36
- 3.2 Workflow . . . 37
- 3.3 nML: a structural processor language . . . 39
- 3.4 Processor architecture . . . 41
  - 3.4.1 Processors' description . . . 42

3.4.2 Memory interface 43
3.5 Processor customization 44
3.5.1 APB interface 44
3.5.2 Load and Store instructions of Interface Memory 50
3.5.3 On-Chip Debugging 56
3.6 Final considerations 58

4 Algorithm and Profiling 59
4.1 Data processing Algorithm 59
4.1.1 Input data 60
4.1.2 Application-specific algorithm 65
4.2 Processor Profiling 68
4.3 Final considerations 80

5 Physical implementation 81
5.1 First implementation 82
5.2 RTL optimization of Tmicro microprocessor 88
5.3 Frequency improvement 92
5.3.1 First frequency improvement 93
5.3.2 Second frequency improvement 94
5.4 Final considerations 98

6 Conclusions 99

A Network on Chip analysis on MATLAB 101

Bibliography 105

List of Tables

2.1 Summary table for NoC specifications 24
2.2 Cut-off particle rate for the eight analyzed cases. 30

5.1 Summary table of total instances and area for Trv32p3 and Tmicro microprocessors after synthesis and after PnR. 84
5.2 Summary table of total instances and area for Tmicro microprocessor without trace buffer after synthesis and after PnR. 90
5.3 Summary table of clock frequency, total instances, area, density, total power consumption for Trv32p3, Tmicro, and Tmicro without trace buffer microprocessors after PnR. 92
5.4 Summary table of total instances, area, density, and total power consumption after PnR for Trv32p3 microprocessor implemented at 333 MHz and 666 MHz of clock frequency. 93
5.5 Summary table of standard cell library, total instances, area, density and total power consumption after PnR for Trv32p3 microprocessor implemented at 333 MHz, 666 MHz and 1 GHz of clock frequency. . 97

List of Figures

1.1 Illustration of the LHC accelerator and experiments. 2
1.2 Block diagram of a typical pixel detector readout flow. 7
1.3 Front-end circuit of a typical readout ASIC for pixel detectors. . . . 7
1.4 Configuration of data processing at the periphery. 12
1.5 Configuration of on-pixel data processing. 13

2.1 Maximum traffic on a Network-on-Chip applied to a pixel array. . . 19
2.2 Network-on-Chip router model with a switch, FIFO buffers and four channels. 20
2.3 4×4 pixels NoC node dimension. 25
2.4 32×32 pixels NoC node dimension. 25
2.5 3D graph of latency depending on bandwidth delay and particle rate with the legend on the top. 28
2.6 2D graph of latency depending on bandwidth delay and particle rate with the legend on the top. 29
2.7 Log-log plot of cut-off particle rate vs the number of pixels per NoC node at operating frequencies of 80 MHz and 320 MHz. 31

3.1 Flexibility and efficiency diagram for several types of data processing solution. 35
3.2 Tree diagram of the optimization space provided by ASIP Designer. 36
3.3 Flow of ASIP Designer by Synopsys. 38
3.4 Integration of customized interface from the processor towards an Interface Memory. 45
3.5 Read transfer signals in AMBA APB system bus protocol. 46
3.6 Write transfer signals in AMBA APB system bus protocol. 46
3.7 Read transfer signals toward Interface Memory in customized Trv32p3 microprocessor using AMBA APB system bus. 47
3.8 Write transfer signals toward Interface Memory in customized Trv32p3 microprocessor using AMBA APB system bus. 47

3.9 Read transfer signals toward Interface Memory in customized Tmicro microprocessor using AMBA APB system bus. 47
3.10 Write transfer signals toward Interface Memory in customized Tmicro microprocessor using AMBA APB system bus. 47

4.1 Flow chart of current data analysis. 59
4.2 First measurement results of Timepix3 at the Environmental Research Station Schneefernerhaus on Zugspitze. 60
4.3 Flow chart of the application-specific data processing algorithm. . . 66
4.4 Graphical representation of clustering algorithm. 67

5.1 Place and Route implementation of Trv32p3 microprocessor. 83
5.2 Place and Route implementation of Tmicro microprocessor. 84
5.3 Total power consumption graph of Trv32p3 microprocessor at different frequencies and input activities. 86
5.4 Total power consumption graph of Tmicro microprocessor at different frequencies and input activities. 87
5.5 Summary table of power analysis of Trv32p3 microprocessor. 88
5.6 Hierarchy of Trv32p3 microprocessor instances. 89
5.7 Hierarchy of Tmicro microprocessor instances. 89
5.8 Place and Route implementation of Tmicro microprocessor without trace buffer. 90
5.9 Power consumption of Tmicro microprocessor without trace buffer module at different frequencies and input activities. 91

Acronyms

p_T Transverse momentum. 8

AGU Address Generation Unit. 42, 43, 53–55, 77

ALICE A Large Ion Collider Experiment. 2

ALU Arithmetic Logic Unit. 42, 43, 72, 74, 78

AMBA Advanced Microcontroller Bus Architecture. 14, 44–46, 100

APB Advanced Peripheral Bus. 14, 44–46, 48, 56, 58, 68, 100

ASIC Application Specific Integrated Circuit. 5, 6, 8, 9, 36, 59, 63, 81

ASIP Application Specific Instruction-set Processor. 14, 34–36, 39, 58, 99, 100

ATLAS A Toroidal LHC ApparatuS. 2, 3, 9

BX bunch-crossing. 3

CAD Computer-Aided Design. 6

CERN European Organization for Nuclear Research. 1, 3, 4, 14, 15, 99

CLIC Compact Linear Collider. 4

CMOS Complementary Metal Oxide Semiconductor. 5, 6, 8–10, 14, 34, 82, 96, 100

CMS Compact Muon Solenoid. 2, 3, 8, 9, 21

CTS Clock Tree Synthesis. 81, 82, 84

DAQ Data Acquisition. 6, 13

DM Data Memory. 41, 44, 45, 50, 51, 56, 63–65, 72, 74, 76–80, 100

DRC Design Rule Checking. 81

DSP Digital Signal Processor. 6, 33, 35, 59

EHVT Extremely-High Threshold Voltage. 82

EP Experimental Physics. 1, 4

EPFL École Polytechnique Fédérale de Lausanne. 1

ESE Electronic Systems for Experiments. 1

FCC-ee Future Circular Collider for electron-positron collisions. 4

FCC-he Future Circular Collider for proton-electron collisions. 4

FCC-hh Future Circular Collider for hadron-hadron and proton-proton collisions. 4

FIFO First-In-First-Out. 19

FinFET Fin Field-Effect Transistor. 5, 6

FPGA Field-Programmable Gate Array. 6, 59

FSM Finite State Machine. 35

GPU Graphics Processing Unit. 33

HDL Hardware Description Language. 41, 49, 50

HEP High Energy Physics. 1, 5, 14, 15, 42, 59

HL-LHC High Luminosity-LHC. 3, 4, 8, 9

HPD Hybrid Pixel Detector. 9, 10

HVT High Threshold Voltage. 82

IO Input-Output. 41, 43–45, 49

IP Intellectual Property. 5, 42, 44

ISA Instruction Set Architecture. 38, 42, 43

ISS Instruction Set Simulator. 38, 39, 41, 49, 50, 75

LEC Logic Equivalence Checking. 81

LHC Large Hadron Collider. 1–3, 8, 9, 25, 61

LHCb LHC-beauty. 2, 21

LS3 Long Shutdown 3. 4

LVS Layout versus schematic. 81

LVT Low Threshold Voltage. 82, 95, 96

MAC Multiply and Accumulate Unit. 43

ME Micro-Electronics. 1, 44, 99

MPA Macro-Pixel ASIC. 8–10

NoC Network-on-Chip. 12, 14–19, 24–27, 30–32, 98–100

OCD On-Chip Debugging. 56–58

P2P Point-to-Point. 15, 16

PCU Processor Control Unit. 41, 49, 58

PDG Primitives Definition and Generation. 41, 44, 48

PE Processing Element. 11–14, 17, 18, 31, 32, 98, 99

PM Program Memory. 41, 44, 45, 50, 56, 79

PnR Place and Route. 39, 81–83, 85, 90, 94–96

R&D Research and Development. 4

RAW Read After Write. 77

RISC-V Reduced Instruction Set Computer five. 11, 12, 42, 43, 100

RTL Register Transfer Level. 38, 39, 41, 49, 50, 81, 84, 89

SDK Software Development Kit. 38

SIMD Single Instruction Multiple Data. 37

sLVS scalable Low Voltage Signaling. 8

SM Standard Model. 3

SNR Signal to Noise Ratio. 6, 7

SoC System-on-a-Chip. 14, 44, 100

SP Super-Pixel. 9, 10, 12, 17, 18, 21, 32, 98, 99

SSA Short Strip ASIC. 8–10

SVT Standard Threshold Voltage. 82, 83, 93, 95–97

TCE Transport triggered architecture-based Co-design Environment. 36

TID Total Ionizing Dose. 9, 82

ToA Time-of-Arrival. 8, 10, 21, 62, 64–68

ToT Time-over-Threshold. 8–10, 21, 62, 64, 65, 67

TSV Through-Silicon Via. 5

UDM Ultra Density Memories. 82

UHVT Ultra-High Threshold Voltage. 82

ULVT Ultra-Low Threshold Voltage. 82, 96, 97

VHDL Very high speed integrated circuits Hardware Description Language. 39, 41

VLIW Very Long Instruction Word. 37

VLSI Very Large-Scale Integration. 33

WP Work Package. 4–6

WWW World Wide Web. 3

Chapter 1

Introduction

This master's degree thesis has been conducted at the Micro-Electronics (ME) section of the Electronic Systems for Experiments (ESE) group under the Experimental Physics (EP) department of CERN, in cooperation with the university of École Polytechnique Fédérale de Lausanne (EPFL).

1.1 CERN

The study of the nature of particles that compose matter and their interactions is known as High Energy Physics (HEP). It is performed by colliding accelerated charged particle beams and observing the interaction of their daughter products with detectors placed around the collision point. The European Organization for Nuclear Research (CERN), hosts the world's largest and most powerful high-energy particle accelerator and collider known as the Large Hadron Collider (LHC) [1]. It is located at the Franco-Swiss border close to Geneva, Switzerland. The CERN acronym origins from the name Conseil Européen pour la Recherche Nucléaire, an organization established in 1954 to become a world-class scientific research facility in nuclear physics. Today, CERN officially counts 23 Member States and several others involved in numerous experiments.
CERN has four main missions:

- Research: investigating and answering Universe-related queries
- Collaboration: bringing people from all around the world together to advance in science for the benefit of everyone
- Technology: pushing further the frontiers of technology
- Education: training the next generation of scientists and engineers

The LHC is a 27 km long ring of superconducting magnets, placed 100 m underground, where many accelerating devices are used to increase the particle energy as it travels through the system. Two proton beams are created and progressively accelerated through serial accelerators before reaching the LHC, where they are further accelerated in opposite directions in two separated ultrahigh vacuum tubes. In the LHC accelerator, these beams are accelerated via superconductive electromagnets of 8.3 T operating at a temperature of 1.9 K. When both proton beams reach a velocity close to the speed of light up to record energy of 6.5 TeV each, they are made to collide in four different locations along the ring equivalent to the four experiments: A Toroidal LHC ApparatuS (ATLAS) [2], Compact Muon Solenoid (CMS) [3], LHC-beauty (LHCb) [4] and A Large Ion Collider Experiment (ALICE) [5], as shown in figure 1.1. These experiments are made of several sub-detectors

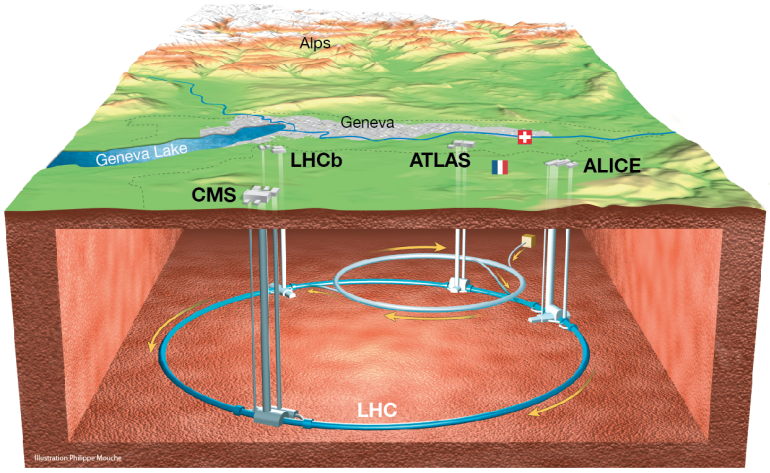

Figure 1.1: Illustration of the LHC accelerator and experiments. [6]

used to collect daughter products and produce information about the energy, mass, position, momenta, and scattering angle of particles to recreate collision images. This is done through different devices present in the same experiment such as:

- Electromagnets: used to bend out-coming particles from the collision point through a strong solenoid magnet to measure momentum and charge
- Calorimeters: used to stop particles by passive layers to measure their energy
- Trackers: these silicon sensors are placed in several concentric layers around

the collision point to measure the direction of particles previously deviated by the electromagnets.

Additional devices might be integrated for specific measures such as the muon detector installed in CMS to detect muon particles unable to be stopped by calorimeters. The LHC accelerator can deliver proton-proton collisions with a center-of-mass energy up to 13 TeV and it reached an instantaneous luminosity of 10^{34} $cm^{-2}{\cdot}s^{-1}$ in 2018 [7]. Luminosity measures the number of collisions occurring in a square centimeter per second, it represents the particles' hit rate. These particles bunch are brought into collision at a reference frequency of 40 MHz also known as bunch-crossing (BX) frequency.
A complete description of the LHC accelerator regarding the main ring, infrastructures, general services, and injector chain is provided in its design report [8].
A remarkable achievement at CERN, in terms of the development of new technologies, is the World Wide Web (WWW) created to ease data and information communication among scientists. Although, the most important achievement at CERN, thanks to the LHC accelerator, is the discovery of a new particle at approximately 125 GeV named Higgs boson: the last missing particle to conclude the Standard Model (SM) [9]. Such a discovery was separately achieved by the two general-purpose experiments: ATLAS and CMS.

Numerous other phenomena outside of the SM remain to be investigated such as the matter-antimatter asymmetry, super-symmetry extensions to the SM to solve pending issues, and the inconsistency of neutrino oscillation with the SM. Indeed, an upgrade of the LHC is required to reach higher luminosity and finally answer these queries.

1.1.1 High Luminosity-LHC upgrade

The High Luminosity-LHC (HL-LHC), represents a significant upgrade of the current LHC accelerator in the next years [10]. The luminosity will be quintupled reaching a baseline value of 5 $\cdot 10^{34}$ $cm^{-2}{\cdot}s^{-1}$ and an ultimate luminosity of 7.5 $\cdot 10^{34}$ $cm^{-2}{\cdot}s^{-1}$ [8]. Such luminosity will allow physicists to better understand the Higgs boson by increasing its probability of occurrence. With the current LHC, their interactions are measured with a 20 % of uncertainty and this is expected to drop below 1 % with the HL-LHC upgrade. Moreover, since the number of collisions increases, novel particles might be discovered to explain dark matter and electroweak symmetry breaking.
In order to reach such a high luminosity, new superconductive electromagnets able to reach 11-12 T are needed as well as novel power converters to distribute more current. In addition, a new infrastructure is necessary to dispense a novel cryogenic system to cool down superconductive electromagnets.

Consequentially to HL-LHC upgrade, the four experiments need to be upgraded as well. Original solutions are necessary for higher radiation tolerance devices to sustain harsher environments. Additionally, novel solutions are going to be implemented in terms of detectors and readout electronics to sustain such a high particle rate. Finally, due to the higher hit rate and pile-up density, a new data processing approach becomes mandatory to increase computation capabilities. A novel design for on-chip data processing is going to be presented in this thesis.

1.2 R&D Programme on Technologies for Future Experiments

At CERN, the EP department has established a Research and Development (R&D) programme on technologies for future experiments started in 2020 for at least five years [11]. The aim is to advance in experimental high-energy physics, thus instrumentation becomes essential. Up to now, CERN has well detailed the future schedule until the realization of the HL-LHC upgrade between 2024 and 2026 during the Long Shutdown 3 (LS3). The next 15 years will completely focus on the exploration of HL-LHC potentialities until its conclusion, estimated in 2040. On the other hand, further studies are required for all those projects that are going to be realized after the LHC such as the Compact Linear Collider (CLIC) [12], Future Circular Collider for hadron collisions such as heavy ions or proton-proton (FCC-hh), electron-positron collisions (FCC-ee) and proton-electron (FCC-he) [13]. The aim is to reach 100 TeV of center-of-mass energy in a 100 km long ring through superconductive electromagnets with a magnetic field of 16 T. For each of these projects, a complete conceptual design report has been published and reviewed by an international committee. These projects will provide huge benefits in particle research but also challenges in all fields of magnets and cooling systems, mechanics, and electronics. For these reasons, R&D guidelines have been established to determine the upcoming experimental difficulties and explore solutions. This R&D programme is sub-divided into eight different Work Packages (WPs) with different development purposes:

- WP1: New detector technologies for both monolithic and hybrid pixel detectors with an emphasis on silicon tracking and vertexing detectors.

- WP2: More performing gas-based detectors for radiation detection with a good signal-to-noise ratio for single particle sensitivity

- WP3: Novel high granularity noble liquid calorimetry and light-based detectors with high resolution of energy and position with an excellent radiation hardness

- WP4: Detector mechanics with a focus on large dimensions, high radiation level, higher spatial resolution, very low material budget, and better detector cooling systems
- WP5: High-performance integrated circuits in 28 nm and 16 nm technology with a focus on higher density and speed and lower power consumption along with radiation validation.
- WP6: Increase performance of high-speed data links by increasing data rate and radiation hardness
- WP7: Software development of data acquisition systems and triggers for data taking, simulation, and analysis
- WP8: Advanced detector magnets and magnet systems with a focus on powering, low mass superconducting cables, cryostat studies, controls, and safety

This thesis has been carried out under the WP5 which is focused on Application Specific Integrated Circuits (ASICs) development for HEP particle detectors. WP5 focuses on two main activities: the first one consists of design and Intellectual Property (IP) with an emphasis on low-voltage and low-power design and in high efficient power distributions. The second activity consists of CMOS technology such as radiation effects and design workflows, but also CMOS-related assembly technologies such as CMOS wafer stacking and Through-Silicon Via (TSV). Finally, all these activities are performed on novel technology nodes that are going to be implemented in future experiments.

1.2.1 Technology transition

The ASICs design for HEP is currently implemented in 130 nm and 65 nm CMOS technologies, while the industry already reached 7 nm FinFETs technology and 5 nm prototypes. Similar behavior should occur for HEP ASICs to exploit the benefits of smaller transistors such as:

- Intrinsic higher density
- Intrinsic higher speed
- Intrinsic lower power consumption
- Higher radiation hardness

Beside these benefits, several drawback and challenges are also present such as:

- More expensive in terms of design and masks
- Higher complexity of design kits, layout for manufacturability, and CAD tools
- Increase of noise, variability, and mismatch parameters
- No reusability of old design techniques (for technologies with 3D shapes of transistors)

Moreover, current technologies are going to be obsolete and unused, thus, new technologies become mandatory for future experiments. For these reasons, WP5 started to explore the 28 nm CMOS technology, which represents the last planar transistor technology, as well as 16 nm FinFETs technology for next-generation ASICs. These new technologies are necessary to cope with requirements for future experiments in terms of higher spatial and temporal resolutions. Consequently, the 28 nm CMOS technology will be used in this thesis for a technology evaluation of the design.

1.3 Data readout and processing in HEP

A particle detector, also known as a radiation detector, is used in particle physics to find, track and possibly identify ionizing particles. It exploits the Bethe-Bloch formula which describes the stopping power of materials, corresponding to the mean rate of ionization energy loss over a distance for charged particles traversing a medium as a semiconductor. This formula well approximates the energy loss rate of standard physics particles [14]. Particle detectors are mainly built using silicon as semiconductor material because it presents a good Signal to Noise Ratio (SNR) compared to the others. Moreover, silicon devices and technologies are well developed and supported. These particle detectors are integrated as pixel arrays with a well-defined regularity. A pixel detector consists of a sensing component and a readout component. A typical pixel detector readout flow is depicted in figure 1.2. The impinging radiation, made of ionizing particles, creates electron-hole pairs inside pixels of a pixel detector. Thanks to the readout ASIC, the generated signal by these charge carriers is amplified and digitized. After the readout ASIC, digitized information is sent outside the chip to the Data Acquisition (DAQ) system for data processing. Such data processing can be performed in several ways such as through a Digital Signal Processor (DSP), an existing ready-made microprocessor, or even by a Field-Programmable Gate Array (FPGA). After the elaboration of information, data is submitted to the final user for further data analysis and examination.
The front-end circuit of this readout ASIC is depicted in figure 1.3. It consists of a first analog front-end starting with a charge-sensitive amplifier to convert

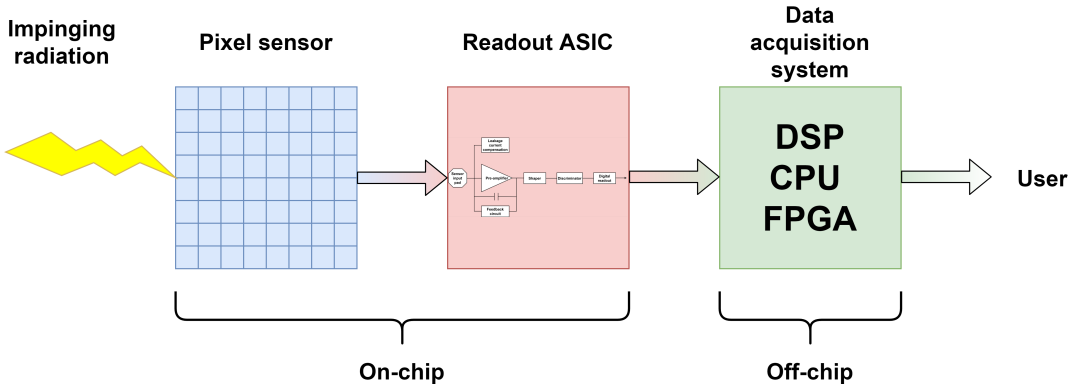

Figure 1.2: Block diagram of a typical pixel detector readout flow.

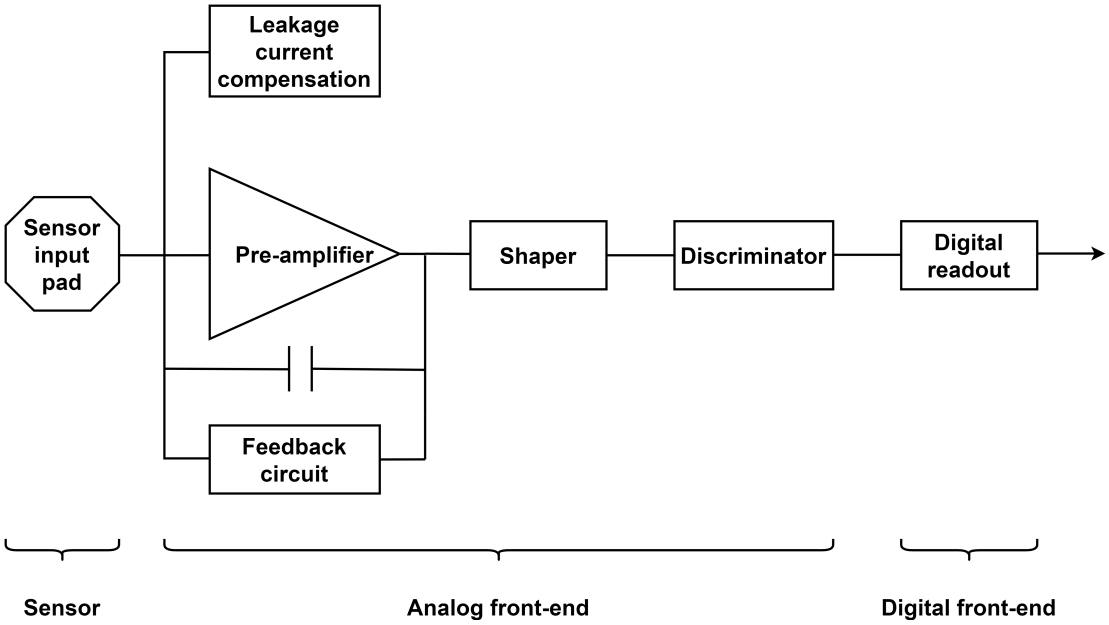

Figure 1.3: Front-end circuit of a typical readout ASIC for pixel detectors.

the input charge signal into a voltage. Such an amplifier is used as an integrator thanks to the coupling with a feedback capacitor. Moreover, its DC operating point is set by a feedback circuit which also clears the coupled feedback capacitor from signal charges once the signal is amplified. In some cases, an additional circuit for leakage current compensation might be connected to sink the sensor leakage. The next analog front-end block is a shaper used to filter out lower and higher frequencies, thanks to a band-pass filter, to increase the SNR of the signal. Moreover, the shaper also defines the output bandwidth of the pulse. Finally, the last analog front-end block is a discriminator used to compare the pulse coming

from the shaper with a customized threshold voltage. The discriminator provides a pulse width that is linearly proportional to the input charge signal. This pulse width is known as Time-over-Threshold (ToT) and it gives information about the amplitude of the signal related to the energy loss of the traversing ionizing particle [15]. Then, ToT is forwarded towards the digital front-end for further processing. In addition, another parameter is required to provide information regarding the time stamp: the Time-of-Arrival (ToA), which is implemented as a counter of clock cycles. Moreover, pixel coordinate information is also provided by each hit pixel for position measurements.

1.3.1 Literature review of HEP detector electronics

The state of the art presents several readout ASIC chips for different purposes in high energy physics at high rates. Most of them have been realized for the next HL-LHC upgrade in order to sustain higher hit rates, pile-up density, and total radiation dose. Among them, there are Macro-Pixel ASIC and Short Strip ASIC (MPA-SSA), RD53A, and Timepix4 as novel readout ASIC chips.

MPA-SSA

The Macro-Pixel ASIC and Short Strip ASIC (MPA-SSA) [16] are two Read-Out Chips (ROC) for the Pixel-Strip module for tracking applications, used to perform particle selection and data reduction through on-chip real-time particle discrimination with zero-suppression and trigger-free readout. This module is part of the CMS Outer Tracker upgrade for the HL-LHC. The MPA is implemented in a 65 nm CMOS technology and pixel arrangement consists of 118×16 pixels with a pixel size of 100×1446 µm^2. The SSA is a strip readout chip with a strip sensor that executes zero-suppression of particle hit coordinates and forwards this information to the MPA, which meanwhile performs the on-chip particle discrimination. Input data types of MPA consist of raw strip data or strip clusters while its output data types consist of pixel and strip clusters or stubs. Input and output data ports are made of several custom scalable Low Voltage Signaling (sLVS) transmitters and receivers. The input data port of the MPA consists of 9 custom-sLVS lines coming from the SSA at 320 Mbps reaching up to 2.88 Gbps; while the output data port consists of 6 custom-sLVS at 320 Mbps reaching up to 1.92 Gbps. The chip performs binary readout of silicon modules in continuous acquisition at the bunch-cross frequency of LHC events of 40 MHz. Moreover, the readout of the chip can be performed in two ways: the first one is a triggered readout for the full frame and the second one is a continuous trigger-free readout used for high transverse momentum information, p_T. The first readout type reaches a maximum latency of 12.8 µs and a maximum trigger rate of 750 kHz while the second readout type can

reach a lower latency of approximately 500 ns. The maximum sustainable hit rate is $53 \cdot 10^6$ hits/cm^2/s. In terms of radiation tolerance, MPA-SSA can tolerate a Total Ionizing Dose (TID) up to 200 Mrad. Finally, MPA-SSA can perform data reduction from 30 Gbps/cm^2 to 0.7 Gbps/cm^2 with a 95 % of particle discrimination efficiency.

MPA-SSA represents the first on-chip real-time particle discrimination readout useful to reduce throughput. It will be able to sustain the data rate of the HL-LHC upgrade. However, it provides an efficient but very specific processing that requires a complex module architecture and a fast chip-to-chip communication.

RD53A

RD53A [17] is a large-scale pixel chip intended to be used in the next upgrade for the HL-LHC. It is born from a collaboration between ATLAS and CMS experiments to sustain very high rates of approximately five times larger than the current LHC luminosity. Thus, new readout chips are required to work under a higher level of total dose and hit rates. RD53A is not the final chip but the prototype for future developments in terms of design methodology, radiation hardness, and characterization of the chip with a full-size pixel array. It sets down the basis for future developments of pixel readout ASICs needed in the HL-LHC. RD53A is implemented in a 65 nm CMOS technology for higher density and radiation tolerance. Pixel arrangement consists of 400×192 pixels with a pixel size of 50×50 µm^2 and the total chip size is 20×11.8 mm^2 and it is equivalent to half size of the production chip. RD53A manages to sustain a maximum hit rate of 3 GHz/cm^2 with an average of 75 kHz/pixel and a loss hit of less than 1 % at the maximum hit rate. Moreover, it is capable to work up to 500 Mrad of radiation. Charge resolution consists of at least 4 bits for ToT. The readout data is performed via parallel links, from 1 to 4, each of them at 1.28 Gbps and thus a maximum bandwidth of approximately 5.12 Gbps. All these RD53A specifications are well-suitable for the HL-LHC upgrade.

Even though RD53A represents the state-of-the-art of very high hit rates pixel detector readouts, it is not able to perform any kind of data discrimination or processing to lower throughput meaningfully.

Timepix4

Timepix4 [18] is a Hybrid Pixel Detector (HPD) readout ASIC presenting a design with tiling on 4 sides. Pixel array consists of 512×448 pixels where each pixel size is 55×55 µm^2 reaching up to 6.94 cm^2 of a sensitive area. These pixels are regrouped into Super-Pixels (SPs) of 2×4 pixels. Tiling becomes mandatory in pixel detectors with a large area such as in this case. It implements a 65 nm CMOS technology

and it used up to 10 metal layers. Timepix4 can operate in two modes: data-driven for tracking and frame-based (or photon counting) for imaging. In data-driven mode, each pixel transmits information towards the output only if the hit is higher than a configurable threshold. The output information from each pixel consists of two components: Time-of-Arrival (ToA) and Time-over-Threshold (ToT). Later on, additional information is latched along with two components regarding pixel coordinates. In the end, each pixel sends out event packets of 64 bits per hit. The maximum achievable hit rate of Timepix4 is approximately $3.58 \cdot 10^6$ hits/mm^2/s and the maximum pixel rate is up to 10.8 kHz/pixel. This occurs by using the maximum available readout bandwidth of 163.84 Gbps when using all 16 links each of them at 10.24 Gbps. ToA binning resolution in data-driven mode presents a value of 195 ps and ToA dynamic range is 1.6384 ms while ToT energy resolution reaches a maximum of 1 KeV. The second mode is frame-based and consists of putting two pixel-counters in each pixel with two different programmable depths of 8 and 16 bits. They work in a continuous read and write mode where one counter is counting whilst the second one is transmitting. In the end, each SP produces an event packet of 64-bit if pixel counters are 8-bit while it produces two event packets of 64-bit if pixel counters are 16-bit. This mode performs full-frame readout without pixel address and reaches a maximum count rate of $5 \cdot 10^9$ hits/mm^2/s. Moreover, latency depends on readout mode and application. Although, readout in full frame is performed in 726 µs.

Timepix4 shows the state-of-the-art timing resolution for HPD readout chips. It is used for several purposes such as high energy physics, time-of-flight mass spectrometry, Compon camera, and X-ray imaging for medical diagnostics, etc. where timing resolution represents an important parameter. In these fields, the utilization of flexible chips like Timepix4 is vital. Nevertheless, Timepix4 is not implemented in any experiment at CERN. In addition, it is not able to sustain high hit rates as RD53A and it does not perform any kind of data reduction as MPA-SSA.

1.4 Objective of the thesis

Thanks to the scaling down of CMOS technology, pixel dimensions decrease hence pixel density increases. This brings to an increment of spatial resolution. Moreover, another benefit consists also in increment of time resolution thanks to higher achievable frequencies. These benefits become challenging for the readout systems described before. In future experiments, due to higher luminosities, an increment of the amount of data coming from the pixel array might become unsustainable for the current pixel readout architecture hence their efficiency is going to drop

steeply. Furthermore, data toward the experiment back-end is going to increase as well, creating challenges to system bandwidth and off-chip data processing.
This thesis aims to solve such issues by proposing a novel system. This approach consists of placing an embedded processor directly on the chip, thus, closer to the pixel array. The objective is to perform a preliminary programmable data processing and readout, which is not present in the aforementioned chips, to decrease the amount of data to forward to the data acquisition system. Such a solution would provide a significant increase in flexibility, reusability, and modularity compared to the currently designed pixel imagers.

1.4.1 On-chip data processing for future experiments

On-chip data processing can be performed through an embedded processor placed either as a single processor at the periphery of the chip or as a distributed one based on Processing Elements (PEs) on the pixel array. Both of them come with several advantages and disadvantages.

Processing in periphery

Processing in periphery consists of integrating an embedded processor at the periphery of the pixel array for data elaboration as shown in figure 1.4. This embedded processor can be either a Reduced Instruction Set Computer five (RISC-V) core or something simpler.
During boot-up, memories are loaded through a Slow Control Interface and a reset interrupt is issued to the processor. All communication between the external world and the chip is handled by such a processor. This helps with programmability in sequencing commands and a fast context switching of configurations. In each pixel, during readout, counter values are written into memory, and; if required, the embedded processor at the periphery may process the raw pixel values to produce data to be transmitted out of the chip. The processor programs and triggers the High-Speed Data Interface, which is similar to a Direct Memory Access, to move data out of the chip through a serial interface. Data processing in the periphery presents benefits such as

- A relatively simple architecture
- Presence of many reusable blocks, i.e. system interconnect, memory, interfaces...

On the other hand, the following issue arises:

- Interface between the pixel array and main memory might become a bottleneck

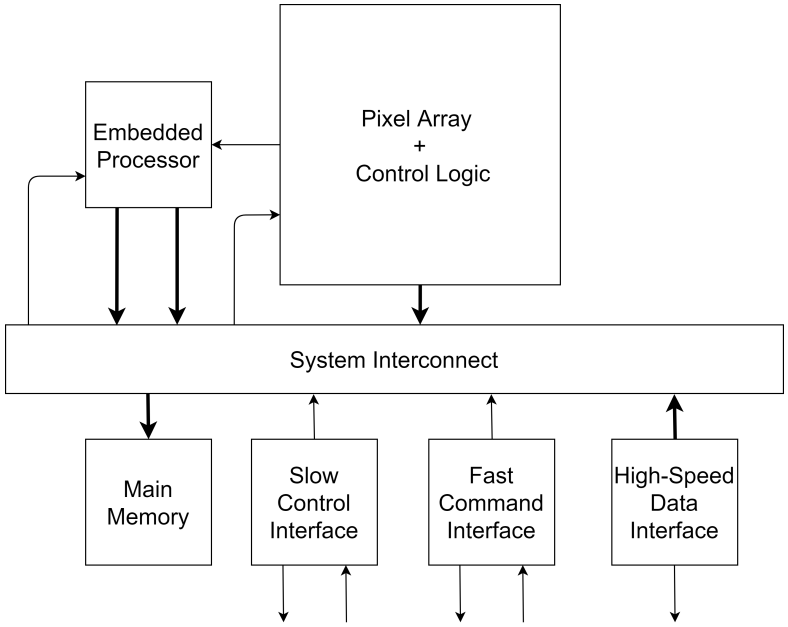

Figure 1.4: Configuration of data processing at the periphery.

In the processing in periphery, several challenges emerge such as the design of banked memories and interconnects to sustain a throughput of three bus masters: embedded processor, High-Speed Data Interface, and pixel array. Additionally, a focus on system modeling for architectural studies is required, for instance, to size interconnects, memories, etc. Moreover, a special firmware architecture, such as a client-server model, might be necessary for the optimization of the data acquisition system. Finally, challenges are also present in the design and physical implementation of such a model.

Processing on-pixel

The second option for on-chip data processing is on-pixel data processing. A schematic representation is provided in figure 1.5. It consists in placing PEs not at the periphery of a chip but directly on its pixel array. Each Processing Element might be placed for groups of pixels named Super-Pixels (SPs). These PEs can be RISC-V cores or something even simpler. They are interconnected with each other through a Network-on-Chip (NoC). Each PE is connected to its respective router of such a network. During boot-up, pixel configuration and PE memories are loaded. Upon a readout request, PE reads the pixel data and, if required performs a preliminary programmable data processing by communicating with neighboring PEs through the NoC. Moreover, the same network can also be used to output the

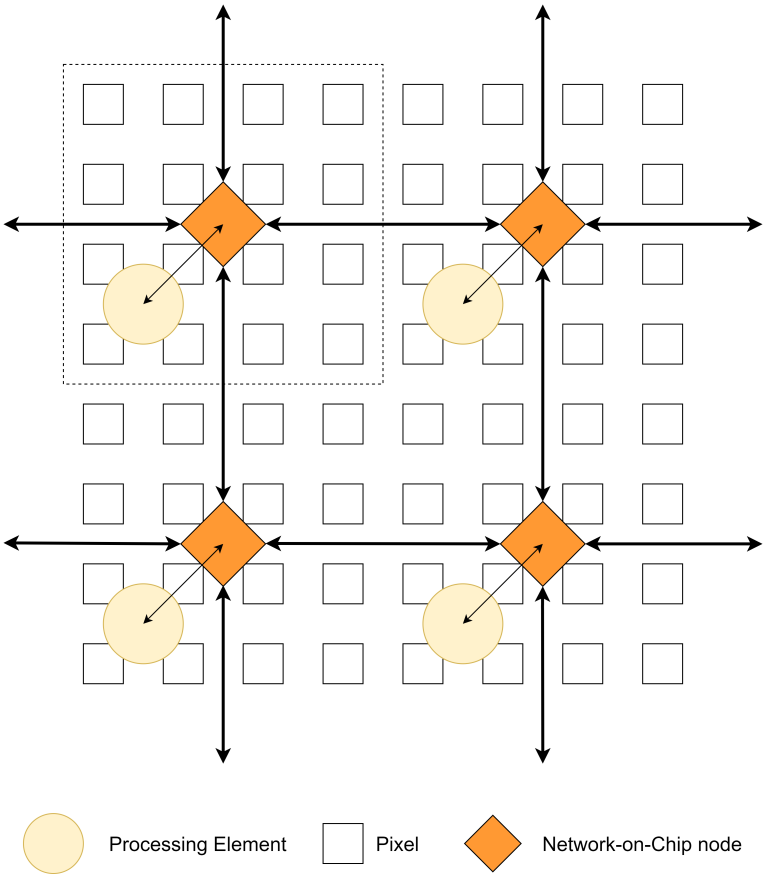

Figure 1.5: Configuration of on-pixel data processing.

elaborated data to forward to the off-chip DAQ system for further data processing. On-pixel data processing presents the following benefits:

- More processing close to the pixel: it lowers the throughput of data by forwarding off-chip only meaningful information.
- Potentially reduces power consumption: since the data processing is pushed closer to pixels, not all information from all pixels is sent to the periphery.

On the other hand, on-pixel data processing shows several challenges such as:

- Complex architecture: physical implementation becomes more difficult and congested.
- Area considerations: area of the Processing Element and router must be small enough to be accommodated on a group of pixels.

- Distributed pixel data processing algorithms: fast data computation and communication with neighboring PEs become mandatory.

This master's thesis focuses on setting down the background of the exploratory study of on-chip data processing with a focus on both Network-on-Chip and Processing Element. It establishes a starting point of on-chip data processing for future experiment applications.

1.4.2 Thesis organization

The thesis is arranged into 6 chapters:

- Chapter 1: the current chapter presents an overview of CERN and future upgrades, the working principles of particle detectors and readout systems with an emphasis on the state-of-the-art and related challenges. Finally, the objective and description of the project are provided.

- Chapter 2: it provides a conceptual modeling and analysis of a Network-on-Chip architecture implemented in on-chip processing. It also presents important figures of merit related to communication timing and its dependencies on several parameters.

- Chapter 3: introduces modelization, development, and optimization of performance, area, and energy-efficient Processing Elements. This is performed through a Application Specific Instruction-set Processor (ASIP) for the first time in HEP field. The ASIP has been customized with an AMBA APB system bus protocol interface towards the external world for future integration in a System-on-a-Chip (SoC) environment.

- Chapter 4: the first part of this chapter focuses on an application-specific test code for distributed pixel processing algorithm useful to test the PEs. While, the second part shows profiling results of these PEs regarding instructions, functional units, primitive operations, hazards, and code coverage. Both application-specific test code and profiling are executed on the ASIP Designer tool developed by Synopsys.

- Chapter 5: this chapter presents physical implementation and frequency optimization of the Processing Elements. They are implemented in a 28 nm CMOS technology and relevant figures of merit such as area occupation, power consumption, and achievable clock frequency are shown.

- Chapter 6: this chapter draws the conclusions of this thesis with a discussion on future improvements for on-chip data processing in High Energy Physics.

Chapter 2

On-chip communication architecture for data processing

In HEP at CERN, the particle collision rate is going to increase in future experiments [10]. Novel techniques are required to sustain high performance. The proposal of this thesis, as described in chapter 1, considers a novel approach to decrease off-chip data throughput by performing on-chip data processing. This system requires a specific on-chip communication architecture to interface the different modules used to execute such data processing. Integration of future technologies rises big challenges to communication architectures. Latency, network traffic, cost, power consumption, and scalability become crucial parameters for the choice of an adequate communication architecture. Among several possible network structures, Network-on-Chip presents the best trade-off among the aforementioned parameters [19].

This chapter establishes an exploratory study of a Network-on-Chip communication architecture applied to on-chip data processing in HEP and it sets the analytical background for future applications.

2.1 On-chip communication networks

The most common on-chip communication networks are the following: Point-to-Point (P2P), crossbar, and bus-based.

In a P2P, every node is connected to every other. This network gives the best

performances in terms of low latency and low contention but it is not so practicable due to its high cost and especially non-scalability [20, 21]. For instance, a novel node introduced in the network needs to be connected to every other node already existing and the routing of wires becomes difficult. Thus, it results in a dense network and interconnections start to dominate dynamic power dissipation.

The crossbar on-chip communication network allows every node to be connected to every other in a crossbar configuration and it gives quite similar benefits as the P2P. On the other hand, it becomes impracticable as the number of nodes grows and particularly if the distance from one node to another is longer than the clock period that is intended to use [22, 23].

Finally, the bus-based on-chip communication network is simpler and cheaper for a small number of nodes since there are no more dedicated interconnections between each node thus the complexity is reduced and modularity increased. On the other hand, the bandwidth is limited, and delay and energy consumption increase significantly since bus drivers have larger capacitive loads. Therefore, this architecture is not scalable to a large number of communicating nodes and it also saturates fast due to a high contention [20, 21].

Due to these several issues of the aforementioned networks, a new on-chip communication network is necessary: the Network-on-Chip, which is deeply discussed in section 2.2.

2.2 Network-on-Chip

Differently from P2P, crossbar, and bus-based networks, a Network-on-Chip connects two nodes via a packet-switching network based on routers. Each router is connected to the neighboring ones through bi-directional P2P connections. Because of this, the additional integration of nodes is quite simple. A router consists of an input buffer for arriving packets, a switch implementation, a control logic to guide the packet through the router, and finally an output buffer for outgoing packets. For the same application, a communication architecture based on Network-on-Chip shows performances similar to the P2P [21]. For an increasing number of nodes, the area occupation of P2P increases exaggeratedly whilst NoC shows a modest area overhead similar to the bus-based network. Moreover, power consumption is lower than P2P and bus-based and way more scalable than these two [20]. NoC is an advantageous communication network because it is design-friendly, easily scalable, flexible, and predictable [19].

2.2.1 Network-on-Chip for data processing

In this case study, as described in chapter 1, a pixel matrix of a pixel detector is subdivided into Super-Pixels (SPs). For each SP, a Processing Element (PE) is placed to read pixel data upon a readout request and, if required, to perform preliminary data processing. Such data processing requires communication with neighboring PEs because an event might involve several SPs. The information sharing among neighboring PEs to perform on-chip data processing is executed by a specific Network-on-Chip as depicted in the figure 1.5. It is also important to mention that such a network can also be used to push data off-chip as presented in [24]. In particular, PEs are connected directly to their local routers which send packets through a network interface that connects each PE to the general network. Each packet is sent from a source node to a destination one and it goes through several nodes. During the servicing of the packet, it needs also to go through input channels at the beginning, routers, and switches, and so the additional timing given by these should not be neglected.

A particle detector is characterized by a regular structure of its pixel array. To avoid breaking this regularity, the assumed Network-on-Chip architecture has a mesh topology.

In pixel detectors involving PE and relative NoC node for each SP, performances, area occupation, and power consumption depend on several parameters. The size of each node, thus the number of pixels for each SP, and the pixel matrix dimension influence all these three figures of merit. The particle rate sets performance limitations and possible failures of the network. Finally, the operating frequency has a huge impact on performance but also power consumption.

2.2.2 Packet switching technique

Concerning a Network-on-Chip, an important decision to take is the technique of packet switching to apply to such a network. Packets transfer mode through switches and routing decisions change among several different switching techniques. There are mainly two groups of switching techniques: circuit switching and packet switching.

The circuit switching technique implements a physical circuit that goes from one node, the source, to another one, the destination, and without any mediation in-between all the packets are transferred.

The packet switching technique contains several modes: wormhole, virtual cut-through, and store-and-forward switching. For this case of the NoC study, a wormhole switching technique will be used, and it consists of dividing the packet into sub-units, named flits, that are sent one after the other in a pipelined fashion throughout the network. The first flit of each packet, named header flit, has all the routing information to go from the source node, throughout the whole network towards the destination node. Since all the flits move in a pipelined fashion, due to congestion, the header flit might be stopped at a certain position in the network and the same occurs to all its following flits at their respective positions. The wormhole switching is usually a good switching technique for a Network-on-Chip with high dynamic traffic [25]. Moreover, it does not require the presence of large buffers at each node since the packet is sub-divided into flits and presents a relatively low latency. This is why the wormhole technique is also the most used one for Network-on-Chip [21]. Moreover, in this kind of technique, latency is dominated by the packet size.

The other two techniques, store-and-forward, and virtual cut-through switching are quite similar. The first one, which is self-explanatory, stores the packets at halfway nodes and these are sent to the next node only if both of the following conditions are respected: the output channel is free and this latter node has a buffer with enough space to store the incoming packet.
On the other side, the virtual cut-through switching only requires the first condition to be fulfilled and thus improves the problem related to latency in the store-and-forward technique [21].

2.3 Network-on-Chip analysis

In this case study, as mentioned before, the Network-on-Chip architecture is applied to a pixel array. A NoC node is placed at each SP with its corresponding PE. Moreover, in this application, a simple analysis has been conducted considering a packet that moves by a maximum of two positions, and then it gets reprocessed again in the latter PE. In the worst-case scenario, when a particle hits the corner of a SP, the PE requests the information from the adjacent SPs, so packets coming from lateral SPs move by one position and by two for packets coming from the diagonally opposite SPs. The probability of occurrence of such a scenario strongly depends on the size and number of SPs in the pixel array. The dimension of this SP has been varied during the network analysis to see its effects. The maximum traffic on a single port of a NoC node router is shown in figure 2.1. So, even though it is a simplification, this approximation suits well for this application.

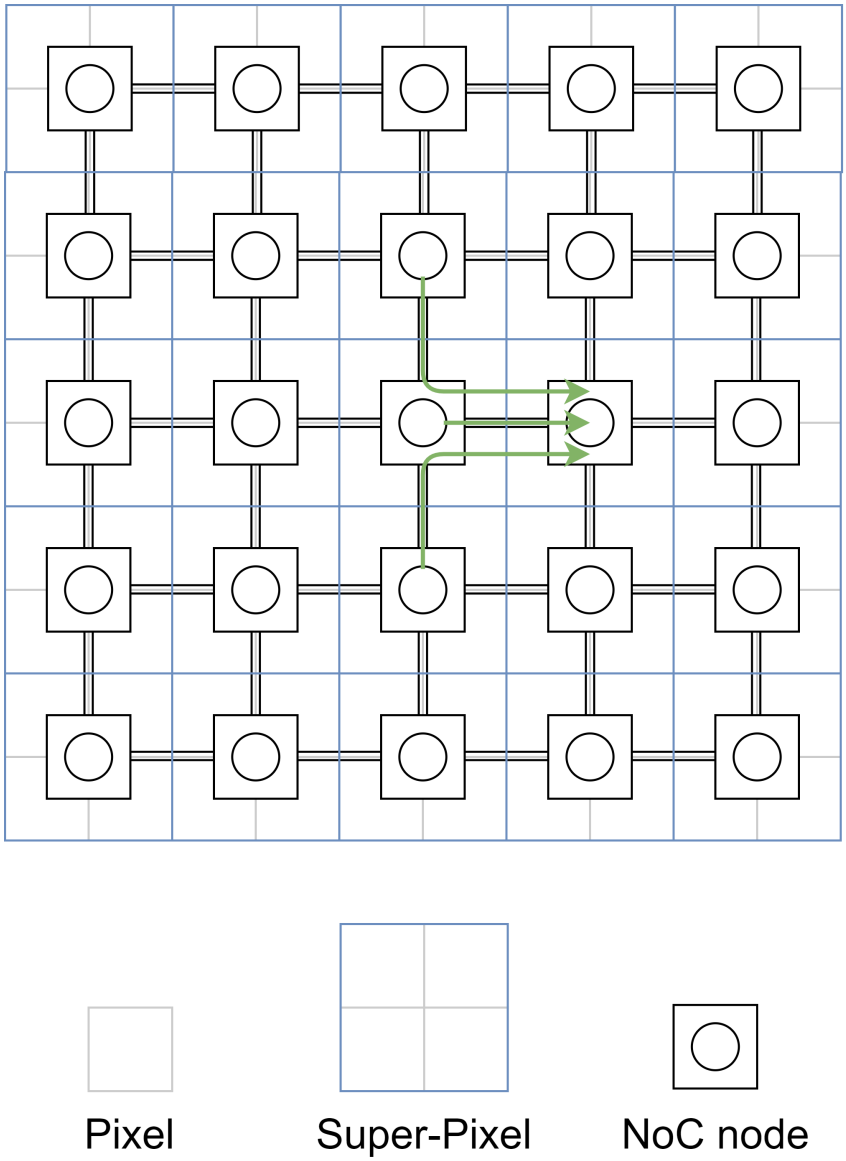

Figure 2.1: Maximum traffic on a Network-on-Chip applied to a pixel array.

In this case study, as shown in the figure 2.2, each router has four channels with some First-In-First-Out (FIFO) buffers. In order to reach a final formula for the latency in a wormhole switching technique Network-on-Chip, the starting point is the packet's service time which is computed as in [21]:

$$T = H_s + \frac{S}{W} \tag{2.1}$$

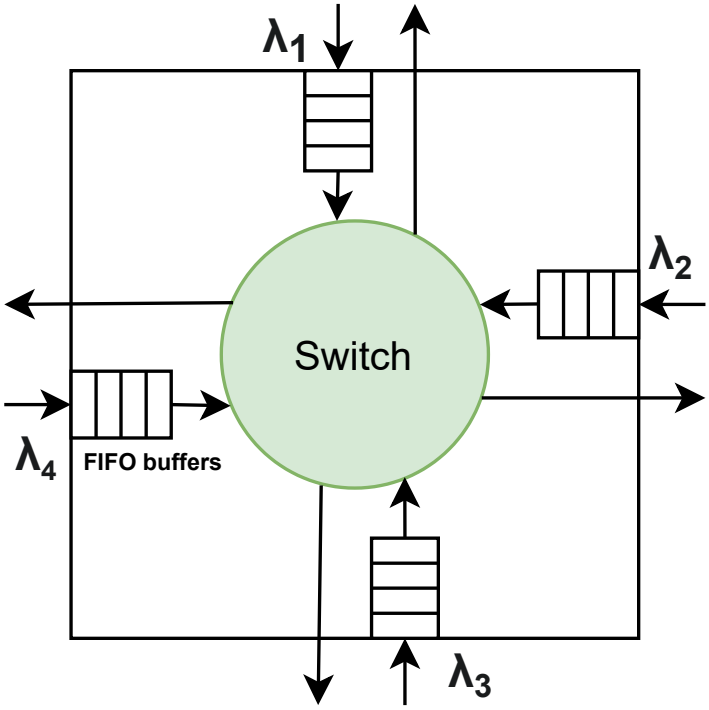

Figure 2.2: Network-on-Chip router model with a switch, FIFO buffers and four channels.

Where,

- T is the packet's service time [cycle]
- H_s is the service time for the header flit [cycle]
- S is the packet's size [bit]
- W is the bus-width [bit/cycle]

To go more into depth, for this case study, H_s is equal to two cycles where one cycle is due to service time and one due to routing of the packet with the header flit. In general, a Poisson distribution for the header flit arrival time is a good approximation in this kind of network analysis at not too high traffic rates [21].

The ratio between S, the size of the packet, and W, the bus-width, gives an interesting parameter known as bandwidth delay. This is defined as the amount of time it takes for a sender to transfer a packet over the network. This parameter takes into account both the size of the packet and the available bus-width.

As explained in chapter 1, each pixel of a particle detector usually outputs to the readout electronics the following information: coordinate of the pixel, Time-of-Arrival (ToA) which provides information about the time stamp, and Time-over-Threshold (ToT) of the hit which gives information on the amplitude of the signal depending on the particle's energy loss [15].
VeloPix, which is an application-specific integrated circuit coming from Timepix and Medipix chips, is used in the LHCb experiment for hybrid pixels as a readout chip. This chip provides as output 9 bits for ToA, 4 bits for ToT and 4 bits for pixel's position inside the SP [26].
Similarly, Timepix3 provides 10-bit ToT and 18-bit ToA [27]. It is noticeable how the packet size changes among different chips. Moreover, the system does not transmit all the information from all pixels to the network thus the size and number of packets transmitted vary depending on the kind of event.

Furthermore, no restriction has been applied to the bus-width W to give more freedom to such a parameter in the bandwidth delay. The CMS tracker hybrid pixel detectors send small size information packets, thus, the required bus-width for binary readout in these detectors is narrower [28]. On the other hand, the family of Timepix chips [29, 30, 31, 18] provides data packets with a larger size and so, the required bus-width is wider. For instance, a packet size of 64 bits might be sent either on bus-width of 1 bit/cycle, equivalent to a bandwidth delay of 64 cycles, or 64 bit/cycle, equivalent to a bandwidth delay of 1 cycle. On the other side, by keeping the same bus-width, packet size might change depending on the application.

Finally, a sweep has been performed on $\frac{S}{W}$ from 1 cycle to 64 cycles in order to take the asymptotic value of achievable latency or frequency. These values have been chosen for parameter space exploration.

The service time of the packet previously computed is necessary in order to compute the average number of packets at each input buffer of the router, which is well expressed in [21] by the following formula:

$$N = (I - T\Lambda C)^{-1}\Lambda\bar{R} \tag{2.2}$$

Where,

- N is the average number of packets at each input buffer of the router
- I is the identity matrix
- T is the packet's service time [cycle]
- Λ is the traffic arrival rate [1/cycle]

- C is the contention matrix
- $\bar{R}$ is the residual time

N is a column vector defined as $N = \begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix}$ where each row represents each of the four input channels present in the router.
The identity matrix is $I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.
T has already been explained previously in the formula 2.1.
The residual time is equal to $\bar{R} = R \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ and R is the residual packet waiting time equal to one for simplicity.

In order to compute the contention matrix C, the forwarding probability matrix F is necessary first.
The latter one is defined as [21]:

$$F = \begin{bmatrix} 0 & f_{12} & f_{13} & ... & f_{1P} \\ f_{21} & 0 & f_{23} & ... & f_{2P} \\ ... & ... & ... & ... & ... \\ f_{P1} & f_{P2} & ... & ... & 0 \end{bmatrix}$$
$$f_{ij} = \frac{\lambda_{ij}}{\sum_{k=1}^{P} \lambda_{ik}}, 0 \leq i, j \leq P \tag{2.3}$$

Where f_{ij} gives the probability for a packet to go from a channel i and to exit at channel j and λ_{ij} is the traffic arrival rate expressed in [21] as follows:

$$\lambda_{ij} = \sum_{\forall s} \sum_{\forall d} x_{sd} \cdot \mathcal{R}(s, d, i, j) \tag{2.4}$$

This formula considers all the packets that go from a source node s to a destination node d across a router passing through an input channel i toward an output channel j of it. This is expressed by the routing function $\mathcal{R}(s, d, i, j)$ which is equal to one when the packet respects the previous routing condition otherwise it is equal to zero. Furthermore, x_{sd} is the packet transmission rate from node s to node d. Since

in this case a packet can move only by a maximum of two positions as previously shown in figure 2.1, it goes only through one router and so all the λ_{ij} are equal to x_{sd}.

Considering a router with four ports, for simplicity, it is assumable that a packet at each port has the same probability to be transmitted to any of the remaining three ports. From equation 2.3, it is noticeable that all the elements on the diagonal of the matrix are equal to zero such that $f_{ii} = 0$. This is because, in this case study, a packet cannot be forwarded to the same port from which it arrived. From these hypotheses, equation 2.3 results as:

$$F = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1/3 & 0 & 1/3 & 1/3 \\ 1/3 & 1/3 & 0 & 1/3 \\ 1/3 & 1/3 & 1/3 & 0 \end{bmatrix}$$

From the forwarding probability matrix F, the contention matrix C can be computed by using the following formula [21]:

$$\begin{aligned} 0 \leq i, j \leq P, i \neq j, c_{ij} = \sum_{k=1}^{P} f_{ik} \cdot f_{jk} \\ i \neq j, c_{ii} = 1 \end{aligned} \tag{2.5}$$

Thus, the contention matrix is equal to:

$$C = \begin{bmatrix} 1 & 2/9 & 2/9 & 2/9 \\ 2/9 & 1 & 2/9 & 2/9 \\ 2/9 & 2/9 & 1 & 2/9 \\ 2/9 & 2/9 & 2/9 & 1 \end{bmatrix}$$

Finally, the traffic arrival rate Λ is a diagonal matrix equal to:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{bmatrix} \tag{2.6}$$

In this application, Λ is a 4×4 matrix because the router considered has four ports. Since the considered system is isotropic, it is easily remarkable that all the lambdas are equal, thus, $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4$. Using the formula 2.4, these values can be expressed by the following formula [21]:

$$\lambda_j = \sum_{\forall s} \sum_{\forall d} x_{sd} \cdot \mathcal{R}(s, d, j)$$

Since λ_j is the arrival rate at the input buffer of channel j, this is proportional to the traffic arrival rate λ_{ij} and equal to:

$$\lambda_j = 3 \cdot \lambda_{ij} = 3 \cdot x_{sd}$$

This is because, in this specific application as it is shown in figure 2.1, for each input buffer of a router there might be a maximum of three packets coming from three different routers. This has been deducted considering the already mentioned assumption where a packet moves by a maximum of two positions.

Furthermore, the packet transmission rate x_{sd}, from source node s to destination node d, strictly depends on the specifications of the required Network-on-Chip and it is expressed as follows:

$$x_{sd} = \frac{particle\ rate \cdot (pixel\ dimension \cdot pixel\ matrix)}{\frac{pixel\ matrix}{NoC\ node\ dimension} \cdot operating\ frequency}\ [\frac{packet}{cycle}] \tag{2.7}$$

An example of specifications used to compute x_{sd} are listed in table 2.1. Notice that these values strongly depend on the application.

Table 2.1: Summary table for NoC specifications

Particle rate	$1 \longrightarrow 10^{11}$ [cm$^{-2}\cdot$ s^{-1}]
Pixel dimension	55×55 [µm^2] (pixel pitch = 55 µm)
Pixel matrix	256×256
NoC node dimensions	1×1, 4×4, 32×32 and 256×256
Operating Frequencies	320 and 80 [MHz]

Considering a maximum hit rate of $40 \cdot 10^6$ cm$^{-2}\cdot$ s^{-1} in Timepix3 chip [31], during the analysis of this case study, a sweep on the particle rate has been performed from 1 cm$^{-2}\cdot$ s^{-1} to 10^{11} cm$^{-2}\cdot$ s^{-1} to see the saturation effects given by it. These values have been taken for the sake of parameter space exploration. This parameter influences significantly the performance of the NoC architecture.
The considered pixel pitch and pixel matrix are respectively 55 µm and 256×256 as in Timepix3 chip [31].
Moreover, four different NoC node dimensions have been analyzed: 1×1, 4×4, 32×32 and 256×256, in order to see the different behavior of latency depending on this parameter.

Furthermore, the same analysis has been conducted considering two different operating frequencies, a fast one at 320 MHz and a slower one at 80 MHz. These values have been considered as multiples of the typical bunch crossing frequency of 40 MHz in the LHC experiment [32].

Figures 2.3 and 2.4 show the representation of two NoC topologies that have been analyzed. The first one has a NoC node dimension of 4×4 pixels which means a NoC matrix of 64×64 pixels considering the full pixel matrix of 256×256. While the second one has a NoC node dimension of 32×32 pixels thus a NoC matrix of 8×8 considering the same pixel matrix as the first topology. Similarly, in 1×1 NoC node dimension, a node is applied to each pixel of the pixel array whilst in 256×256 there is only one node for the whole pixel array. The latter is equivalent to performing on-chip data processing in the periphery as shown in chapter 1. The last two topologies are borderline cases.

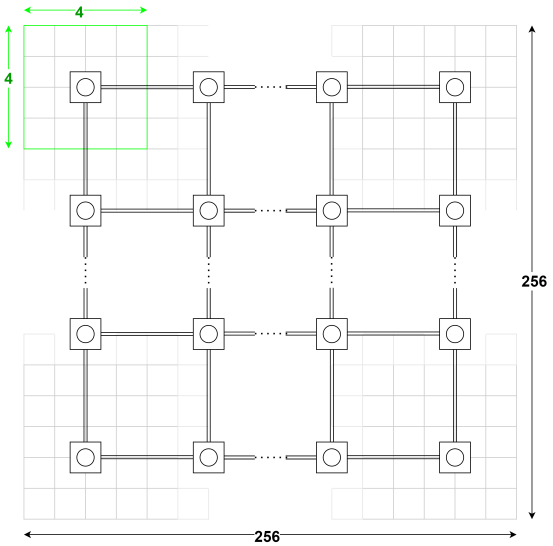

Figure 2.3: 4×4 pixels NoC node dimension.

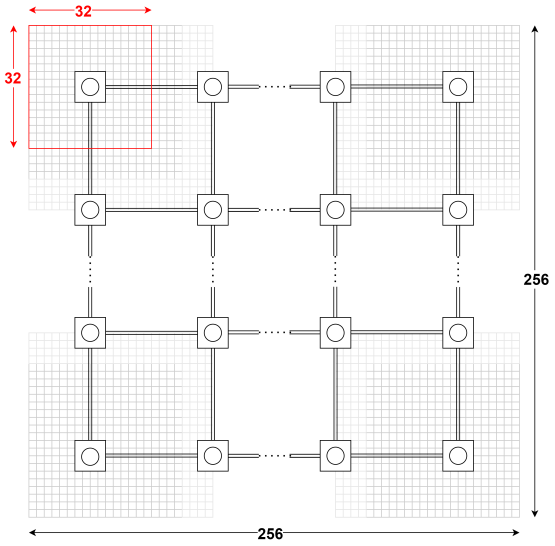

Figure 2.4: 32×32 pixels NoC node dimension.

Starting from equations 2.4 of traffic arrival rate λ_{ij} and 2.2 the average number of packets at each input buffer of the router N_{ij}, applying Little's theorem [33], it is possible to get the average waiting time for each channel buffer of each router [21]:

$$W_{ij} = \frac{N_{ij}}{\lambda_{ij}} \tag{2.8}$$

This is necessary in order to compute the average latency from source node s to

destination node d as follows [21]:

$$L_{sd} = W_s + \sum_{(i,j)\in\prod_{sd}} (W_{ij} + T) \tag{2.9}$$

Where, W_{ij} is the average waiting time for each channel buffer of each router and T is the average service time as previously computed. The summation term considers all the input buffers of the routers that a single packet must go through to travel from a source node s to a destination node d. W_s is the queuing delay the packet experiences at the source node s and it is set to one cycle for simplicity. From this, it is possible to compute the overall average latency for a packet as [21]:

$$L = \frac{1}{\sum_{\forall s,d} x_{sd}} \sum_{\forall s,d} x_{sd} \cdot L_{sd} \tag{2.10}$$

Consequently, since the packet moves by a maximum of two positions and all the various x_{sd} from any source node s to any destination node d are the same, equation 2.10 can be simplified as:

$$L = L_{sd}$$

Moreover, in equation 2.9, it is possible to simplify the summation term because a packet moves by a maximum of two positions and so through a single router. This is true by assuming that a packet is serviced at the output of its corresponding router node. So, under this assumption, the previous equation gets simplified even more as follows:

$$L = L_{sd} = W_s + (W_{ij} + T) = W_s + (\frac{N_{ij}}{\lambda_{ij}} + H_s + \frac{S}{W}) \tag{2.11}$$

Equation 2.11 is the starting point of the Network-on-Chip analysis explained in the next section.

2.4 Network-on-Chip results

Four different cases have been analyzed by changing the following parameters: NoC dimension and operating frequency. For all these cases sweeps on the particle rate and bandwidth delay have been performed. These cases are listed here:

- NoC node dimension 1×1 pixels and operating frequency 320 MHz
- NoC node dimension 1×1 pixels and operating frequency 80 MHz
- NoC node dimension 4×4 pixels and operating frequency 320 MHz
- NoC node dimension 4×4 pixels and operating frequency 80 MHz
- NoC node dimension 32×32 pixels and operating frequency 320 MHz
- NoC node dimension 32×32 pixels and operating frequency 80 MHz
- NoC node dimension 256×256 pixels and operating frequency 320 MHz
- NoC node dimension 256×256 pixels and operating frequency 80 MHz

In figures 2.5 and 2.6, the previously listed topologies are well summarized and marked with different colors in the legend on the top. The first graph presents a 3D plot of latency depending on particle rate and bandwidth delay while the second graph shows the same dependency but on a 2D plot.

For all the topologies, at a fixed particle rate, latency increases linearly with the bandwidth delay. On the other hand, at a fixed bandwidth delay, the increment of latency versus the particle rate is less significant up to a certain value after which it starts to suddenly diverge. Before such a divergence, latency values are almost similar among all the analyzed cases. The difference between one case and another is the particle rate at which latency starts to diverge and this is well shown by the different colors in figures 2.5 and 2.6. Such a value is named the cut-off particle rate and, in this case study, it corresponds to the maximum particle rate the network can sustain in the worst-case scenario. Considering that during this analysis a sweep on bandwidth delay has been performed, the worst-case scenario corresponds to its highest value, thus, equal to 64 cycles. In the same figure, it is also shown how the cut-off particle rate changes depending on the NoC node dimension and the operating frequency.

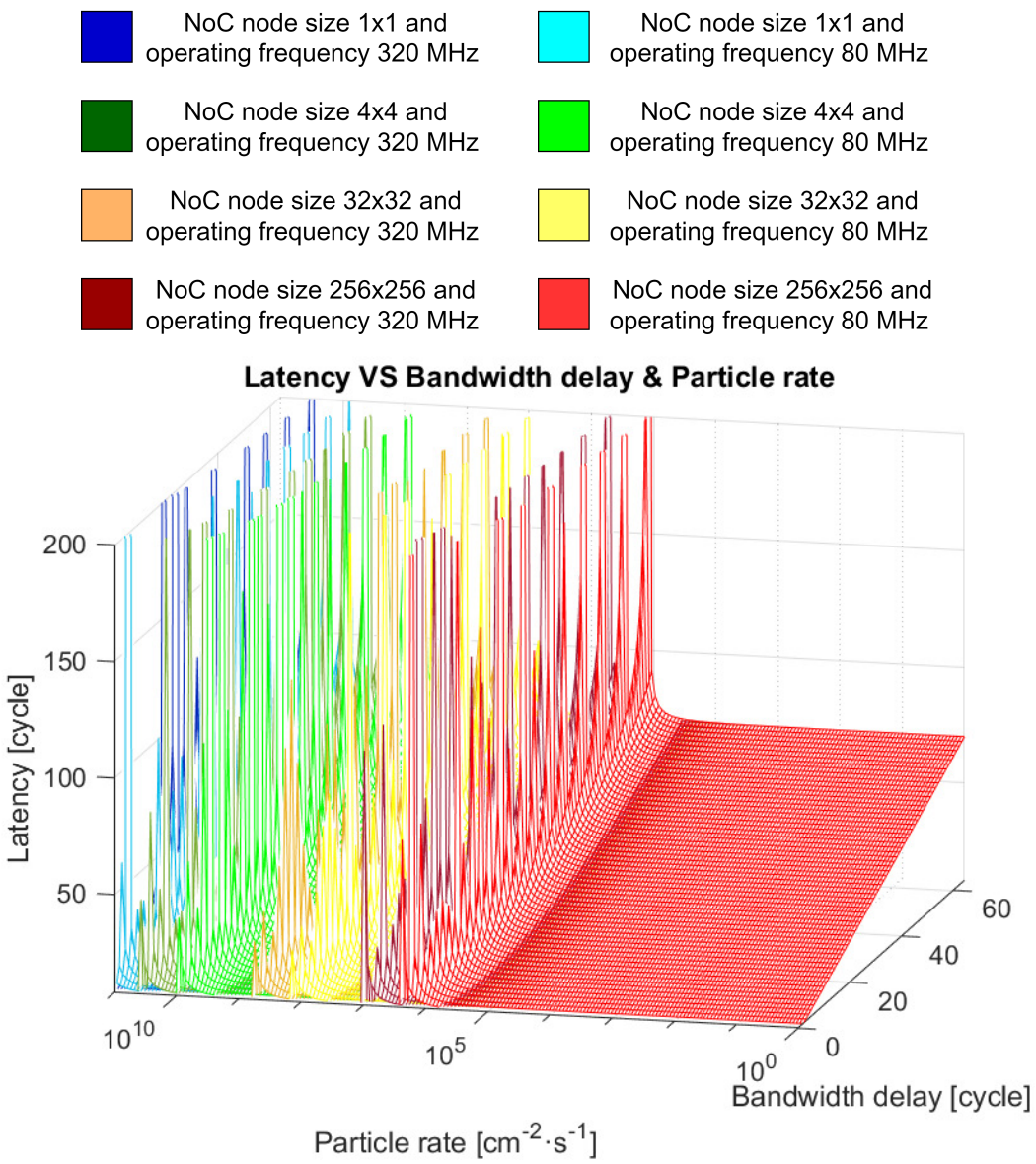

Figure 2.5: 3D graph of latency depending on bandwidth delay and particle rate with the legend on the top.

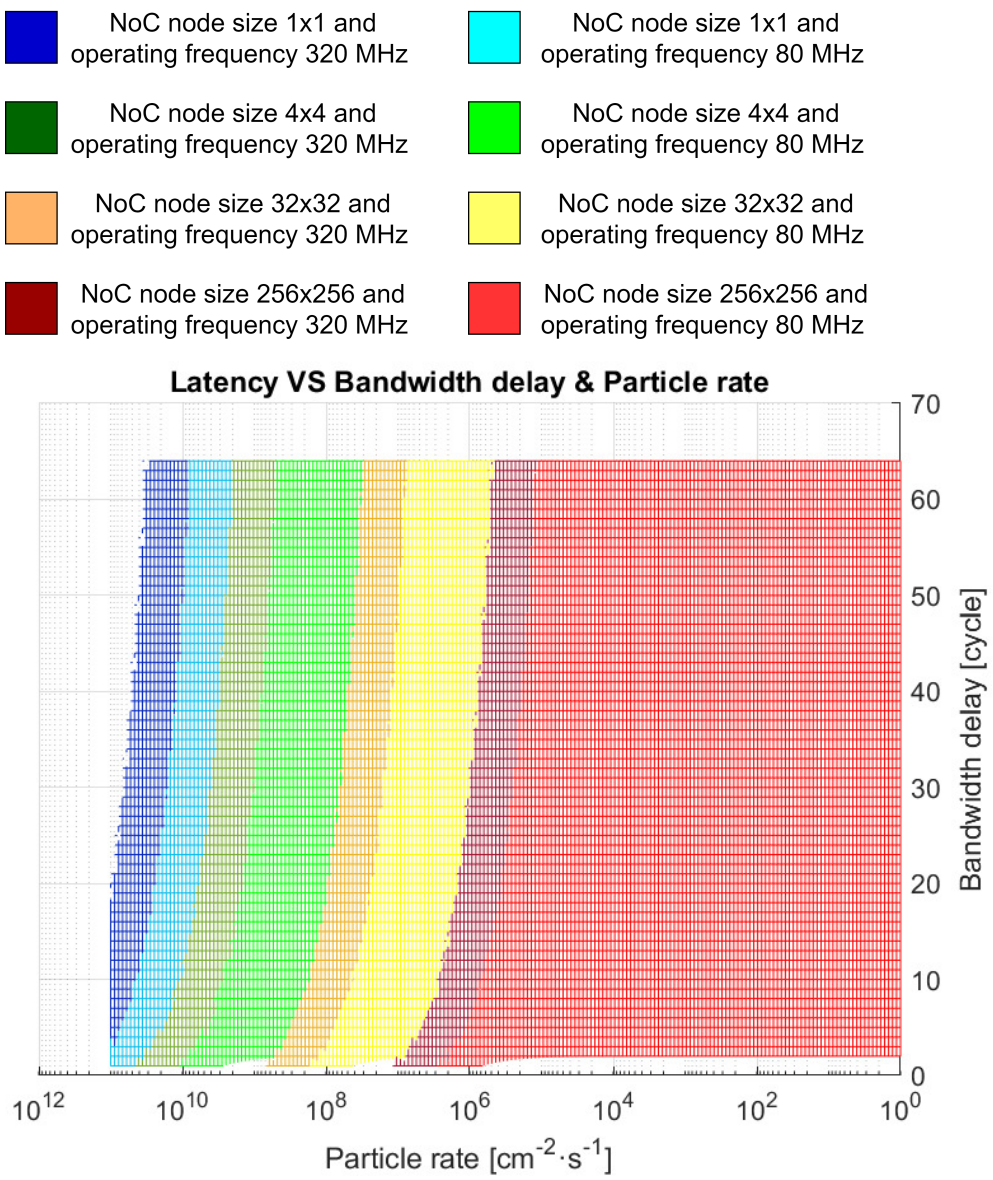

Figure 2.6: 2D graph of latency depending on bandwidth delay and particle rate with the legend on the top.

The cut-off particle rate has been summarized in the table 2.2:

Operating frequency / NoC node size	80 MHz	320 MHz
1×1	$\sim 8 \cdot 10^{9}$ [cm$^{-2}\cdot$ s^{-1}]	$\sim 3 \cdot 10^{10}$ [cm$^{-2}\cdot$ s^{-1}]
4×4	$\sim 5 \cdot 10^{8}$ [cm$^{-2}\cdot$ s^{-1}]	$\sim 2 \cdot 10^{9}$ [cm$^{-2}\cdot$ s^{-1}]
32×32	$\sim 7 \cdot 10^{6}$ [cm$^{-2}\cdot$ s^{-1}]	$\sim 3 \cdot 10^{7}$ [cm$^{-2}\cdot$ s^{-1}]
256×256	$\sim 1 \cdot 10^{5}$ [cm$^{-2}\cdot$ s^{-1}]	$\sim 5 \cdot 10^{5}$ [cm$^{-2}\cdot$ s^{-1}]

Table 2.2: Cut-off particle rate for the eight analyzed cases.

From the figure 2.6 and the table 2.2, it is noticeable how the cut-off particle rate changes among these topologies. For the same NoC node dimension, topologies implementing 320 MHz show a cut-off particle rate approximately four times larger than the ones implementing 80 MHz. On the other hand, at a fixed operating frequency, the cut-off particle rate increases by decreasing the NoC node size through a power function. These two trends are shown in the log-log plot in figure 2.7.

From the figure 2.7 and table 2.2, it is also possible to derive the cut-off occupancy, which gives information about the maximum data rate and flux a pixel detector can sustain. This is obtained by dividing the cut-off particle rate by the bunch crossing frequency of 40 MHz.

It is noticeable from table 2.2 and figure 2.7, that for an increasing NoC node dimension, the NoC matrix will be smaller thus the packet transmission rate from source node s to destination node d increases. This causes a higher level of congestion and saturation of the packets across the whole network and so a lower cut-off particle rate and occupancy. On the other hand, a higher NoC node dimension brings to a lower complexity during the hardware implementation of such a network due to the lower number of NoC nodes and routing channels.

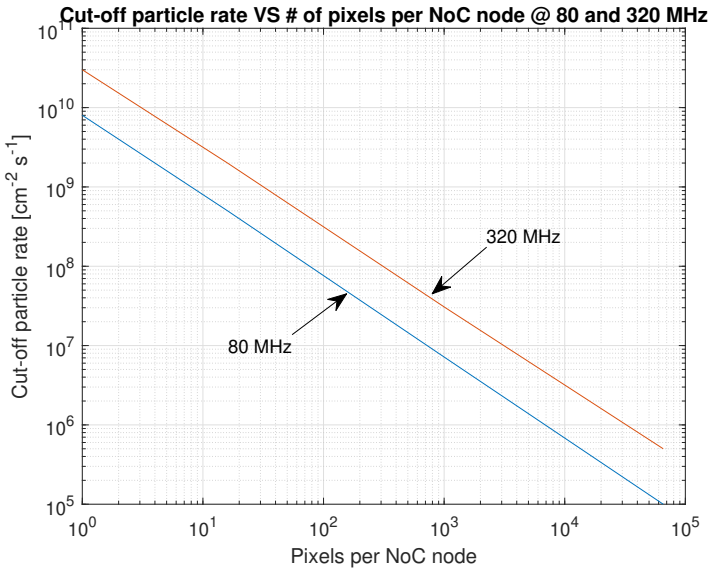

Figure 2.7: Log-log plot of cut-off particle rate vs the number of pixels per NoC node at operating frequencies of 80 MHz and 320 MHz.

A visible trend is also the increment of the possible cut-off particle rate while the NoC node dimension decreases. This might lead to suppose that the optimal case is reached when the NoC node size is equal to a single element 1×1, equivalent to a Network-on-Chip node, and its corresponding PE, for each pixel in the pixel array. This is only an idealistic supposition but not feasible at all since it is not possible to put a PE and its corresponding NoC node for each pixel due to area limitations during the implementation process. On the other hand, the NoC node dimension might be pushed up to 256×256 pixels, equivalent to a single NoC node as a matrix applied to the whole pixel array. This denotes that information from all pixels goes through this single node towards the periphery and data processing does not occur anymore on-array.

2.5 Final considerations

As expected, the achievable particle rate increases by increasing the operating frequency and by decreasing the NoC node dimension. Both borderline cases for NoC node dimension equal to 1 × 1 and 256 × 256 are not feasible. The first one is due to area constraints and the second one is due to low performance. Moreover, the second case does not represent anymore a Network-on-Chip and it is equivalent

to performing on-chip data processing in the periphery. Furthermore, high operating frequencies also impact negatively the power consumption of the whole system.

As previously described in chapter 1, the considered case study involves a Network-on-Chip node and a Processing Element for each Super-Pixel. From the analysis and results described in this chapter, a suitable exploration study of Network-on-Chip applied to pixel detectors has been established for future experiments. This chapter sets down the basis for latency computation depending on important parameters such as particle rate, the packet size of information, available bus-width, operating frequency, Network-on-Chip node size, and matrix dimension.

An analysis of the considered Processing Element, described in chapter 3, is necessary to complete the exploration presented in this chapter. Information regarding area occupation, achievable operating frequency, and power consumption will be provided from the physical implementation of such a PE in chapter 5. Moreover, it might put a constraint on routing resources that consequently limits the bus-width of the Network-on-Chip. Furthermore, information regarding the necessary number of cycles to perform data processing is present in chapter 4, to better specify parameters such as: queuing delay, packet service time, required bus-width, and average packet size.

Finally, from all this information, it is possible to achieve optimal values of the Network-on-Chip dimension and operating frequency to have the best trade-off in terms of area occupation, power consumption, and latency.

For more details, the analysis presented in this chapter, and applying formula 2.11, has been conducted through a script on MATLAB that can be found in appendix A.

Chapter 3

On-chip Processor for data processing

Data processing, or information processing, is the manipulation and modification of collected data to create useful information. Several functions might be implied to perform data processing such as: validation to verify that data provided is accurate and pertinent, sorting to re-arrange data by specific order, summarization to distill complex information down to the essentials, aggregation of different data sets, analysis and reporting of information, and finally, classification of data into several categories [34]. Such data processing units are nowadays present as integrated circuits in computer processors named microprocessors.

A microprocessor is an integrated circuit able to perform the aforementioned data processing functions through specific arithmetic and logic units. It receives inputs as binary data and outputs processed information also in binary representation [35]. Very Large-Scale Integration (VLSI) made possible the scaling down of microprocessors' dimension and their cost. Today, microprocessors are used everywhere and for everything. Their usage depends on the application in terms of the trade-off of several parameters such as performance, area occupation, and power consumption. Computers might require high performance regarding speed and computational complexity while, on the other hand, portable electronics concern more about low power consumption and area occupation. Due to the high demand for several kinds of processing, microprocessors are mainly developed with a general-purpose design able to perform various tasks on the same physical entity. On the other hand, some microprocessors are developed as special-purpose entities for specific processing such as signal processing in Digital Signal Processors (DSPs) or image processing in Graphics Processing Units (GPUs) [35].

Scaling down of CMOS technology has a huge impact on microprocessors. On one hand, more transistors are available in the same area thus increasing the computational power and also the speed; this gives a huge boost to performance. On the other hand, a higher transistor density increases complexity in terms of design, physical implementation, and verification. Moreover, it presents drawbacks such as higher power consumption due to the shrinking of the transistor's size and higher heat dissipation due to the higher transistor density. To limit such problems, a re-direction of design purpose is needed.

General-purpose microprocessors present high flexibility but also high area occupation and power consumption. Moreover, in most of the applications, they are not even completely exploited thus resulting in a waste of power and resources. On the other hand, special-purpose microprocessors present high performance for specific tasks along with a smaller overload on power consumption and area occupation. Additionally, they lack flexibility and programmability forcing engineers to re-design them for different applications. A link between general-purpose microprocessors and special-purpose microprocessors is required to mitigate their drawbacks and enhance their advantages. Such a bridge is named Application Specific Instruction-set Processor (ASIP).

This chapter will set down an exploratory study of an ASIP by first describing its architecture definitions, followed by the customization of two microprocessors with relevant issues and solutions.

3.1 Application Specific Instruction-set Processor

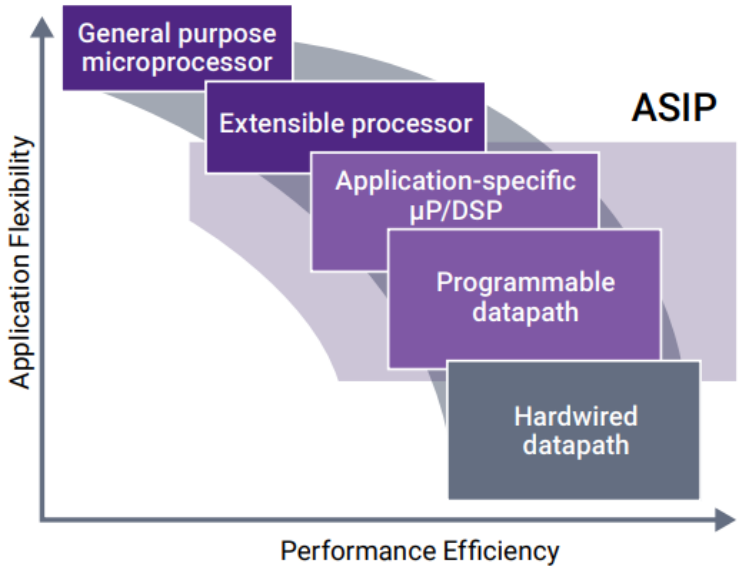

Figure 3.1: Flexibility and efficiency diagram for several types of data processing solution.

[36]

Figure 3.1 shows graphically the application flexibility versus power and performance efficiency for several kinds of data processing solutions. On the left, there are the general-purpose microprocessors that are very flexible in terms of application but with the drawback of being very less efficient for what concerns power and performance. On the other hand, there are hardwired datapaths on the right side that are very efficient for performance and power but for a single application. In between the general-purpose microprocessor and the hardwired datapath, there is a wide range for the Application Specific Instruction-set Processor (ASIP) which reasonably connects the two extremities.

ASIP provides both benefits, a hardware efficiency like a customized datapath combined with software programmability. ASIPs are useful, for instance, in hardware accelerators since usually, their datapath consists of handwritten Finite State Machines (FSMs) that are not easily reusable, and this makes them less flexible. Moreover, it also helps to improve other processors like DSPs and to create ad hoc processors focused on specific tasks. ASIP optimizes the performance of the

processor by implementing a customized architecture with ad hoc instructions or operations or by introducing several types of parallelism to enhance the application's execution.

Thanks to the aforementioned specializations, the processor benefits in terms of power consumption, and ASIP can introduce low power optimizations by adding clock and power gating. Finally, ASIP provides the designer with good programmability to avoid the ASIC to go through several respins. As a drawback, ASIP design requires a specific knowledge to describe a customized processor with a high-level language.

This thesis aims to develop such a knowledge and to demonstrate the flow to develop an ad-hoc processor for preliminary programmable data processing on pixel imagers. To create ASIPs from a high-level language, there are various commercial solutions available like Transport triggered architecture-based Co-design Environment (TCE) that is open source, Studio by Codasip, and ASIP Designer by Synopsys [37]. The latter one is the tool used in this thesis.

3.1.1 ASIP optimization space

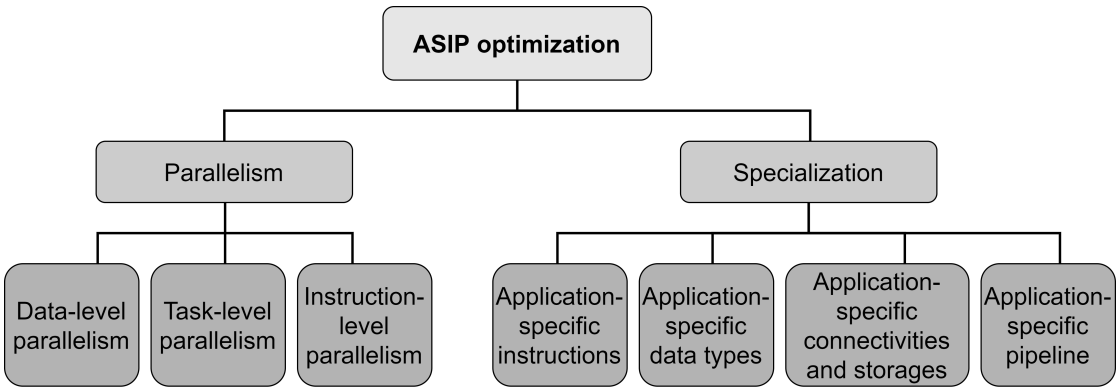

Figure 3.2: Tree diagram of the optimization space provided by ASIP Designer.

The possibilities in terms of processor optimization are quite wide as it is shown in figure 3.2. They are mainly separated into parallelism and specialization.

Parallelism can be performed on different levels such as:

- Data-level parallelism: to perform the same operations on different data samples executed by multiple processing elements to increase the throughput.

Usually, this is achieved by Single Instruction Multiple Data (SIMD) mode [38].

- Task-level parallelism: to perform multiple tasks in parallel by exploiting multi-core and multi-threading. This is in contrast to data-level parallelism.
- Instruction-level parallelism: to have multiple instructions executed in parallel from a single instruction. This applies to such cases where some instructions, belonging to the same thread, do not depend on each other and can be performed in parallel. This is different with respect to concurrency where multiple threads are executed in alternation on a single core, giving the impression that they are executed in parallel because of the high switching frequency [39]. An example is the Very Long Instruction Word (VLIW) as an instruction set architecture where instructions are executed in parallel.

On the other side, specialization of the processor can be achieved by:

- Application-specific instructions: introduces additional instructions for memory addressing, data processing, or control processing to perform specific instructions thus optimizing the processor. For instance, it might be useful to have an application-specific operator that performs an operation on a user-defined number of cycles.
- Application-specific data types: introduces additional data types which were not present such as fractional numbers, and floating-point in order to perform specific operations on these data types that usually are not included in classic processors.
- Application-specific connectivities and storages: a specification is required in terms of register files and their allocations, if central or distributed and how to connect them to the datapath. Similar decisions must be taken for memory elements such as their number, their position, and how to connect them to functional units.
- Application-specific pipeline: the pipeline depth might change depending on the required clock frequency from specifications. Moreover, solutions for introduced hazards are necessary to be defined.

3.2 Workflow

In this thesis, the workflow to generate a novel customized processor model is based on ASIP Designer by Synopsys. Thanks to the retargetable environment,

ASIP Designer can change to fit the provided processor. Thus, it needs a special-purpose high-level language to describe the Instruction Set Architecture (ISA). This language is called nML, a domain-specific but also a Register Transfer Level (RTL) language able to describe the instructions of the processor that trigger register transfers of its datapath [40].

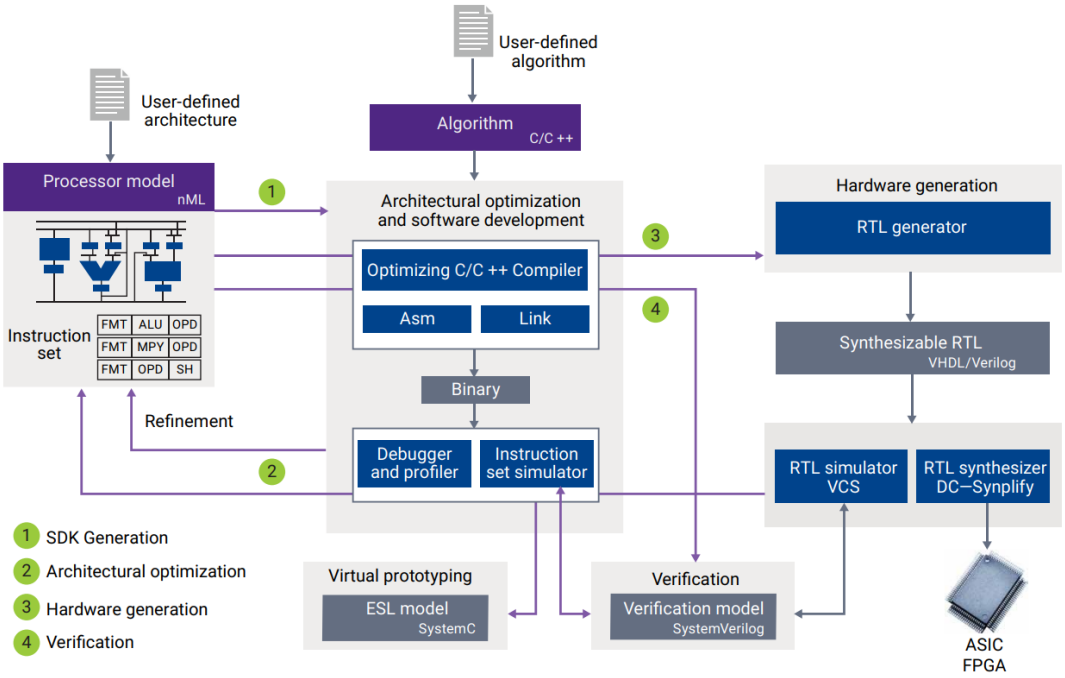

Figure 3.3: Flow of ASIP Designer by Synopsys. [36]

As shown in the figure 3.3, the processor model, written in nML language by the designer, is an input of the actual tool. Such a processor model is designed with a specific user-defined architecture. ASIP Designer is provided with a Software Development Kit (SDK) that contains multiple features such as an optimizing C and C++ compiler, a linker, and an assembler.

From a user-defined application-specific test code written in C or C++, which is another input of the tool, it is possible to simulate the processor model behavior. This might be done either in an instruction-accurate or cycle-accurate way thanks to the Instruction Set Simulator (ISS) already present in this SDK.
Furthermore, from the user-defined algorithm applied to the processor model, it is possible to exploit the debugger and profiler, already present in the SDK, to solve bugs and optimize the processor architecture. Profiling is obtained by running

the algorithm in the ISS and it gives information about instructions, functional units, primitive operations, nML coverage, hazards, and storage access to get a report about what and what not the processor is using. This is the starting point for iterative optimizations of the processor. For instance, data types might be added or removed from the architecture, the same might be applied to specific instructions from the instruction set. It is also possible to add or remove some level of parallelism. Operations might be removed if unused, or created if necessary. The same applies to specific functional units or pipeline stages to be well suited for the application-specific test code.

ASIP Designer tool is also provided with an RTL Generator which, starting from the processor model written in nML language, automatically generates the RTL code either in VHDL or Verilog. The generated code has a specific structure starting from the top level with different modules to their behavioral description. Several configuration options might be enabled to generate an optimized RTL code by introducing clock gating or reducing critical paths.

To provide a complete analysis of the ASIP performance, the generated RTL can be implemented from RTL to Place and Route (PnR) with a specific technology. From the figures of merit extracted from this implementation, such as achievable clock frequency, area occupation, and power consumption, it is possible to further optimize the processor model to meet specifications.

The verification part consists of running the user-defined algorithm written in C/C++ on three different levels. The first one consists of compiling and executing the C code on the host compiler, and it is used as a reference. The second one simulates the same C code on the ISS of the processor model while the third one performs the simulation on the RTL Simulator, already present in ASIP Designer. Then, these three levels are compared to check for possible differences and mismatches.

3.3 nML: a structural processor language

A processor architecture, in ASIP Designer, is defined through several important characteristics and attributes. This is done by modeling the processor at a high level written in nML language. This language describes both the instruction-set architecture and hardware microarchitecture of the target processor. The following aspects are defined in nML:

- Primitive data types: can be either classic C language data types or user-defined ones such as floating-point types or complex types. The primitive

data types are declared as C++ language classes where the size and type of data, for instance, signed or unsigned, are described through class properties. Once defined, data types can be used in the processor model, and the compiler will use C built-in types to map these data types. This mapping is explicitly declared in the processor model. Moreover, also conversions between data types can be performed and these are declared as C++ conversion constructors.

- Primitive operations: can be either classic C language operations or user-defined ones. Primitive operations are executed on primitive data types. Primitive operations are modeled as C functions with inputs and outputs. Similarly to primitive data types, C built-in operators are translated into these primitive operations. This mapping is explicitly declared in the processor model.

- Functional units: are groups of primitive operations executed on the same hardware unit. They are not mandatory but convenient to perform specific operations together.

- Memories: are static storage elements with well-defined load and store operations where access timing and addressing mode are both specified. Both data type of stored values and address type must be coherent with previously declared primitive data types. In nML, memories are defined as a model and they are far from actual external memory implementations.

- Registers: are static storage elements that do not require load and store operations for access. Direct addressing is usually applied. Registers can be either individuals or piled up to create a register file with a defined size. For individual registers, only the data type of stored values is necessary while in a register file also address type is required. Depending on the application, register files can be either central or distributed. Interconnections of registers with functional units or primitive operations must also be specified.

- Pipeline registers: are defined in terms of the number of pipeline stages. Moreover, introduced hazards are also resolved by specifying hardware stalls, software stalls, bypasses, etc.

- Instruction-set: is defined through several rules and particular grammar and attributes to uniquely specify instructions. This is done by describing their actions, assembly language, and instruction encoding. These nML actions are described as a register-transfer model based on primitive functions. These actions can be customized by declaring and describing them also in different pipeline stages of the same instruction-set description. Moreover, also control instructions, used to deal with the flow of the program, such as subroutines,

interrupts, zero overhead loops, etc are specifically defined and modeled with primitive operations.

- Properties: are used to uniquely determine the purpose of special storage elements such as Program Memory, Program Counter, Data Memory, etc.
- Hardwired constants: are not required but useful if instantiated multiple times in the processor model.

In addition to defining primitive data types and operations, nML builds the structural skeleton of the processor model, as described before. Nevertheless, the behavior of primitive functions is captured separately in PDG language which stands for Primitives Definition and Generation [41].
PDG language is a language developed by ASIP Designer to describe these primitives once. It is a mix between the C language and Hardware Description Language (HDL) as Verilog; and it is used to define purely functional primitives, controller primitives and other intricate primitives. It can use primitive data types declared by the designer in the nML language as well as C language operators with additional features.
Without this PDG language, primitives defined in a high-level language as nML need to be implemented both into C++ language to be run on the ISS as well as implemented into VHDL or Verilog to be run on the HDL simulator. These two implementations might rise inconsistencies between them. Primitives defined in PDG description avoid these issues and also the double effort. PDG tool is able to easily generate C++, VHDL and Verilog implementations without inconsistencies. Moreover, other components necessary in a processor are described in PDG such as the Processor Control Unit (PCU) and additional IO interfaces as transition layers between the abstract view of memories in nML and the external physical memories.

3.4 Processor architecture

ASIP Designer is equipped with numerous working example processors with different specifications and modeling features. They go from processors meant for educational purposes to domain-specific processors such as specialized accelerator processors for filtering, motion estimation, images, etc. There is also an example of a digital signal processor and several examples of microcontrollers implementing different instruction set architectures. For each of them, ASIP Designer has dispensed a complete project file with all the necessary libraries and processor model files. All the configuration projects needed to generate the C++ code to be run on ISS and RTL code to be run on the RTL simulator are also present with a regression setup

for the verification flow. Despite the regression verification, all these processors are not formally verified by Synopsys as IP processors. Moreover, many of these processors have manuals to have in-depth knowledge and description of their working principles.

3.4.1 Processors' description

After a general overview of these examples, two microcontrollers have been selected as starting points: Trv32p3 and Tmicro. The scope is to reach the minimum set of instructions necessary to perform on-chip data processing in HEP. This is achieved by a fair comparison of optimization between two microprocessors, which are implementing different instruction-set architectures and microarchitectures. Trv32p3 is a RISC-V-based microprocessor while Tmicro implements a customized ISA. A complete description of both microprocessors is given as follows.

Trv32p3 presents the following characteristics and features [42]:

- A 32-bit RISC-V microcontroller supporting the RV32IM [43] as ISA
- An ALU with integer arithmetic, bitwise logical, shift and compare instructions
- An AGU to generate addresses to access memories for load and store operations
- A three-stage protected pipeline for fetching, decoding, and executing stages with register bypasses and both hardware and software stalls
- A 32-bit hardware multiplier that gives 64-bit results
- A 32-bit iterative multi-cycle divider
- A central general-purpose register file with 32 registers
- No specific move instructions and the ALU is used to perform such instructions
- Control flow instructions for conditional branches, direct jumps, indirect jumps and links

Trv32p3 belongs to the TRV family of processors provided by ASIP Designer in which other microcontrollers are present with different features but all of them are RISC-V based. There are 32-bit or 64-bit processors with three or five pipeline stages. Extensions of RISC-V are present for single-precision floating-point number instructions or compressed instructions. Trv32p3 has been chosen because it presents the simplest RISC-V architecture among all these processors.

Tmicro presents the following characteristics and features [44]:

- A 16-bit microcontroller with a customized ISA
- An ALU-Shift with integer arithmetic, bitwise logical, compare and shift instructions
- A MAC unit for multiply-accumulate operations
- AGU to generate addresses to access memories for load and store operations where the addressing mode is indirect with optional post-modification.
- A three-stage protected pipeline for fetching, decoding, and executing stages with software stalls that complement user-defined hardware stalls and bypass rules
- A 16-bit multiplier that gives 32-bit results
- A 16-bit iterative multi-cycle divider
- A central general-purpose register file with 8 registers
- Move instructions
- Control flow instructions for jumps (conditional or unconditional and relative or absolute), call to subroutines (direct or indirect), return from subroutines, loops...
- Vectored interrupts
- Zero overhead loops

Tmicro has been chosen because it is one of the smallest non-RISC-V-based processors with a complete basic instruction set architecture provided by ASIP Designer. These are all the instructions and features included in these two examples. From these microprocessor examples as a starting point, it is possible to reach an application-specific processor that fulfills our specifications. This is achieved by adding specific instructions or functional units to perform some operations efficiently. Another optimization comes from removing instructions and functional units that are not used at all to lighten processor complexity and resource usage.

3.4.2 Memory interface

IO interfaces, in general, are useful to connect the processor core to its memories by representing the latter ones in an abstract view. This is done by describing the data and address types, the timing, and the ports of the memory to give an

abstract presentation in nML. Real memories have more properties that are not described in this module. IO interfaces take nML memories as input with additional inports and outports, describe the memory behavior for load and store operations employing registers and transitories; and finally connect them to external interfaces by declaring local memories [41]. In these examples, microprocessors are already provided with a Data Memory (DM) that acts as a default memory for program data storage, and a Program Memory (PM) for program code storage.
In the Trv32p3 microprocessor, DM size is 2^{32} and stores 32-bit signed data with addressing of 32-bit unsigned. Additionally, also PM size is 2^{32} but it uses 32-bit unsigned data type both for stored data and address.
On the other hand, in the Tmicro microprocessor, DM size is 2^{16} and stores 16-bit signed data with addressing of 16-bit unsigned. Additionally, also PM size is 2^{16} but it uses 16-bit unsigned data type both for stored data and address.
In figure 3.4, DM is represented on the left side of the processor and PM on the bottom-right side of it. The interface of DM has been kept intact while an extra stage has been added to the PM which will be well described later on.

3.5 Processor customization

Starting from these two examples, an additional interface is needed for each microprocessor toward the external world. This allows a future integration of microprocessors in a System-on-a-Chip (SoC) environment and possible further extensions of these processor models involving multiple memories or processors. This interface is used to output the processed data. AMBA APB system bus [45] has been chosen as a protocol for this interface. The choice of such a bus system is due to consistency since it is already integrated into other SoC modules and IP blocks developed by the Micro-Electronics (ME) group at CERN.

3.5.1 APB interface

The AMBA APB system bus is realized through serially connected IO interface modules written in the processor model in PDG language.
Moreover, a custom Interface Memory has been declared, defined, and included in the processor model as a novel memory in nML language. This Interface Memory has been used as a testing feature for this APB system bus.

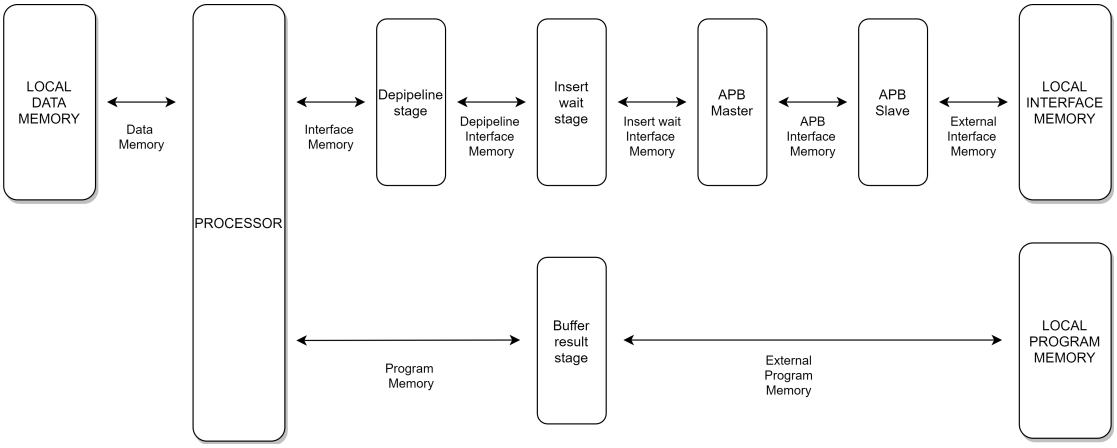

Figure 3.4: Integration of customized interface from the processor towards an Interface Memory.

These IO interfaces are shown in figure 3.4 as stages represented as rounded boxes and the memory interfaces are the connections between these boxes. In addition to Data Memory and Program Memory previously described, the custom Interface Memory has been connected to the APB system bus of both microprocessors and will be used thereafter in the application-specific test code. The IO interface stages introduced for this APB interface are outlined as follows:

System bus

The interface between the local Interface Memory and the processor core exploits the AMBA APB system bus as previously described. The acknowledge response from the APB testbench slave, represented by the Interface Memory, is emulated since there is no physical presence of such an element. This signal tells if the memory element has accepted or not any request of load or store operation from the processor through the master.

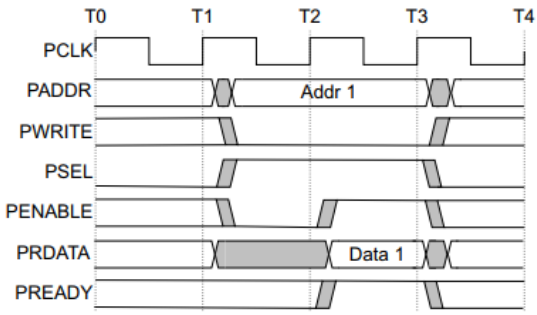

Figure 3.5: Read transfer signals in AMBA APB system bus protocol. [45]

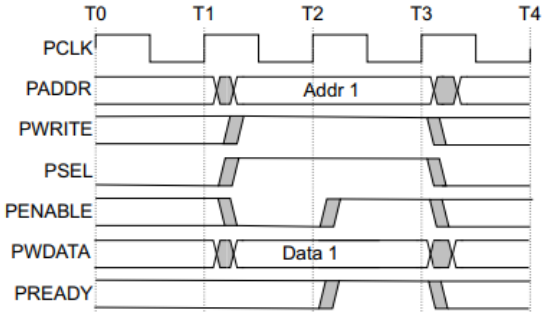

Figure 3.6: Write transfer signals in AMBA APB system bus protocol. [45]

Figures 3.5 and 3.6 respectively show read and write transfer signals for the AMBA APB system bus [45]. PCLK is the clock signal and all the APB signals are sensitive to its rising edge. PADDR is the APB address bus. PWRITE is the direction signal that tells if the transfer is a read (low) or a write (high). PSEL is the selection signal generated by the master and there is one for each slave to tell which one is selected. PENABLE is the enable signal referring to the second cycle (or more) of the transfer. PRDATA carries the read data from the slave toward the master when PWRITE is low. Similarly, PWDATA carries the write data from the master toward the slave when PWRITE is high. Finally, PREADY is the ready signal of the transfer.
In both figures 3.5 and 3.6, at T1 there is the setup stage for read and write transfer by triggering the PSEL used as a select signal. At this point, PADDR, PWDATA (for write transfer) or PRDATA (for read transfer), and PWRITE must be stable. Moreover, PENABLE goes low. The next stage is the access stage at T2. At this moment, PENABLE goes high as the PREADY signal. The latter signal indicates that at T3, the write data will be acknowledged (for a write transfer) or the read data will be presented (for a read transfer) prior to the read transfer's conclusion. During this transfer, all the control signals must be valid and stable. Finally, at T3, if there is no following transfer on the same slave, PSEL goes low.

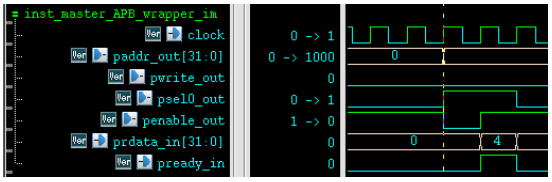

Figure 3.7: Read transfer signals toward Interface Memory in customized Trv32p3 microprocessor using AMBA APB system bus.

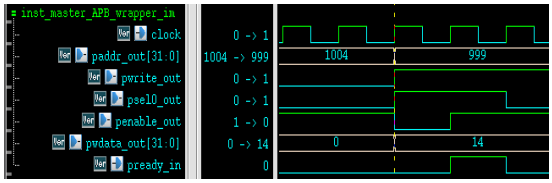

Figure 3.8: Write transfer signals toward Interface Memory in customized Trv32p3 microprocessor using AMBA APB system bus.

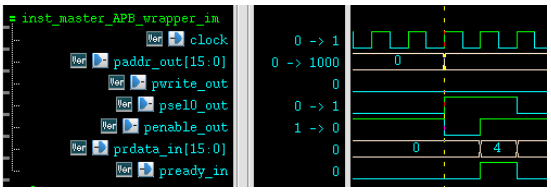

Figure 3.9: Read transfer signals toward Interface Memory in customized Tmicro microprocessor using AMBA APB system bus.

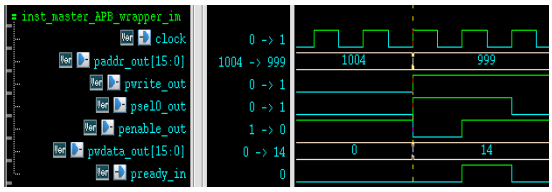

Figure 3.10: Write transfer signals toward Interface Memory in customized Tmicro microprocessor using AMBA APB system bus.

A similar behavior for read and write transfers has been applied to the customized Trv32p3 and Tmicro microprocessors. Figures 3.7 and 3.8 respectively show the read and write transfer signals for customized Trv32p3 microprocessor, in which PADDR, PRDATA and PWDATA are 32-bit signals. Similarly, in figures 3.9 and 3.10 there are the same read and write transfer signals for customized Tmicro microprocessor, in which PADDR, PRDATA, and PWDATA are 16-bit signals.

In both microprocessors, the selection signal PSEL has been named PSEL0 for possible further integration of more slaves, and there is no priority algorithm due to the presence of a single slave.

In figures 3.7 and 3.9, reading operation of value 4 from address 1000 of the Interface Memory works correctly for both microprocessors. The selection signal is asserted for two consecutive clock cycles and the enable signal is de-asserted during the first one and asserted during the second one. Moreover, PRDATA is presented during the second cycle as well as PREADY signal.

Analogously, in figures 3.8 and 3.10, writing operation of value 14 at address 999 of the Interface Memory runs accurately on both microprocessors. Also here, the selection signal is asserted for two consecutive clock cycles and the enable signal is de-asserted during the first one and asserted during the second one. Furthermore, PWDATA is valid and stable since the triggering of write transfer while PREADY signal is asserted during the second cycle.

These correct behaviors have been achieved after several attempts and modifications of the master and slave stages of the interface. These stages are better described later on. The refinements have been done by changing the interface description in PDG language in terms of signal timings.
The APB transfer behavior has been precisely checked thanks to the waveforms generated by Verdi, the automated debug system provided by Synopsys, and present in ASIP Designer. The inspection has been performed in the early stage of the interface creation because it is easier to do corrections at this level rather than later on when the processor model gets too complicated. These waveforms have been examined several times during the exploration and integration of the customized processor model.

Depipeline stage

The timing of load and store operations, described in the processor model, must be consistent with the specifications of the protocol chosen. For a load operation, request of the operation and generation of the address happen in the decode stage, while loading of the data occurs in the execute stage. On the other hand, for a store operation, request of the operation, generation of the address, and storage of the data happen all in the execute stage. An equivalent memory access timing can be declared in the processor model along with the memory declaration. The problem with such timing is the following: if a load operation follows a store operation, a structural hazard occurs because of resource conflict.

In processors such as the Trv32p3, both load and store operations are described in a pipeline fashion. The request and the generation of the address happen in the decode stage while the corresponding operation on the data occurs in the execute stage. This helps to avoid structural hazards.

To be completely consistent for load and store operation timing between peripheral memory and the one declared in the processor model, a de-pipeline stage is necessary for store operation. This stage becomes pointless when both timings of the processor and peripheral memory are consistent with each other.

Insert wait state stage

The next IO interface module is the insert wait stage. This stage acts when the processor sends a request, but the memory is busy and cannot acknowledge it. It also helps in the opposite case when the memory provides a result, but the processor cannot receive it because it is in a wait state due to other modules.
In the first case, this stage buffers the request raised by the processor to extend it and make it available in the next cycles until the memory acknowledges it. During this time, wait states are introduced in the processor as no-operations also known as no-op or nop.
In the second case, this stage behaves similarly by buffering the read results coming from the memory when the processor is in a wait state, and forwarding it when the processor becomes available.

Further modifications have been introduced to the processor model in order to correctly act during a wait state:

- A no-operation wait state declaration has been introduced in the PCU
- A no-operation wait state declaration has been introduced in the behavioral model of the processors
- An extra option for no-operation has been added to stall the processor in the HDL generation configuration file
- An extra option has been added to the ISS project file to define the wait state configuration as a no-operation

Buffer result stage on Program Memory

In the beginning, only the novel interface had been added but this raised some problems due to the introduction of wait states. This can be seen clearly in listing 3.1 where a comparison of register change dumps between the ISS and the RTL is presented.

Listing 3.1: Comparison of register change dumps between ISS and RTL

```
1                ISS                          common                        RTL
2                ---                          ------                        ---
3     EDM_0[518] = 0                             !
4     EDM_1[518] = 0                             !
5     EDM_2[518] = 0                             !
6     EDM_3[518] = 0                             !
7                                        PC_ID = 00000048
8                                             22 (76)
9                                        PC_ID = 0000004C
```

```
10    X[3] = 2076                    !                    X[6] = 1341
11                                 23 (80)
12                                   !                  EDM_0[518] = 0
13                                   !                  EDM_1[518] = 0
14                                   !                  EDM_2[518] = 0
15                                   !                  EDM_3[518] = 0
16                               PC_ID = 00000050
17                                X[6] = 1341
```

A register change dump is a log file containing the cycle number, the value of the program counter, also written in parenthesis, and the assignment of a value to a register at a particular address. This is executed on both ISS and RTL simulations through the application-specific test code. A comparison between these two register change dump files shows possible errors, mismatches, or misalignment between these two simulations. This is done to verify the correct functioning of the generated RTL compared to the ISS, since the latter is already compared to the native simulation for verification, and thus, can be used as a reference. If the comparison presents some anomalies, the HDL needs to be debugged [46].
This comparison is used for running regression tests to verify the correct functionality of the processor model. This has been performed at each modification of the processor model to check for possible bugs in the HDL.

In listing 3.1, a clear misalignment is present between the ISS and RTL simulations. The assignment of value zero to the external DM at address 518 takes place at two different times. This phenomenon occurs for each register assignment along the whole simulation, which means it is consistent and related to the no-operations introduced by the insert wait stage of the custom interface.

Because of this mismatch between register change dumps, additionally to this novel interface, a specific buffering stage must be introduced to the PM, which is always reactive in these simple microprocessors. This stage is equivalent to the insert wait stage of the novel interface. When the processor is in a wait state (nop operation), this stage buffers the read results coming from the memory since they cannot be received by the processor.

3.5.2 Load and Store instructions of Interface Memory

Once the structure of the additional interface is well defined and verified, an actual link between this and the processor is necessary. This connection is achieved by describing the load-store instructions of Interface Memory in the nML model,

exploiting the central register file of the processor. The description of such instructions differs between Trv32p3 and Tmicro processors, in order to adapt to the already existing load-store instructions on DM in the hosting processor.
Inside such an nML model, a functional unit for address generation is generally used. The load-store instructions are defined as operations with specific actions, syntax, and image attributes. In the action attribute, a detailed behavior of the operation is described separated into pipeline stages.

Trv32p3 microprocessor

Listing 3.2: nML model of load and store instructions for Interface Memory in Trv32p3 processor

```
fu im_agu;
trn im_aguA <w32>;        // opA for IM
trn im_aguB <w32>;        // opB for IM
trn im_aguR <w32>;        // agu result for IM

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// ~~~ Top-Level Rule for this nML file

opn im_load_store_instrs (
    im_load_instr |
    im_store_instr
);

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// ~~~ X[] Load Operations

opn im_load_instr (rd: mX1w_EX, rs1: mX3r_ID, offs: c12s)
{
    action {
      stage ID:        im_aguA = rs1;
                       im_aguB = offs;

      stage ID:        im_aguR = add (im_aguA,im_aguB) @im_agu;
                       im_addr = im_aguR;

      stage ID..EX:
                      im_read`EX` = IM[im_addr`ID`]`EX`;
      stage EX:
                       rd = im_read;
    }
          syntax : "IM LD " PADMNM " " rd "," PADOP1 offs "(" rs1 ")
    ";
          image : offs :: rs1 :: "000" :: rd :: opc32.reserved3
                    class(im_load), class(im_load_store);
```

```
34
35 }
36
37 // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
38 // ~~~ X[] Store operations
39
40 opn im_store_instr (rs1: mX3r_ID, rs2: mX2r_EX, offs: c12s)
41 {
42     action {
43       stage ID:        im_aguA = rs1;
44                        im_aguB = offs;
45
46       stage ID:        im_aguR = add (im_aguA,im_aguB) @im_agu;
47                        im_addr = im_aguR;
48
49       stage ID..EX:
50                      IM[im_addr`ID`]`EX` = im_write`EX` = rs2;
51     }
52           syntax : "IM ST " PADMNM " " rs2 "," PADOP1 offs "(" rs1 ")
     ";
53           image : offs :: rs1 :: "111" :: rs2 :: opc32.reserved3
54                   class(im_store), class(im_load_store);
55
56 }
57
58 // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
59 // ~~~ Chess Views for AGU Operations
60 // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
61
62 // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
63 // ~~~ Indirect Addressing
64
65 chess_view () {
66   im_aguR = add (im_aguA, im_aguB = 0);
67 } -> {
68   im_aguR = im_aguA;
69 }
70
71 // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
72 // ~~~ Direct Addressing
73
74 chess_view () {
75   im_aguR = add (im_aguA = zero, im_aguB);
76 } -> {
77   im_aguR = im_aguB;
78 }
```

In listing 3.2, there is the novel nML model of load-store instructions for Interface

Memory in Trv32p3 microprocessor.

In the first part, the functional unit for the AGU is declared with its three transitories. Notice that the data type is $w32$ which is a 32-bit signed value previously declared as a primitive data type.
In the second part, there is a top-level rule in which a generic load-store instruction is subdivided respectively into load and store instruction.
The third part contains the load operation. It has three variables as inputs: rd, $rs1$, and $offs$.
The first two are referred to as different mode rules to access the central register file X of the processor: rd accesses this register in a write mode during the execute pipeline stage; while $rs1$ accesses this register in a read mode during the decode pipeline stage. Finally, $offs$ takes a constant 12-bit signed value. The action attribute of load operation describes its actual behavior. During the decode stage, the address is generated as an addition between $rs1$ and $offs$. This takes place in the AGU functional unit described at the beginning. Then, the address im_addr is sent to the memory during the decode stage while the access to it happens in the execute stage as well as the data available on the im_read bus.
Finally, during the execute stage, the loaded value from the Interface Memory is written in the central register file of Trv32p3 through rd. In line 31, the syntax attribute defines how this instruction will present itself in the generated microcode. It is a combination of text and input variables. Then, there is the image attribute which tells how this instruction will be encoded and saved in the program memory. Here, it is a concatenation of the three input variables and some dedicated major operation codes, such as $opc32.reserved3$, which is a declared constant to uniquely specify the instruction. The instruction can also belong to specific classes, such as im_store and im_load_store at line 54, to facilitate debugging and profiling.

Analogously, the fourth part describes the store operation to the Interface Memory. $rs1$ and $offs$ are still used as input variables and instead of rd, now there is $rs2$ which accesses the central register X in a read mode during the execute pipeline stage. The address generation is equivalent to the load operation one. Moreover, $rs2$ forwards the value present in the central register to the im_write bus during the execute pipeline stage. During the same stage, the memory core is accessed while the address was sent to the memory in the previous pipeline stage of decode. The images of both load and store operations, besides the usage of rd or $rs2$, differ by a specific 3-bit value which is "000" for load operation and "111" for store operation. If any couple of instructions have the same image, the tool will not build the processor model and will alert with errors.

Finally, at the end of listing 3.2, there is an optimization for the tool applied

to the AGU which states: if one of the addends used to generate the address is equal to zero, then the resulting address is equal to the other addend. This avoids the futile usage of adders.

Tmicro microprocessor

Listing 3.3: nML model of load and store instructions for Interface Memory in Tmicro processor

```
opn im_load_store_instr( im_load_store_wreg_indirect     )

{
    image : "1100"::"00"::im_load_store_wreg_indirect
    ;
}

fu im_ag1;
trn im_ag1p<word>;
trn im_ag1m<word>;
trn im_ag1q<word>;

enum im_load_store_op { ld, st };


// load/store instructions with linear address manipulations

enum im_ag1_op { indir "", incr "++", decr "--" };

opn im_ag1_opn(op: im_ag1_op, r: rrid)
{
    action {
    stage ID:
    switch (op) {
    case indir: im_ag1p = r;
    case incr:  r = im_ag1q = add(im_ag1p=r,im_ag1m= 1) @im_ag1;
    case decr:  r = im_ag1q = add(im_ag1p=r,im_ag1m=-1) @im_ag1;
    }
    }
    syntax : r op;
    image  : r::op;
}

opn im_load_store_wreg_indirect(op: im_load_store_op, rr: wreg, ag:
    im_ag1_opn)
{
    action {
    stage ID:
```

```
38      ag;
39      stage ID..E1:
40      switch (op) {
41      case ld:
42          Tmicro_im_addr‘ID‘ = im_ag1p‘ID‘;
43          rr‘E1‘ = Tmicro_im_read‘E1‘ = Tmicro_IM[Tmicro_im_addr‘ID‘]‘
        ID‘;
44      case st:
45          stage ID:
46          Tmicro_im_addr_pipe = im_ag1p;
47          stage E1:
48          Tmicro_im_addr = Tmicro_im_addr_pipe;
49              Tmicro_IM[Tmicro_im_addr] = Tmicro_im_write = rr;
50      }
51      }
52      syntax : op " " rr ",Tmicro_im(" ag ")";
53      image  : op::ag::rr;
54  }
```

Similarly, in listing 3.3, there is the novel nML model of load-store instructions for Interface Memory in the Tmicro microprocessor.

In the beginning, there is a top-level rule used to properly encode the actual load-store operation. The image is the concatenation of "1100", "00" and the image of *im_load_store_wreg_indirect* operation. The first two values are defined to avoid possible conflicts with images of other operations and thus to uniquely identify this instruction. Then, there is the declaration of the AGU functional unit used to generate the address with its three transitories. The data type is *word* which is a 16-bit signed value previously declared among the other primitive data types.

In Tmicro, load and store instructions are only defined with linear address manipulation. The address generation is described in a separate operation named *im_ag*1*_opn* at line 20. It takes the type of address operation such as indirect, increment, or decrement; and r the decode pipeline stage read port of central register R of the processor, as input variables. The action attribute, in line 22, describes the instruction behavior. During the decode pipeline stage, the address might be generated indirectly or; incremented or decremented using an adder present in the previously defined functional unit.

Differently from Trv32p3, in Tmicro microprocessor, store and load instructions are defined in a single operation called *im_load_store_wreg_indirect*. It takes the operation type, load or store, as an input variable. Moreover, also *rr* is an input

variable representing the memory access mode either in read or write during any pipeline stage. Finally, another input is *ag* which carries the address value generated by *im_ag1_opn* operation which is performed during the decode stage. At line 40, load and store operations are separately described thanks to a switch statement. For a load operation, the address is sent to memory during the decode stage as well as the access to it. Data becomes available on *micro_im_read* bus in execute stage as well as the writing of such value in the central register of Tmicro. On the other hand, in the store operation, an extra pipeline signal *Tmicro_im_addr_pipe* is present and it is used for On-Chip Debugging purposes as shown in listing 3.5. Then, sending the address to memory, accessing it, and availability of data on the bus, all happen during the execute stage. Notice that, the type of operation on Interface Memory will be the discriminator to uniquely determine the image.

Once the APB interface has been created and connected to the microprocessor through custom load and store instructions, On-Chip Debugging must be updated as well. Every time a new memory element is introduced, such as the Interface Memory in this case, it needs to be connected to the On-Chip Debugging module to maintain consistency and coherency of the processor model. This is better described in the next subsection.

3.5.3 On-Chip Debugging

Both microprocessors implement the On-Chip Debugging (OCD) feature to create a debug interface and debug controller modules useful for hardware debugging. The connection between the processor and the debug controller is done through specific registers for address, data, and instruction. These registers are used to access memories from the debug interface. The load and store operations must be specified in nML language for each memory element declared in the processor model [46].

The OCD specifications are already described for DM and PM memories. Novel OCD memory accesses have been declared in nML for the newly introduced Interface Memory as shown in listing 3.4 for Trv32p3 microprocessor and in listing 3.5 for Tmicro microprocessor. These nML models are defined in an *always*() nML rule so they occur every time. Load and store accesses to memories are guarded by transitories with specific names to create a bridge between the processor and debug controller.

For Trv32p3, the debug moves are described along the three pipeline stages. During the first stage, OCD guarding transitories are used to raise a load or store request. During the second stage, the OCD address is forwarded to the Interface Memory address bus and used to access the memory in the third stage. In a load

operation, this data becomes available on *im_read* bus used to forward it towards the *ocd_imd* register of the debug controller. Analogously, in a store operation, the data coming from the debug controller, present in *ocd_imd_r* register, is stored in the Interface Memory through the *im_write* bus. Notice that *ocd_imd_r* is just a buffer register of *ocd_imd*.

Listing 3.4: nML model for OCD memory accesses to Interface Memory in Trv32p3 processor

```
1     // IM debug moves
2     stage 1..3:
3       guard (ocd_ld_IM‘1‘){
4         stage 2:
5           ocd_addr = ocd_addr_w = incr1(ocd_addr_r) @ocd_addr_incr;
6         stage 2..3:
7           ocd_imd‘3‘ = ocd_imd_w = im_read = IM[im_addr‘2‘=ocd_addr_r
    ‘2‘]‘3‘;
8       }
9     stage 1..3:
10      guard (ocd_st_IM‘1‘){
11        stage 2:
12          ocd_addr = ocd_addr_w = incr1(ocd_addr_r) @ocd_addr_incr;
13        stage 2..3:
14          IM[im_addr‘2‘=ocd_addr_r‘2‘]‘3‘ = im_write‘3‘ = ocd_imd_r
    ‘3‘;
15      }
```

Similar behavior of OCD memory access is present in the Tmicro processor where *ocd_data* is used as the data register of debug controller. For the load operation, the OCD address is forwarded to the Interface Memory address bus which takes place in the decode stage as well as the access to the memory. In execute stage, data is available on the *Tmicro_im_read* bus and carried out to the *ocd_data* register. On the other hand, in store operation, *pm_addr_pipe* signal defined in the previous paragraph is used for an intermediate state; while the forwarding of data present in *ocd_data* register towards the Interface Memory occurs in execute stage.
Notice that in Trv32p3, the pipeline stages have been named 1, 2, and 3 to generalize the OCD. On the contrary, in Tmicro, pipeline stages in OCD have the same names as the ones defined in the processor model.

Listing 3.5: nML model for OCD memory accesses to Interface Memory in Tmicro processor

```
1     stage ID..E1:
2         guard(ocd_ld_Tmicro_IM){
3             ocd_data ‘E1‘ = ocd_data_w = Tmicro_im_read ‘E1‘ =
```

```
4                   Tmicro_IM[Tmicro_im_addr`ID` = ocd_addr_r = ocd_addr
    `ID`]`ID`;
5           }
6           guard(ocd_st_Tmicro_IM){
7           stage ID:
8               Tmicro_im_addr_pipe = ocd_addr_r = ocd_addr;
9           stage E1:
10              Tmicro_IM[Tmicro_im_addr = Tmicro_im_addr_pipe] =
    Tmicro_im_write = ocd_data_r = ocd_data;
11
12          }
```

OCD is an extremely useful feature for hardware debugging. Nevertheless, it is not mandatory and it can be easily disabled when unnecessary.

3.6 Final considerations

General-purpose processors and hardwired datapaths present complementary advantages and disadvantages. An Application Specific Instruction-set Processor permits the creation of a bridge between them and enhances their benefits. This chapter explored the possibility to use an ASIP for on-chip data processing in future pixel detectors. ASIP design is a powerful tool that allows the designer to tailor the processor architecture to comply with an application. Nevertheless, the design of an ASIP requires a high-level architecture description language which is complex. Several specific aspects are needed to fully characterize an ASIP such as data types, operations, instruction-set, PCU, etc.
In this work frame, two microprocessors have been chosen as candidates to perform an exploratory study: Trv32p3 and Tmicro. The two microprocessors have been customized by integrating an interface towards the external world by following the APB system bus. Specific load and store instructions have been introduced in their processor models to fully connect the interface to the processor core.
The newly introduced APB interface is going to be exploited to test the functionality of the processor model and extract important results. Chapter 4 presents a specific-application test code with relevant profiling results to better understand the potentialities of ASIPs.

Chapter 4

Algorithm and Profiling

Once the processor model of a microprocessor is ready, the next step consists of testing it. This chapter is divided into two parts: algorithm and profiling. The first part shows the testing of the processor model through an application-specific test code written in C++ and ran on the processor model. This chapter shows a specific data processing algorithm developed for High Energy Physics applications. Input data of this algorithm are real physics data from old particle detectors. The second part of this chapter presents the profiling results of the processor model using the algorithm presented. This helps to better understand the utilization of processor model features.

4.1 Data processing Algorithm

Data processing of information coming from a detector is currently performed off-chip through FPGA or DSP or processors. A chart showing a generic data processing flow is provided in figure 4.1.

Figure 4.1: Flow chart of current data analysis.

After digitizing signals through the readout ASIC present on the chip, raw data from the pixel detector is forwarded off-chip to the data acquisition system. Raw data is a partly chronologically sorted stream of pixel hits or frames with information about position and intensity.

These pixel hits go through a clustering function that uses spatial and temporal coincidence to assign pixels to clusters. Several methodologies are applied for clustering such as QuadTree [47] able to process approximately 1 Mhits/s and Clustering from Petr Mánek [48] able to process approximately 3 MHits/s. Cluster events, outputted by the clustering function, go through classification and even filtering which depend on the application and provide data compression.
This section proposes an algorithm to perform part of the off-chip data processing previously described directly on the chip. The objective is not to obtain a software-optimized algorithm but to develop a test case to evaluate the processors described in section 3.4 and perform further optimizations.

4.1.1 Input data

Figure 4.2 shows a typical event to be processed: the first measurement results of Timepix3 chip [31] for radon decay products at the Environmental Research Station Schneefernerhaus. This image is the result of an integration time of 60 minutes. It shows a real-time measurement of several particles with their trajectories. These particles show various shapes of trajectories. α-particles usually present a rounded shape while muons present a straight trajectory. Data from all pixels

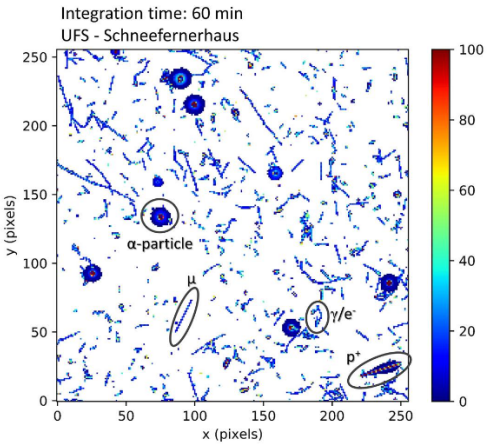

Figure 4.2: First measurement results of Timepix3 at the Environmental Research Station Schneefernerhaus on Zugspitze.
[49]

might be redundant and unnecessary. For instance, α-particles present raw data of approximately 100 - 200 pixels but the only necessary parameters are the mean position of its center, radius, and total energy. Similarly, particles with rectilinear trajectories, present raw data of approximately 10 - 200 pixels, and the necessary parameters are the coordinate of its most left pixel, the coordinate of its most right pixel, and its total energy.

Real raw data from Timepix3 has been used as input for such data processing. This set of data consists of information about pion particles, also known as pi mesons, denoted as π. A pion is a sub-atomic particle made of a quark and an anti-quark and represents the lightest meson and hadron. They are unstable particles and so they decay into muons as soon as possible. Through the process of photo-production, pion beams are created when multi-TeV proton or deuteron beams traveling through the LHC collide with photons from an X-ray Free Electron Laser [50].
Several files of raw data are present for different incident angles of pions of 0, 30, 50, 70, and 90 degrees on the detector. The clustering time window within a single track is 200 ns and, since the clock frequency is equal to 40 MHz, it is equivalent to 8 cycles. The wait time between the first pixel in the cluster and the finish flag is 500 µs.
Clustering event outputs are also provided, exploiting the QuadTree clustering method, as a reference to compare results with outputs coming from the data processing algorithm that will be described later on. These clustering events are computed off-chip and, in these reference output files, each line contains the clusters' start time and clusters' size.

Raw data format comes as follows in listing 4.1:

Listing 4.1: Pions raw data of Timepix3 at 120 GeV and incidence angle of 0 degree and integration time of 300 s

```
# Ikrum:              15   (1.087V)
# Vfbk:               164  (0.793V)
# Vthreshold_fine:    390  (0.778V)
# Vthreshold_coarse:      7    (0.778V)
# Ibias_DiscS1_ON:    100  (1.070V)
# Ibias_DiscS1_OFF:   8    (1.278V)
# Ibias_DiscS2_ON:    128  (0.341V)
# Ibias_DiscS2_OFF:   8    (0.196V)
# Ibias_PixelDAC:     128  (0.938V)
# Ibias_TPbufferIn:   128  (1.132V)
# Ibias_TPbufferOut:      128  (1.034V)
# VTP_coarse:         128  (0.642V)
# VTP_fine:       256  (0.635V)
# Ibias_CP_PLL:       128  (0.493V)
```

```
15 # PLL_Vcntrl:          128 (0.813V)
16 # BandGap output:     ----      (0.634V)
17 # BandGap_Temp:       ----      (0.683V)
18 # Ibias_dac:          ----      (1.198V)
19 # Ibias_dac_cas:      ----      (0.968V)
20 # DACs:      128 8    128 15   164 390 7    100 8    128 8    128 128 128
       128 256 128 128
21 # DACs Scans:    1.164V  1.306V  0.635V  1.087V  0.793V  0.778V  0.778
       V  1.070V  1.278V  0.341V  0.196V  0.938V  1.132V  1.034V  0.642V
         0.635V  0.493V  0.813V  0.634V  0.683V  1.198V  0.968V
22 # ------------------------------------------------------------------------
23 21248    3550080 2    16
24 41070    6762306 3    30
25 41065    6762306 6    12
26 41066    6762306 9    19
27 41068    6762306 7    24
28 40814    6762307 10   7
29 41067    6762306 9    23
30 41069    6762306 7    33
31 11894    19404263     11   9
32 11895    19404263     10   14
33 11638    19404263     12   16
34              .
35              .
36              .
37 # Frame:            1
38 #Start time:              5E-08
39 #End time:     300.00000005
40 #Hits: 17,814,239
41 #Lost Hits: 0
42 #UDP Tranfer: 99.7%
```

After a first listing of setting parameters such as threshold voltages and bias currents, data lines start at line 23. Each data line is composed of four parameters:

- The first column represents the pixel coordinate for a 256×256 pixel matrix. The division of such pixel coordinate by 256 gives a quotient, equivalent to y coordinate, and a remainder, equivalent to x coordinate.
- The second column represents a coefficient of Time-of-Arrival (ToA) of pion particle in cycles. Since clock frequency is 40 MHz, such a coefficient needs to be multiplied by 25 ns to obtain ToA in seconds.
- The third column represents a correction factor for ToA. For simplification, such a factor has not been taken into account.
- The fourth column represents the value of Time-over-Threshold (ToT) as the amplitude of the signal equivalent to the energy loss of the particle.

These raw data have been converted into C files thanks to a Python script. These C files are used as input data for our customized data processing algorithm. Input files for Trv32p3 and Tmicro microprocessors are not the same due to their different architectures. Such input data have been pre-stored in the DM memory of each processor. This has been done by declaring them as unsigned values and assigning them to specific DM memory locations through the "chess_storage" built-in function. The allocation addresses have been casually chosen to not conflict with some reserved DM locations. This is because for the sake of developing a test case, where and how the input data is stored does not matter. Front-end and readout ASIC are responsible for storing data in a specific allocation.

Listing 4.2: Input data file of Trv32p3 microprocessor converted from pions raw data of Timepix3 at 120 GeV and incidence angle of 0 degree and integration time of 300 s

```
#ifndef __DATA__H
#define __DATA__H
unsigned chess_storage(DMb:0x1000000)
in[23] = {22,       // Coord     ToA-#cycle    ToT

        3550080,    //           3550080
        0x53000010,// 21248                    16

        6762306,    //           6762306
        0xA06E001E,// 41070                    30

        6762306,    //           6762306
        0xA069000C,// 41065                    12

        6762306,    //           6762306
        0xA06A0013,// 41066                    19

        6762306,    //           6762306
        0xA06C0018,// 41068                    24

        6762307,    //           6762307
        0x9F6E0007,// 40814                     7

        6762306,    //           6762306
        0xA06B0017,// 41067                    23

        6762306,    //           6762306
        0xA06D0021,// 41069                    33

        19404263,   //          19404263
        0x2E760009,// 11894                     9

```

```
33          19404263,  //          19404263
34          0x2E77000E,// 11895                  14
35
36          19404263,  //          19404263
37          0x2D760010 // 11638                  16
38          };
39  #endif
```

Listing 4.2 shows input data file for Trv32p3 microprocessor. Information for each pixel occupies two DM memory locations. The first location is occupied by the ToA value in cycles while the second location is a concatenation between pixel coordinate and ToT value. This is because Trv32p3 is a 32-bit microprocessor and, pixel coordinate takes a maximum of 16 bits. After some time, the ToA value exceeds 32 bits and thus a reset on ToA occurs.

Listing 4.3: Input data file of Tmicro microprocessor converted from pions raw data of Timepix3 at 120 GeV and incidence angle of 0 degrees and integration time of 300 s

```
1  #ifndef __DATA__H
2  #define __DATA__H
3  unsigned chess_storage(DM:0x0fff)
4  in[34] = {33,        //      Coord          ToA-#cycle          ToT
5
6              1,       //                          1
7              0x5300,  // 21248 (0, 83)
8              0x0010,  //                                          16
9
10             2,       //                          2
11             0xA06E,  // 41070 (110, 160)
12             0x001E,  //                                          30
13
14             2,       //                          2
15             0xA069,  // 41065 (105, 160)
16             0x000C,  //                                          12
17
18             2,       //                          2
19             0xA06A,  // 41066 (106, 160)
20             0x0013,  //                                          19
21
22             2,       //                          2
23             0xA06C,  // 41068 (108, 160)
24             0x0018,  //                                          24
25
26             2,       //                          2
27             0x9F6E,  // 40814 (110, 159)
28             0x0007,  //                                           7
29
30             2,       //                          2
```

```
31              0xA06B, // 41067 (107, 160)
32              0x0017, //                                   23
33
34              2,      //                          2
35              0xA06D, // 41069 (109, 160)
36              0x0021, //                                   33
37
38              5,      //                          5
39              0x2E76, // 11894 (118, 46)
40              0x0009, //                                    9
41
42              5,      //                          5
43              0x2E77, // 11895 (119, 46)
44              0x000E, //                                   14
45
46              5,      //                          5
47              0x2D76, // 11638 (118, 45)
48              0x0010  //                                   16
49
50          };
51  #endif
```

Similarly, listing 4.3 shows the input data file for the Tmicro microprocessor. Information for each pixel occupies three DM memory locations. The first location is occupied by the ToA value in cycles, the second one is occupied by pixel coordinates, and finally, the third one by the ToT value. This is because Tmicro is a 16-bit microprocessor and, pixel coordinate takes a maximum of 16 bits. Since ToA values exceed 16 bits from the beginning, a reset and re-computation have been performed on ToA to fit in the 16-bit DM memory.

In both data input files, the first memory element corresponds to the total number of the following values. Moreover, coordinate and ToT are written in the hexadecimal notation for compactness. For clearness, both files include a commented part to explicitly show coordinate, ToA, and ToT values for each information.
Finally, only three kinds of events at different ToA windows have been used as input data due to run-time limitations: a single pixel without neighbors, a large cluster event and a small cluster event.

4.1.2 Application-specific algorithm

Figure 4.3 shows the flow chart of the application-specific data processing algorithm implemented in this thesis.

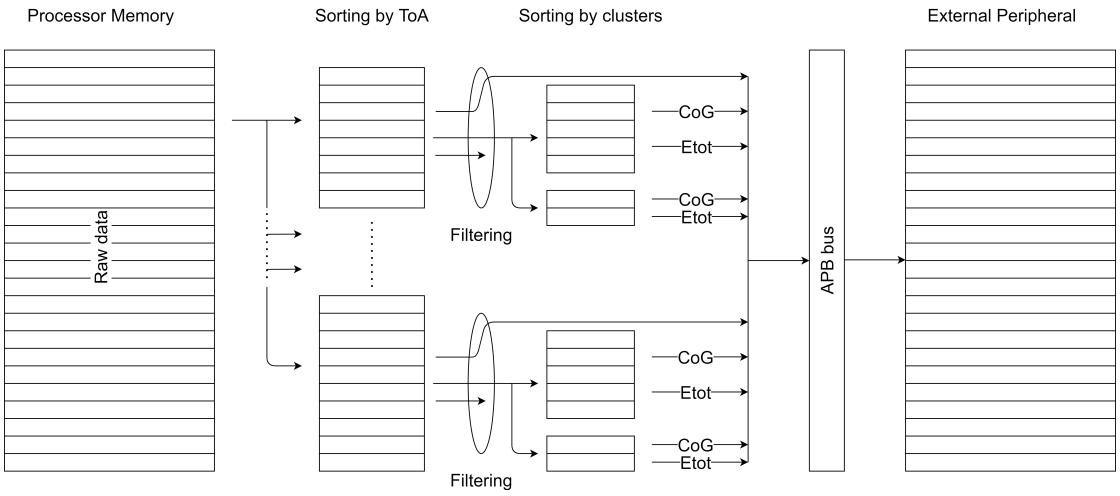

Figure 4.3: Flow chart of the application-specific data processing algorithm.

The algorithm has been divided into four different steps:

Sorting by ToA

The first step consists of sorting input data depending on ToA values. Since input data already comes as a partly chronologically sorted stream of pixel hits, this step just consists of fetching input data. This fetching by ToA takes into account a clustering time window of 200 ns equivalent to 8 cycles for a clock frequency of 40 MHz.
During this step, x and y coordinates are computed from the single input coordinate value (idx) for each pixel as follows:

$$\begin{aligned} x &= idx\%256\ [integer] \\ y &= idx/256\ [integer] \end{aligned} \tag{4.1}$$

Filtering and Sorting by clusters

The second step consists of two parts:

- Filtering: this part consists of background cancellation. A single hit pixel without any neighbors is probably associated with some kind of noise thus it gets filtered out. On the other hand, if cluster size exceeds the computational boundary conditions, all the data values belonging to the same ToA are directly forwarded towards the output to avoid any information loss. These filtering conditions can be specified depending on the application.

- Clustering: this part consists of checking for neighbors around a specific area. A graphical representation of the clustering algorithm is provided in figure 4.4.

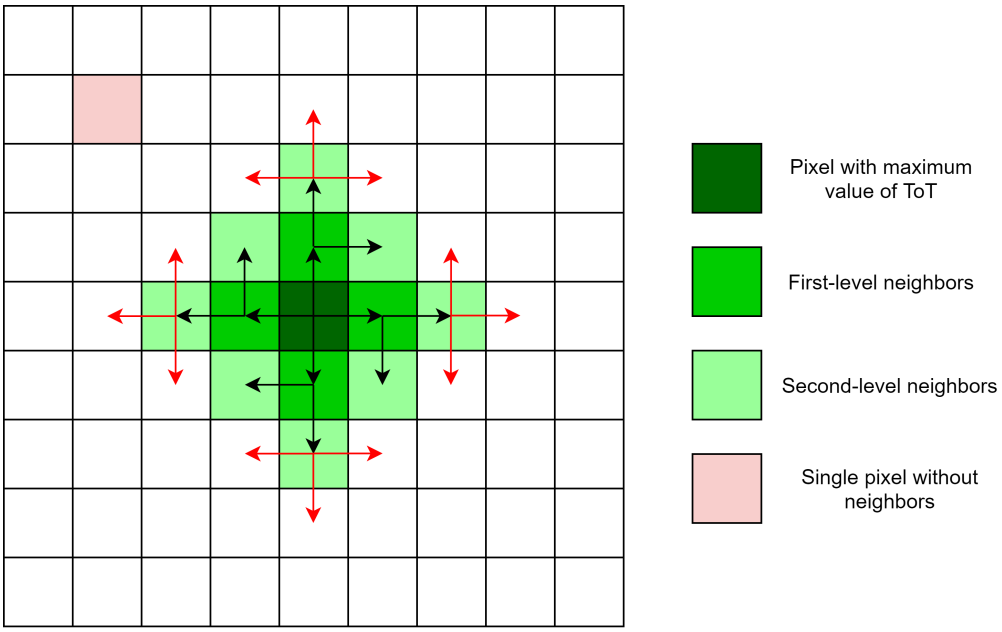

Figure 4.4: Graphical representation of clustering algorithm.

The clustering algorithm starts by searching for a pixel with the maximum value of ToT in a certain ToA window (dark green pixel). This pixel is used as a starting point by the clustering algorithm. From this pixel, the algorithm searches for first-level neighbors and if any are present, it searches for second-level neighbors around the first level. For instance, in figure 4.4 the pixel with the maximum value of ToT is represented in dark green. Then, it looks for a first-level neighbor towards the east. If this neighbor is present, it looks for second-level neighbors towards its east and south. The same principle is applied in the south, west, and north directions. If any neighbor is present from a second-level neighborhood (red arrows), the algorithm rises an "out of border" flag that immediately outputs all data belonging to the same ToA window towards the external peripheral and fetches the next ToA window. Otherwise, if a clustering event is finished and non-processed data values belonging to the same ToA window are still present, the algorithm searches for the next maximum value among the remaining data as the next starting point.
Furthermore, if a single pixel without neighbors is present (pink pixel), it gets filtered out.
The clustering algorithm implements two levels of nested functions. Such depth

can be extended for specific applications depending on how many nested functions the processor can sustain.

Computation of Centre of Gravity and total energy

The third step consists of computing the Centre of Gravity and total energy while regrouping pixels by clusters as follows:

$$\begin{aligned} x_{tot} &= \sum_{i=0}^{n} x_i \cdot E_i \\ y_{tot} &= \sum_{i=0}^{n} y_i \cdot E_i \\ E_{tot} &= \sum_{i=0}^{n} E_i \end{aligned} \tag{4.2}$$

Values of x_{tot}, y_{tot}, E_{tot} are sufficient to fully describe Centre of Gravity of a cluster and its total energy.

Output to External Peripheral

The fourth and last step consists of outputting Centre of Gravity and total energy information towards an external peripheral through the APB interface introduced in section 3.5.1. In this case, the external peripheral is the Interface Memory which is used as a testing feature. Moreover, if input data exceeds the clustering boundary conditions, all data from the same ToA window are outputted too. The output function consists of writing this information in specific allocations of Interface Memory with increasing address. The output values are visible in register change dump files during verification of processor functionality.

4.2 Processor Profiling

Profiling is a sophisticated optimization approach and dynamic program analysis that reorders code inside processes or between operations based on statistical information obtained while the program is executing [51]. Profiling does not impact the program's syntactic functionality however it can increase performance relying on the kind of application. At both the instruction and function levels, profile data is accessible. Profiling reveals which sections of the application software receive the majority of cycles [52]. With the use of profiling, users may locate the application and architectural performance bottlenecks, as well as learn how much code is covered and find uncovered functionality.
ASIP Designer provides the following profiling reports:

- Instructions profiling
- Functional units profiling
- Primitive operations profiling
- Instruction classes profiling
- Hazards profiling
- nML coverage profiling
- Storages accesses profiling

For each kind of profiling, reports on the application-specific data processing algorithm, presented in section 4.1, applied to both Trv32p3 and Tmicro microprocessors will be shown.

Instructions profiling

Instruction profiling reorganizes processor instructions used during the execution of the program code and regroups them by functions.

Listing 4.4: Instruction profiling report of Trv32p3 microprocessor

```
    Total cycle count             :              2560
    Report cycle count            :              2560
    Total instruction count       :              2144
    Report instruction count      :              2144
    Report instruction coverage   :            71.64%
    Total size in program memory  :              2948

Function summary:

Cycles % of total Instruction % of total % Coverage  Function
------ ---------- ----------- ---------- ---------- ----------------
1188     46.41%        952      44.40%     55.86%  sorting_by_clusters
617      24.10%        559      26.07%    100.00%  fetching_by_ToA
322      12.58%        240      11.19%    100.00%  output_to_IM
318      12.42%        288      13.43%    100.00%  main
104       4.06%         96       4.48%    100.00%  computing_CoG_Etot
  6       0.23%          6       0.28%    100.00%  _main_init
  2       0.08%          1       0.05%     33.33%  _start_basic
```

Listing 4.5: Instruction profiling report of Tmicro microprocessor

```
    Total cycle count             :              4284
    Report cycle count            :              4284
```

```
    Total instruction count      :              3406
    Report instruction count     :              3406
    Report instruction coverage  :            69.50%
    Total size in program memory:              1252

Function summary:

Cycles % of total Instruction % of total % Coverage  Function
------ ---------- ----------- ---------- ----------  ----------------
  2275     53.10%        1677     49.24%     53.67%  sorting_by_clusters
   854     19.93%         762     22.37%    100.00%  fetching_by_ToA
   578     13.49%         448     13.15%     99.19%  output_to_IM
   443     10.34%         396     11.63%    100.00%  main
   112      2.61%         112      3.29%    100.00%  computing_CoG_Etot
    16      0.37%           8      0.23%    100.00%  test_init
     2      0.05%           1      0.03%     33.33%  _start_basic
```

Instruction profiling report of Trv32p3 microprocessor, in listing 4.4, presents 2560 cycles while Tmicro microprocessor, in listing 4.5, presents 4284 cycles to perform the complete data processing algorithm for three events. This is because Trv32p3 has a 32-bit architecture while Tmicro has a 16-bit architecture thus it takes more cycles to perform the same tasks.
The total instruction count is respectively 2144 and 3406 cycles due to the presence of multi-cycle instructions.
Instruction coverage of respectively 71.64 % and 69.50 % mean not all instructions present in Trv32p3 and Tmicro microprocessors have been used, thus, optimization of approximately 30 % might be reached in terms of instruction-set. This also needs to take into account that instruction coverage changes depending on application and input data.
Function summary is also present in both reports and shows the list of functions sorted by cycle count. In both cases, *sorting_by_clusters* is the function in which most cycles are spent occupying approximately half of the total cycle count. This function needs a huge work of optimization to lower its cycle count. Moreover, the algorithm is covering approximately 50 % of this function code. After running a complete set of real input data, it is possible to better understand which parts of this sorting function are the most used in order to optimize them. Function $fetching\_by\_ToA$ takes fewer cycles and its C code is fully covered. The same occurs to *output_to_IM* and *computing_CoG_Etot* functions.
The last two functions in both listings are external C file functions used to initiate the algorithm.
The original instruction profiling reports are also provided with a complete description of each function in terms of instructions such as Program Counter value, assembly code, execution count, cycles, and wait states. This information has been

omitted in these listings for compactness.

Instruction profiling helps a lot to understand which functions require most of the cycles but it is incomplete. Other kinds of profiling reports are necessary to have a complete understanding of optimization space.

Functional units profiling

Functional units profiling reorganizes processor functional units used during the execution of the program. Here, functional units are sorted out by count occurrences and also by functions. Listing 4.6 and listing 4.7 respectively show profiling of functional units of Trv32p3 and Tmicro microprocessors. The first column presents how many times a functional unit is used for more complex instructions. The percentage is related to the total instruction count presented during instructions profiling.

Listing 4.6: Functional units profiling report of Trv32p3 microprocessor

Count	% Count	Functional units	Func Count	% Func Count	Function name
1016	47.39%	DMw	483	47.54%	sorting_by_clusters
			259	25.49%	fetching_by_ToA
			137	13.48%	main
			89	8.76%	output_to_IM
			48	4.72%	computing_CoG_Etot
1016	47.39%	agu	483	47.54%	sorting_by_clusters
			259	25.49%	fetching_by_ToA
			137	13.48%	main
			89	8.76%	output_to_IM
			48	4.72%	computing_CoG_Etot
790	36.85%	alu	317	40.13%	sorting_by_clusters
			237	30.00%	fetching_by_ToA
			119	15.06%	main
			88	11.14%	output_to_IM
			24	3.04%	computing_CoG_Etot
			5	0.63%	_main_init
711	33.16%	lx	388	54.57%	sorting_by_clusters
			172	24.19%	fetching_by_ToA
			68	9.56%	output_to_IM
			59	8.30%	main
			24	3.38%	computing_CoG_Etot
292	13.62%	pca	152	52.05%	sorting_by_clusters
			57	19.52%	fetching_by_ToA
			47	16.10%	output_to_IM
			27	9.25%	main
			8	2.74%	computing_CoG_Etot

```
                                          1    0.34%  _start_basic
 232    10.82%    cmp                  131   56.47%  sorting_by_clusters
                                         51   21.98%  fetching_by_ToA
                                         39   16.81%  output_to_IM
                                         11    4.74%  main
  16     0.75%    mpy                   16  100.00%  computing_CoG_Etot
  15     0.70%    IM                    15  100.00%  output_to_IM
  15     0.70%    im_agu                15  100.00%  output_to_IM
```

Listing 4.7: Functional units profiling report of Tmicro microprocessor

```
Count    % Count  Functional units  Func Count  % Func Count  Function name
-------  -------  ----------------  ----------  ------------  --------------------
1947     57.16%   DM                1051        53.98%        sorting_by_clusters
                                    398         20.44%        fetching_by_ToA
                                    278         14.28%        main _main
                                    172          8.83%        output_to_IM
                                     48          2.47%        computing_CoG_Etot
1285     37.73%   ag1               759         59.07%        sorting_by_clusters
                                    257         20.00%        main _main
                                    160         12.45%        fetching_by_ToA
                                    109          8.48%        output_to_IM
441      12.95%   alu               154         34.92%        fetching_by_ToA
                                    143         32.43%        sorting_by_clusters
                                    103         23.36%        output_to_IM
                                     33          7.48%        main _main
                                      8          1.81%        computing_CoG_Etot
245      7.19%    Tmicro_IM         169         68.98%        sorting_by_clusters
                                     49         20.00%        output_to_IM
                                     20          8.16%        fetching_by_ToA
                                      7          2.86%        main _main
221      6.49%    im_ag1            169         76.47%        sorting_by_clusters
                                     25         11.31%        output_to_IM
                                     20          9.05%        fetching_by_ToA
                                      7          3.17%        main _main
  16     0.47%    mul               16         100.00%        computing_CoG_Etot
  11     0.32%    sh                11         100.00%        fetching_by_ToA
   5     0.15%    dlflg              5         100.00%        output_to_IM
```

It is noticeable how Data Memory and address generation functional units take most cycles in both microprocessors. Additionally, in both microprocessors, Data Memory functional unit is called most of the times in *sorting_by_clusters* function followed by $fetching\_by\_ToA$, *main*, *output_to_IM* and *computing_CoG_Etot*. Moreover, the third most used functional unit in both processors is the ALU unit. In Trv32p3, the following functional units are: *lx* used for load operations of central register files, *pca* used as Program Counter adder in control instructions, *cmp* used as a comparator in control instructions, *mpy* used for multiplication operations,

IM is the functional unit representing Interface Memory accesses and im_agu as address generation unit of Interface Memory.
On the other hand, functional units following alu in Tmicro microprocessor are: $Tmicro\_IM$ representing Interface Memory accesses, im_ag1 as address generation unit of Interface Memory, mul used for multiplication operations, sh used as a shifter and $dlflg$ as a hardware loop flag update unit.
Regardless of all similarities, the functional unit for Interface Memory is different. In Trv32p3, Interface Memory is accessed only during the $output\_to\_IM$ function, as it should be, while Tmicro accesses it also during other functions. This shows a flaw in the Tmicro microprocessor which increases its cycle count.

Optimizations might be implemented in these functional units to lower their cycle count and be more application-specific. But it is still difficult which parts of such functional units are used and not. For further understanding, it is important to look at the building blocks of functional units: primitive operations. Moreover, the unused functional units, such as the divider, might be removed to optimize both microprocessors in terms of area.

Primitive operations profiling

Similar profiling can also be performed on primitive operations. Listing 4.8 and listing 4.9 respectively show profiling of primitive operations of Trv32p3 and Tmicro microprocessors. For compactness, only a few primitive operations have been shown. In both listings, the first column represents the number of times a certain primitive operation is used for more complex instructions. The percentage is related to the total instruction count shown during instructions profiling.

Listing 4.8: Primitive operations profiling report of Trv32p3 microprocessor

Count	% Count	Primitive operation
2762	128.82%	rd X
2011	93.80%	add
1546	72.11%	wr X
711	33.16%	rd DMw
305	14.23%	wr DMw
272	12.69%	rd PC_ID
232	10.82%	br
102	4.76%	ne
...	...	...
15	0.70%	wr IM
...	...	...

Listing 4.9: Primitive operations profiling report of Tmicro microprocessor

```
1        Count     % Count  Primitive operation
2 ------------- ---------- ----------------------
3         2483      72.90% rd R
4         2332      68.47% wr R
5         1730      50.79% add
6         1416      41.57% rd DM
7         1246      36.58% rd SP
8          531      15.59% wr DM
9          235       6.90% jump
10         221       6.49% rd Tmicro_IM
11         ...        ...  ...
12          24       0.70% wr Tmicro_IM
13         ...        ...  ...
```

In Trv32p3, *rd X* and *wr X* respectively correspond to read and write primitive operations on central register file *X*. Similarly, in Tmicro, *rd R* and *wr R* respectively correspond to read and write primitive operations on central register file *R*. From both listings, it is noticeable how the total cycle count, in both microprocessors, is extremely dominated by the accesses to central register files.
Furthermore, also primitive operations for reading and writing on Data Memory strongly influence total cycle count.
Moreover, another primitive operation that impacts a lot the total cycle count is the addition *add* in both microprocessors

Accesses to central register files and Data Memory must be optimized to hugely decrease total cycle counts. Another optimization might be introduced in ALU since it exploits a lot of *add* primitive operations.

Instruction classes profiling

Instructions can also be reorganized into classes for a better understanding of profiling. In the processor model, a *class* attribute might be associated with some instructions to collect them under the same instruction class. Listing 4.10 and listing 4.11 respectively show profiling of instruction classes of Trv32p3 and Tmicro microprocessors. For compactness, only a few instruction classes have been shown. In both listings, the first column represents the number of times a certain instruction class is used for more complex instructions. The percentage is related to the total instruction count shown during instructions profiling.

Listing 4.10: Instruction classes profiling report of Trv32p3 microprocessor

```
1        Count     % Count  Instruction class
2 ------------- ---------- ----------------------
3         2142      99.91% i_class_chckrs_Trv32p3
4         1590      74.16% i_class_mX1w
5         1051      49.02% i_class_mX3r
```

```
      1038      48.41% i_class_mX1r
      1016      47.39% i_class_ldst
       711      33.16% i_class_load
       685      31.95% i_class_alu_rri
       673      31.39% i_class_mX2r
       305      14.23% i_class_store
       292      13.62% i_class_ctrl
       232      10.82% i_class_branch
       105       4.90% i_class_alu_rrr
        40       1.87% i_class_jal
        20       0.93% i_class_jalr
        16       0.75% i_class_mpy
        15       0.70% i_class_im_load_store
        15       0.70% i_class_im_store
       ...        ...  ...
```

Listing 4.11: Instruction classes profiling report of Tmicro microprocessor

```
      Count     % Count Instruction class
___________ ___________ ____________________________________
      3404      99.94% i_class_chckrs_Tmicro
      3404      99.94% i_class_Tmicro
      2400      70.46% i_class_wreg
      2386      70.05% i_class_r_reg
      1971      57.87% i_class_load_store_instr
      1845      54.17% i_class_wreg_w
      1836      53.90% i_class_r_reg_w
      1217      35.73% i_class_load_store_wreg_sp_indexed
      1217      35.73% i_class_sp_indexed
       975      28.63% i_class_rrid
       975      28.63% i_class_rrid_r
       730      21.43% i_class_ag1_opn
       730      21.43% i_class_load_store_wreg_indirect
       575      16.88% i_class_wreg_r
       565      16.59% i_class_r_reg_r
       512      15.03% i_class_control_instr
       475      13.95% i_class_rs
       475      13.95% i_class_rs_r
       468      13.74% i_class_alu_instr
       468      13.74% i_class_rr
       468      13.74% i_class_rr_r
       265       7.78% i_class_rrid_w
       245       7.19% i_class_im_ag1_opn
       245       7.19% i_class_im_load_store_instr
       245       7.19% i_class_im_load_store_wreg_indirect
       ...        ...  ...
```

In both listings, the first class is just associated with *CHECKERS* which is the Instruction Set Simulator of ASIP Designer. As shown in primitive operations

profiling, read and write accesses to central register files are the most impacting instructions on total cycle count. In Trv32p3, i_class_mX1w represents a write access mode during execute stage while i_class_mX1r and i_class_mX2r represent two different read access modes during execute stage and i_class_mX3r represents a read access mode during decode stage to central register file X. Similar behavior also occurs in the Tmicro microprocessor.
Moreover, also load and store instructions to Data Memory impact a lot the total cycle count, as shown previously in primitive operations profiling.

Instruction classes profiling confirms what was stated before in primitive operations profiling: accesses to central register files and Data Memory must be optimized to hugely decrease total cycle count.

Hazards profiling

During the application-specific algorithm, several hazards occur due to pipeline stages implemented in both microprocessors. Listing 4.12 and listing 4.13 respectively show introduced stalls in Trv32p3 and Tmicro during the execution of the algorithm to solve such hazards. These stall rules are described in both processor models with a special *class* attribute to ease profiling and debugging.

Listing 4.12: Hazards profiling report of Trv32p3 microprocessor

```
Hardware stalls by class:

Count  % Count  Class name          Func Count  % Func Cnt  Function name
-----  -------  ------------------  ----------  ----------  ---------------
11     0.43%    agu_read_after_write  6         54.55%      fetching_by_ToA
                                      5         45.45%      sorting_by_clusters
```

Listing 4.13: Hazards profiling report of Tmicro microprocessor

```
Software stalls by class:

Count  % Count  Class name          Func Count  % Func Cnt  Function name
-----  -------  ------------------  ----------  ----------  ---------------
152    3.55%    dm_addr_conflict      63        41.45%      sorting_by_clusters
                                      63        41.45%      fetching_by_ToA
                                      16        10.53%      computing_CoG_Etot
                                       6         3.95%      output_to_IM
                                       4         2.63%      main _main
 48    1.12%    read_after_write_R    15        31.25%      output_to_IM
                                      14        29.17%      sorting_by_clusters
                                      11        22.92%      fetching_by_ToA
                                       8        16.67%      computing_CoG_Etot
 21    0.49%    delay_slot             8        38.10%      output_to_IM
```

```
15                                          8     38.10%  computing_CoG_Etot
16                                          4     19.05%  sorting_by_clusters
17                                          1      4.76%  fetching_by_ToA
18    7     0.16%  between_loop_ends        7    100.00%  output_to_IM
19    4     0.09%  offs_conflict            4    100.00%  sorting_by_clusters
```

In Trv32p3, eleven Read After Write (RAW) data hazards occur on the AGU unit: six during *fetching_by_ToA* and five during *sorting_by_clusters*. They occur because AGU uses results from other units. To solve such hazards, a one-cycle hardware stall named *agu_read_after_write* is used for load and store instructions and it is explicitly described in the processor model of Trv32p3. Other hardware and software stalls are described in the processor model too but they do not occur during this algorithm.

On the other hand, when the algorithm is running on the Tmicro microprocessor, 232 hazards occur. These hazards are solved through software stalls described in the processor model of Tmicro. *dm_addr_conflict* is a software stall of one cycle that solves structural hazards occurring on the address of Data Memory. *read_after_write_R* is a software stall of one cycle that solves Read After Write (RAW) data hazards occurring on central register file *R*. Additionally, other *delay_slot* are used. *between_loop_ends* is required to avoid two nested hardware loops ending at the same address. This occurs in the *output_to_IM* function since it is the last nested function of the algorithm.

Hazards profiling reports a good understanding of where and how hazards are present and solved by stalls or bypasses.

nML coverage profiling

nML coverage profiling report shows how much of the processor model is used in terms of operations. Listing 4.14 and listing 4.15 respectively show the ratio of nML operations used for Trv32p3 and Tmicro microprocessors. For compactness, only a small part of nML coverage profilings have been shown.

Listing 4.14: nML coverage profiling report of Trv32p3 microprocessor

```
1 Trv32p3  (0.178988)
2  .i32fmt  (0.178988)
3    .alu_instrs  (0.134167)
4      .alu_rrr_ar_instr  (0.05625)
5        +majOP_fn10  (0.2)
6          -add  (1)
7          -sll  (0)
8          -slt  (0)
9          -sltu  (0)
```

```
10          -xor   (0)
11          -srl   (0)
12          -or   (0)
13          -and   (1)
14          -sub   (0)
15          -sra   (0)
16        +mX1w_EX   (0.28125)
17          +mX1w   (0.28125)
18            +eX   (0.28125)
19              ...   (...)
20        +mX1r_EX   (1)
21          ...                   (...)
```

Listing 4.15: nML coverage profiling report of Tmicro microprocessor

```
1  Tmicro   (0.263301)
2   .alu_instr   (0.41336)
3     .alu_rrr   (0.428571)
4       +alu_op   (0.428571)
5         -add   (1)
6         -addc   (0)
7         -sub   (1)
8         -subb   (0)
9         -and   (1)
10        -or   (0)
11        -xor   (0)
12      +rt   (1)
13      +rr   (1)
14      +rs   (1)
15    .shift_rrr   (0.333333)
16    ...       (...)
```

The algorithm uses approximately 18 % of Trv32p3 microprocessor while it uses approximately 26 % of Tmicro microprocessor. For instance, *add* and *and* are the only operations used in ALU instruction-set *alu_instrs* of Trv32p3 microprocessor. Thus all the other operations might be removed since not necessary. Similarly, in the Tmicro microprocessor, only *add*, *sub* and *and* are used so the other operations can be removed.

From nML coverage reports, it is easier to identify which operations are not used by the application-specific test code and remove them to decrease processor model complexity.

Storages accesses profiling

Another possible profiling consists of storage accesses profiling. Listing 4.16 and listing 4.17 respectively show the storages accesses of Data Memory and Interface

Memory done by Trv32p3 and Tmicro microprocessors. For compactness, only Data Memory and Interface Memory are shown in these listings but information about other memories is possible such as Program Memory, Program Counter, central register file, etc.

Listing 4.16: Storages accesses profiling report of Trv32p3 microprocessor

```
Function storage access summary:

Storage  Read        Read          Write     Write       Function name
         count       count         count     count
         (total)  (function)     (total)  (function)
-------  -----  -------------  --------  -------------  --------------------
DMb       2844     96  (3.38%)    1220     96  (7.87%)  computing_CoG_Etot
                  688  (24.19%)            348  (28.52%)  fetching_by_ToA
                  236  (8.30%)             312  (25.57%)  main
                  272  (9.56%)              84  (6.89%)   output_to_IM
                 1552  (54.57%)            380  (31.15%)  sorting_by_clusters
IM           0      0   (0.00%)      15     15  (100.00%) output_to_IM
...        ...    ...      ...       ...    ..    ...         ...
```

Listing 4.17: Storages accesses profiling report of Tmicro microprocessor

```
Function storage access summary:

Storage  Read        Read          Write     Write       Function name
         count       count         count     count
         (total)  (function)     (total)  (function)
-------  -----  -------------  --------  -------------  --------------------
DM        1416     24  (1.69%)     531     24  (4.52%)   computing_CoG_Etot
                  280  (19.77%)            118  (22.22%)  fetching_by_ToA
                  171  (12.08%)            107  (20.15%)  main
                  126  (8.90%)              46  (8.66%)  output_to_IM
                  815  (57.56%)            236  (44.44%)  sorting_by_clusters
Tmicro_IM 221     20  (9.05%)      24      0  (0.00%)   fetching_by_ToA
                    7  (3.17%)              0  (0.00%)   main _main
                   25  (11.31%)            24  (100.00%) output_to_IM
                  169  (76.47%)             0  (0.00%)   sorting_by_clusters
...        ...    ...      ...       ...    ..    ...         ...
```

In the Trv32p3 microprocessor, Data Memory is accessed 2844 times in read mode and 1220 times in write mode, mostly during the *sorting_by_clusters* function. Similarly, in the Tmicro microprocessor, Data Memory is accessed 1416 times in read mode and 531 times in write mode and also here mostly during the *sorting_by_clusters* function. During such a function, both microprocessors access Data Memory one by one for each single input value and this might be optimized. Moreover, Interface Memory is accessed only in write mode by Trv32p3 as it should be. On the other hand, Tmicro presents some errors since it accesses

Interface Memory also in read mode which should not occur as previously presented in functional units profiling.

4.3 Final considerations

The application-specific algorithm of event clustering proves the functionality of both microprocessors. Moreover, profiling results show a complete picture of the processor model utilization. Both microprocessors use approximately 70 % of their instruction set so an improvement of almost 30 % can be achieved. Furthermore, it shows that access to central register files as well as to Data Memory, in both microprocessors, are the most impacting primitive operations in terms of the number of cycles. A huge improvement must be implemented to drastically decrease this cycle count. On the other hand, unused functional units such as the divider can be removed from both processor models. Finally, all these results depend on the application and input data set, thus they might differ from one case to another.

In this chapter, information regarding latency of data processing has been extracted from both microprocessors. This information is necessary to improve their performance. Nevertheless, information regarding physical implementation is still missing. Chapter 5 provides a complete description of the physical implementation of both Trv32p3 and Tmicro microprocessors. Figures of merit concerning frequency, area occupation, and power consumption are presented as well as additional frequency optimizations.

Chapter 5

Physical implementation

Once the RTL code is generated and tested, to complete an ASIC design flow, physical implementation is required. This is composed of two parts: RTL synthesis and Place and Route (PnR). The first one exploits specific tools to map the RTL code into a gate-level netlist through specific technologies libraries. Logic optimizations are also performed during this part to meet specific time constraints. Moreover, also a preliminary power analysis can be performed at this stage.
Although, for area information, the second step is needed: PnR consists of a first part of setting down a floorplan of the chip in which standard cells are placed. The floorplan also gives information about the total size of the chip. It is important to carefully place blocks and pins to ease the routing part of metal wires to interconnect them but also to minimize the total area of the final chip. Moreover, the placement is done to minimize the timing and length of connections to reduce power dissipation. Then, since the clock signal must be equally distributed to all flops in the chip, Clock Tree Synthesis (CTS) is performed. It introduces buffers and inverters along the clock tree to balance its delay and decrease power consumption. Finally, routing of metal wires is performed to interconnect all blocks taking into account fan-out wire delays.
At the end, such a physical implementation is verified through:

- Design Rule Checking (DRC) to check if the design meets the geometry rules provided by the manufacturing foundry
- Logic Equivalence Checking (LEC) to check the equivalency between a reference design and the revised design through mathematical modeling techniques
- Layout versus schematic (LVS) to check for equivalency between the extracted layout netlist and schematic synthesis netlist

This digital flow from RTL to PnR has been performed on the Cadence platform.

The flow is based on the YAML language which is a human-friendly data serialization language able to manage Cadence Stylus flowKit from a higher level. Such a flow serially performs the following steps: synthesis generic, synthesis mapping, synthesis optimization, floorplan, pre-CTS, CTS, post-CTS, route, post-route, optimization signoff, and export verify.
Cadence platform consists of several tools for different steps: Genus for synthesis, Innovus for implementation, Tempus for timing analysis and Voltus for power analysis.

In this thesis, a 28 nm CMOS technology has been used to compute performances of the physical implementation which presents tap-less standard cells to achieve small dimensions. Various standard cell libraries are present in such a technology depending on several combinations of gate length, threshold voltage, and the number of tracks. All of them present a pitch dimension of 140 nm.
Standard cell libraries present three different gate lengths: 30 nm, 35 nm, and 40 nm. Small gate length standard cells are faster and show a better Total Ionizing Dose (TID) response. On the other side, long gate length standard cells present less leakage.
Additionally, standard cells come with six different threshold voltage flavors: Ultra-Low Threshold Voltage (ULVT), Low Threshold Voltage (LVT), Standard Threshold Voltage (SVT), High Threshold Voltage (HVT), Ultra-High Threshold Voltage (UHVT) and Extremely-High Threshold Voltage (EHVT). Higher threshold voltages are used to lower power consumption and leakage but also the achievable frequency decreases. Oh the other hand, lower threshold voltages are used to push higher the frequency but also power consumption and leakage increase. Moreover, two extra flavors are present for Ultra Density Memories (UDM).
Finally, three different numbers of tracks are possible for these standard cells: 7-track, 9-track, and 12-track. This value represents the number of metal-1 lines available in the routing space of a cell. It is equivalent to the height of cells. Moreover, higher the number of tracks higher the speed due to higher current.

In this chapter, a fair comparison between the physical implementations of Trv32p3 and Tmicro microprocessors will be presented as well as the area comparison between post-synthesis, at *synthesis optimization* step, and post-PnR, at *optimization signoff* step. Moreover, several physical optimizations have been conducted on both microprocessors and important figures of merit are shown.

5.1 First implementation

The first implementation has been performed with the following set of constraints:

- clock period: 3 ns

- clock duty cycle: 50 %
- clock transition time: 0.2 ns
- clock uncertainty time: 0.2 ns
- clock uncertainty time for setup: 0.2 ns
- clock uncertainty time for hold: 0.05 ns
- inputs maximum fanout load: 1
- inputs transition time: 0.1 ns

All the presented implementations are routable and with correct timing closures. A general occupancy of 50 % of the target has been used for floorplan without any further constraints to see the achievable area.
Moreover, only one standard cells library has been used with the following characteristics:

- gate length: 35 nm
- threshold voltage: standard (SVT)
- track: 9-track

Place and Route implementations at signoff step of Trv32p3 and Tmicro microprocessors are presented respectively in figures 5.1 and 5.2.

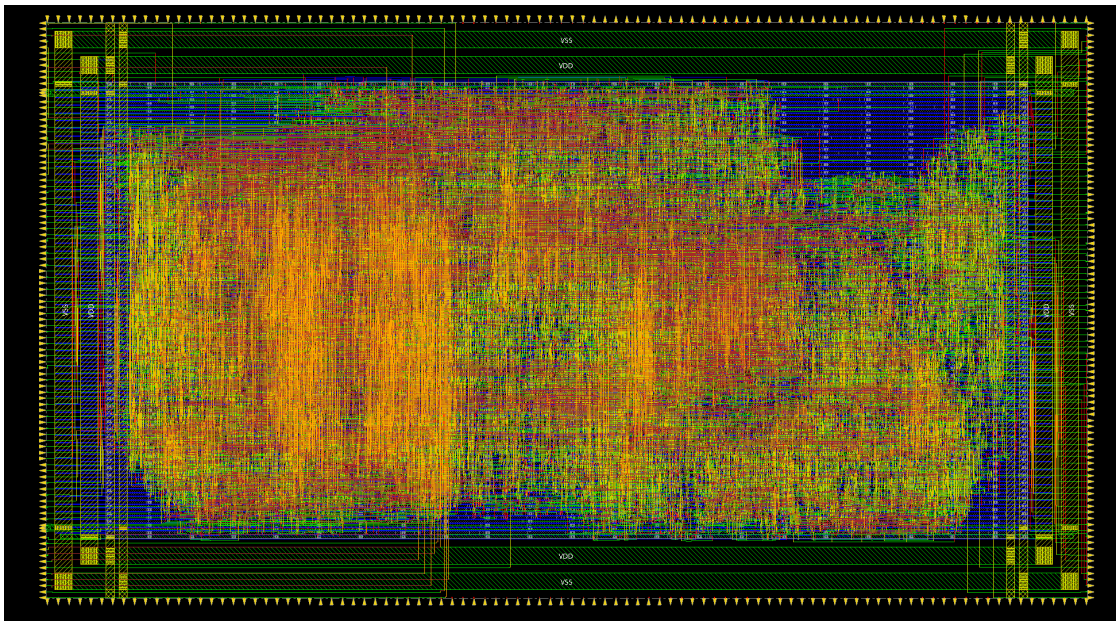

Figure 5.1: Place and Route implementation of Trv32p3 microprocessor.

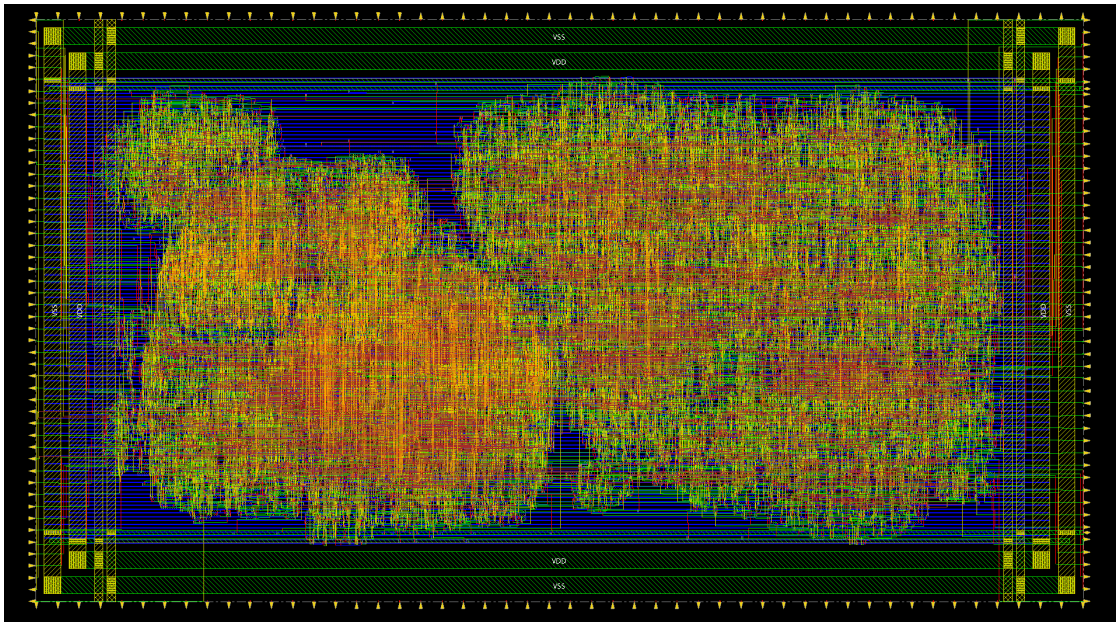

Figure 5.2: Place and Route implementation of Tmicro microprocessor.

	Microprocessor	Instances	Area [µm^2]	Clock [MHz]
Post Synthesis	Trv32p3	11859	12223	333
	Tmicro	12988	12993	333
Post PnR	Trv32p3	12152	12231	333
	Tmicro	13479	13105	333

Table 5.1: Summary table of total instances and area for Trv32p3 and Tmicro microprocessors after synthesis and after PnR.

After synthesis of RTL code, Trv32p3 microprocessor presents 11859 total instances in 12223 µm^2. On the other hand, the Tmicro microprocessor presents 12988 total instances in 12993 µm^2. It is immediately visible that both microprocessors present a similar area occupation and the total number of instances. Area information after synthesis gives preliminary knowledge about the final area occupation but it is still incomplete of placing, CTS, and routing details. Thus, it is important

to check their consistency after the complete implementation flow to check for its reliability. After PnR, the Trv32p3 microprocessor presents 12152 total instances in 12231 µm^2 with a density utilization of 51.61 %. On the other hand, the Tmicro microprocessor presents 13479 total instances in 13105 µm^2 with a density utilization of 52.14 %. It is noticeable again the similarity of area occupation between the two microprocessors. In addition, density utilization is still approximately 50 %. Moreover, design information about the area was already reliable above 99 % after synthesis. All these information are summarized in table 5.1.

From now on, only information after the complete implementation flow will be analyzed and presented for Trv32p3 and Tmicro microprocessors.

Power analysis has been conducted on both microprocessors to extract important information regarding power consumption as internal power, switching power, and leakage power. This analysis has been managed by changing input activity and dominant frequency. Figures 5.3 and 5.4 represent summary graphs of total power consumption of respectively Trv32p3 and Tmicro microprocessors. In both graphs, total power consumption increases with both frequency and input activity. Moreover, Tmicro presents higher power consumption with respect to Trv32p3 for all values of frequency and input activity.

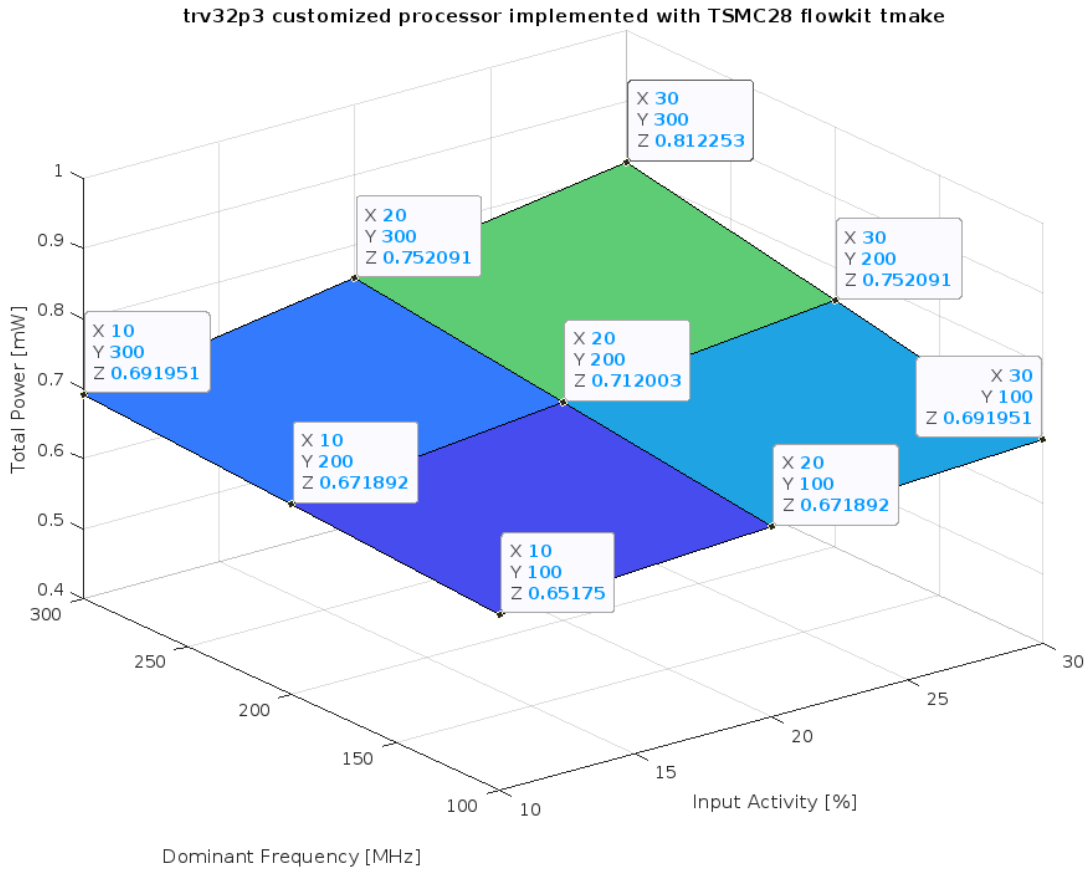

Figure 5.3: Total power consumption graph of Trv32p3 microprocessor at different frequencies and input activities.

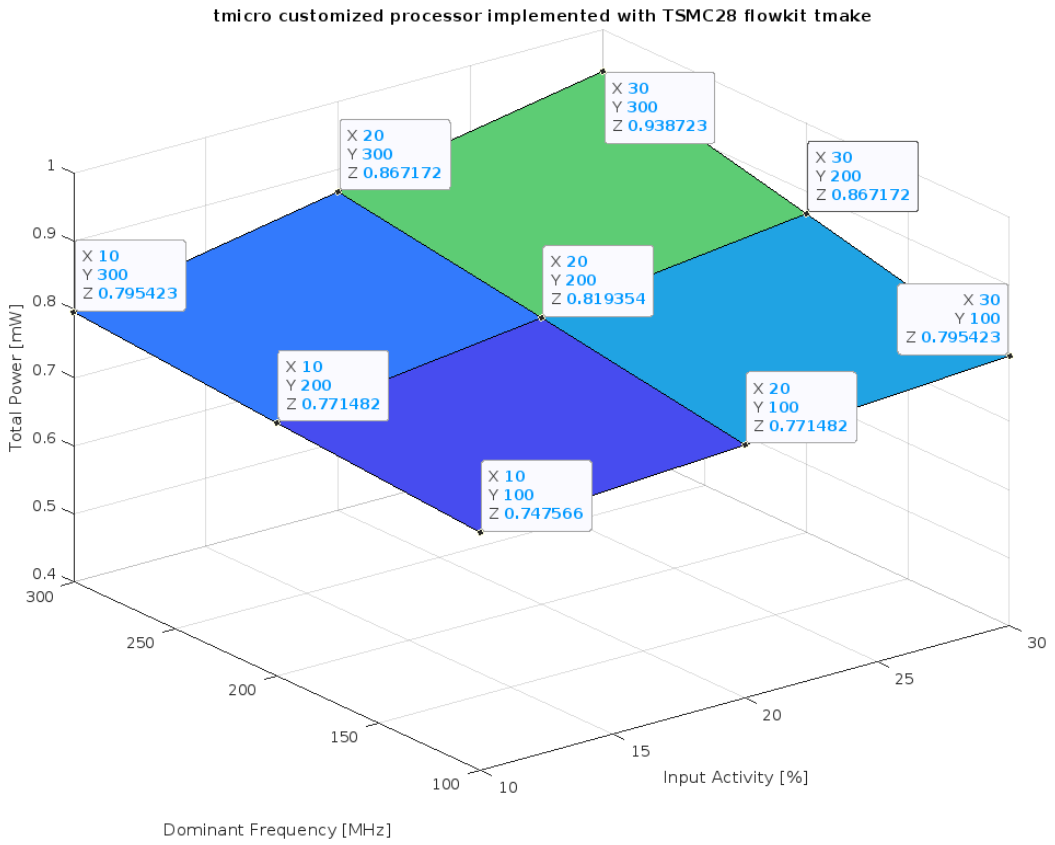

Figure 5.4: Total power consumption graph of Tmicro microprocessor at different frequencies and input activities.

The figure 5.5 shows a summary table of power analysis of the Trv32p3 microprocessor in which all the different kinds of power components are present. It is noticeable how internal power contributes the most to the total power regardless of the input activity and frequency. Although, internal power does not change too much with input activity and frequency with a maximum increment of approximately 6 % from the lowest to the highest value. On the other hand, switching power shows lower values than internal power but it does change largely with input activity and frequency with a maximum increment of 61 % from the lowest to the highest value. At 30 % of input activity and frequency of 300 MHz, it contributes approximately 45 % of total power consumption. Finally, leakage power does not have a meaningful impact on total power with a contribution always lower than 3 %. Moreover, leakage power does not change at all with input activity and frequency. Similar behavior occurs in the Tmicro microprocessor.

Input Activity [%]	Dominant Frequency [MHz]	Total Power [mW]	Total Internal Power [mW]	Total Switching Power [mW]	Total Leakage Power [mW]
10	300	0.69195066	0.4154219	0.25956524	0.01696352
20	300	0.75209102	0.42408825	0.31103926	0.01696352
30	300	0.81225344	0.43277993	0.36250999	0.01696352
10	200	0.671892	0.41253524	0.24239325	0.01696352
20	200	0.71200289	0.41830983	0.27672955	0.01696352
30	200	0.75209104	0.42408825	0.31103927	0.01696352
10	100	0.65175038	0.4095886	0.22519826	0.01696352
20	100	0.671892	0.41253524	0.24239325	0.01696352
30	100	0.69195066	0.4154219	0.25956525	0.01696352

Figure 5.5: Summary table of power analysis of Trv32p3 microprocessor.

Power analysis comparison between these two microprocessors shows that the total power consumption of Tmicro is always larger than Trv32p3 by approximately 15 %. In addition, previously performed area analysis and summarized in table 5.1, presents Tmicro microprocessor with a larger area and also with a larger number of instances with respect to Trv32p3 microprocessor. These results are both in contradiction with the architectural nature of these microprocessors. Tmicro presents a limited instruction set compared to Trv32p3. Moreover, Tmicro is a 16-bit microprocessor while Trv32p3 is a 32-bit microprocessor. Thus, Tmicro should present lower area occupation with a lower number of instances and lower power consumption. The reason for such incongruity is better analyzed in the next section.

5.2 RTL optimization of Tmicro microprocessor

To better understand why Tmicro presents higher values of instances, area occupation, and power consumption compared to Trv32p3, the hierarchy of modules has been checked. Figure 5.6 shows a part of the hierarchy of instances present in the Trv32p3 microprocessor sorted by the number of standard cells. Similarly, figure 5.7 shows a part of the hierarchy of instances present in the Tmicro microprocessor sorted by the number of standard cells.

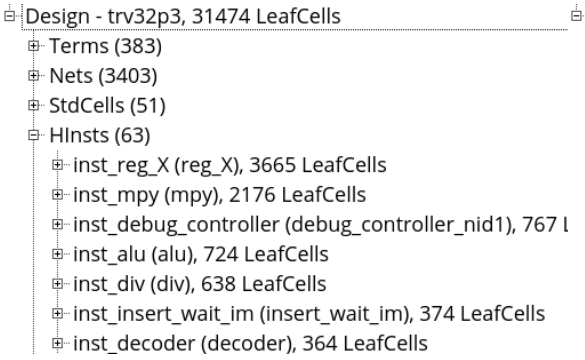

Figure 5.6: Hierarchy of Trv32p3 microprocessor instances.

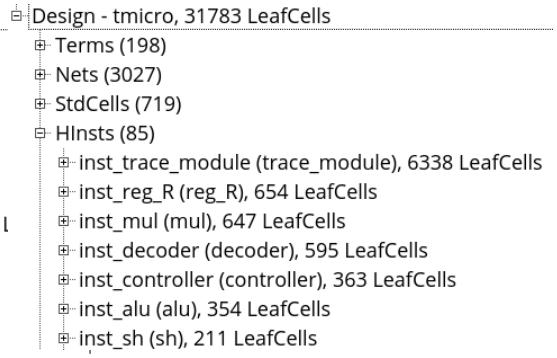

Figure 5.7: Hierarchy of Tmicro microprocessor instances.

The largest instance in Trv32p3 is the central register file X which is coherent. On the other side, in Tmicro, the central register file R is the second largest instance after the trace buffer module. This module is a trace buffer simply implemented as a ring buffer that stores sequentially a certain number of events. It is used for debugging purposes and it is often present on-chip to keep records of the Program Counter values of executed code. Such a module is not present in Trv32p3 microprocessor.
To have a fair comparison between these two microprocessors, the trace buffer module has been removed from Tmicro. This has been done by removing such a module from the processor model of Tmicro, thus inducing an optimization at the RTL level. First, the processor model of the Tmicro microprocessor without trace buffer has been correctly compiled and verified on ASIP Designer. Then, the generated RTL code has been synthesized and implemented under the same constraints and parameter settings of the first implementation flow used for Trv32p3 and original Tmicro.
The novel implementation at the signoff step of Tmicro without trace buffer is shown in figure 5.8.

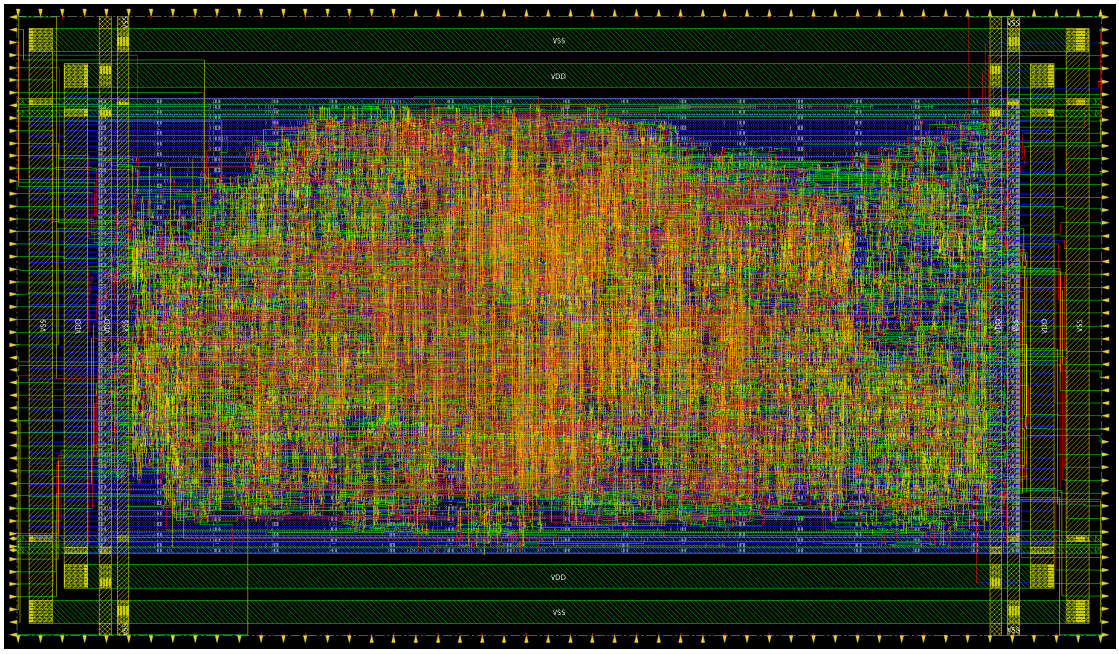

Figure 5.8: Place and Route implementation of Tmicro microprocessor without trace buffer.

Table 5.2: Summary table of total instances and area for Tmicro microprocessor without trace buffer after synthesis and after PnR.

	Microprocessor	Instances	Area [µm^2]	Clock [MHz]
Post Synthesis	Tmicro without trace buffer	6853	6067	333
Post PnR	Tmicro without trace buffer	7195	6147	333

Table 5.2 summarized area information of Tmicro microprocessor without trace buffer. It presents 6853 total instances in 6067 µm^2 after synthesis. Similar values are also present after PnR with 7195 total instances in 6147 µm^2 indicating that area information after synthesis is reliable. Moreover, density utilization is 52.48 %, still close to the nominal 50 % of the previous microprocessors. Tmicro without trace buffer shows an area of 47 % of the original Tmicro microprocessor, as shown

previously in table 5.1 denoting that trace buffer occupies approximately 53 % of the total area. Finally, the novel Tmicro without trace buffer now presents approximately half of total instances and area occupation with respect to the Trv32p3 microprocessor. This is coherent with the architectural differences between these two microprocessors.

Power analysis has also been performed on Tmicro without trace buffer at the same frequencies and input activities as previously performed analysis on Trv32p3 and original Tmicro. The results of total power consumption are summarized in figure 5.9.

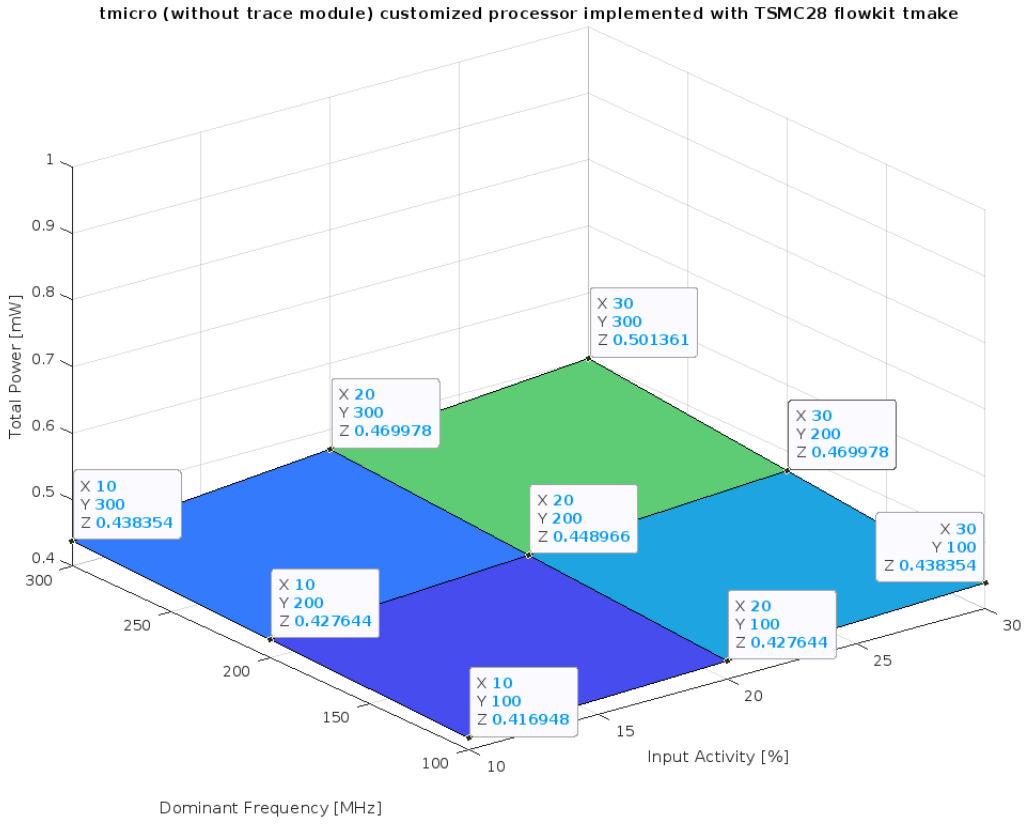

Figure 5.9: Power consumption of Tmicro microprocessor without trace buffer module at different frequencies and input activities.

Similarly to Trv32p3 and original Tmicro, also the total power consumption of Tmicro without trace buffer increases with both frequency and input activity. On the other hand, all values of power consumption are hugely reduced with respect to

Trv32p3 and original Tmicro of approximately 50% because of the lower number of instances.

Microprocessor	Trv32p3	Tmicro	Tmicro without trace buffer
Clock [MHz]	333	333	333
Instances	12152	13479	7195
Area [µm^2]	12231	13105	6147
Density [%]	51.61	52.14	52.48
Total power consumption [mW] (at 300 MHz and 30% input activity)	0.81	0.94	0.50

Table 5.3: Summary table of clock frequency, total instances, area, density, total power consumption for Trv32p3, Tmicro, and Tmicro without trace buffer microprocessors after PnR.

A summary of clock frequency, total instances, area occupation, density utilization, total power consumption at 300 MHz, and 30 % of input activity of Trv32p3, Tmicro, and Tmicro without trace buffer is present in the table 5.3.

During the first implementation, both microprocessors have been implemented using a clock frequency of 333 MHz and a perfect timing closure is present in all of them. In order to seek the maximum achievable frequency, the clock frequency has been increased.

5.3 Frequency improvement

From now on, optimization of frequency will be performed only on the Trv32p3 microprocessor, and results presented at the *optimization signoff* step will be shown. These results are similar to the Tmicro microprocessor.

5.3.1 First frequency improvement

To further push the microprocessor timing performance, the clock frequency has been doubled from 333 MHz to 666 MHz to see figures of merit of timing, area, and power consumption.
The following implementation has been performed with the same standard cells library (gate length of 35 nm, SVT flavor of threshold voltage, and 9-track) and also timing constraints set as for the first implementation with the exception of clock period which is now:

- clock period: 1.5 ns

This implementation presents the timing closure of the design. Nevertheless, the *route* step of the flow shows several paths with a negative slack. At this step, the worst negative slack is presented by the critical path that involves the 32-bit multiplier of Trv32p3. It starts from a register of central register file X, where input data for the multiplication are present; and it ends with another register of central register file X to store back the result of the multiplication. This critical path presents the worst negative slack of -7 ps, which is negligible considering the constraint on clock uncertainty of 200 ps. After this step, the *postroute* and *optimization signoff* steps increase timing performance reaching a timing closure of the design. The slack at *optimization signoff* step is positive and equal to 27 ps.

Clock [MHz]	333	666
Instances	12152	17465
Area [µm^2]	12231	18549
Density [%]	51.61	56.70
Total power consumption [mW] (at 300 MHz and 30% input activity)	0.81	2.91

Table 5.4: Summary table of total instances, area, density, and total power consumption after PnR for Trv32p3 microprocessor implemented at 333 MHz and 666 MHz of clock frequency.

Because of better performance, both area and power consumption increase compared

to the previous implementation at 333 MHz. The number of instances increases by approximately 44 % up to 17465. Similarly, area occupation increased by approximately 52 % up to 18549 μm^2. At the same time, density occupation slightly increases from 51.61 % to 56.70 %. Finally, the worst impact of doubling the clock frequency is on the total power consumption which increases by almost 260 % reaching 2.91 mW at 300 MHz and 30 % of input activity. All these information are summarized in the table 5.4.

5.3.2 Second frequency improvement

Since the design presents a timing closure at 666 MHz, this clock frequency has been pushed even further up to 1 GHz. The following implementation has been performed with the same standard cells library and also timing constraints set as for the previous implementations except for the clock period. The current implementation constraints and characteristics are the following:

- clock period: 1 ns
- clock duty cycle: 50 %
- clock transition time: 0.2 ns
- clock uncertainty time: 0.2 ns
- clock uncertainty time for setup: 0.2 ns
- clock uncertainty time for hold: 0.05 ns
- inputs maximum fanout load: 1
- inputs transition time: 0.1 ns
- reference utilization density: 50 %
- standard cells' gate length: 35 nm
- standard cells' threshold voltage: standard (SVT)
- standard cells' track: 9-track

The implementation of Trv32p3 using these constraints and standard cells completely fails at 1 GHz of clock frequency. The design does not present a timing closure and the critical path at all steps is the same as the one in the previous implementation at 666 MHz. It starts from a register of central register file X, then it goes through the multiplier and it finally reaches another register of central register files X to store back the result. At the end of PnR, regardless of all the optimization

steps during the flow, the worst negative slack at the *optimization signoff* step of this critical path is equal to -308 ps.

To reach timing closure of Trv32p3 design at 1 GHz, it is necessary to improve standard cell libraries. Several optimizations are going to be presented to close this timing.

Optimization 1

For this optimization, a novel standard cells library has been introduced in addition to the previous one and it presents the following characteristics:

- gate length: 35 nm
- threshold voltage: low (LVT)
- track: 9-track

Standard cells with low threshold voltage are able to reach higher speeds. This is because, at the same supply voltage, a lower threshold voltage produces a higher drain current. On the other hand, as threshold voltage decreases, leakage current increases which induces higher static power consumption. Moreover, due to higher current and speed, also dynamic power consumption increases.
The old standard cells library (gate length of 35 nm, SVT flavor of threshold voltage, and 9-track) is used from the beginning of the implementation while the novel standard cells library is activated only at the beginning of the *optimization signoff* step at the end of PnR flow. This has been done to maintain the overall power consumption low and to keep the same placing of cells and routing of wires. Thus, the only optimization consists of exchanging old standard cells along critical paths with the new ones to reach the timing closure.
Despite the new standard cells library, the design still does not present the timing closure. From the beginning of the flow until the *optimization signoff* step, critical paths are the same and present similar negative slacks as before. This is because the standard cell library used is the same. Nevertheless, at the end of the flow, the LVT standard cells library reduces the worst negative slack from -308 ps to -59 ps.

Optimization 2

Since optimization 1 does not present a timing closure, a better performing standard cells library is required. Similarly to optimization 1, two different standard cell libraries are used in this optimization. The first standard cells library has been

used from the beginning of the implementation for both synthesis and PnR. It presents the following characteristics:

- gate length: 30 nm
- threshold voltage: standard (SVT)
- track: 12-track

The second standard cell library has been introduced only at the *optimization signoff* step similarly to optimization 1. It presents the following characteristics:

- gate length: 30 nm
- threshold voltage: ultra-low (ULVT)
- track: 12-track

Both libraries present the smallest gate length and highest number of tracks possible.
Standard cells with shorter gate lengths present higher speed because as the gate length of the transistor decreases, the mobility of carriers increases and so does the transconductance. This effect is caused by a lower resistance and scattering events in the channel. On the other side, shorter gate length transistors lead to higher current leakage and power consumption.
Moreover, standard cells with a higher number of tracks show higher achievable speeds due to higher driving capability. On the other hand, the area occupation increases.
Finally, standard cells with an ultra-low threshold voltage, as the second library presented here, are able to reach higher speeds than LVT and SVT ones as explained in optimization 1.
The second library presents the best timing performance among all the possible libraries in 28 nm CMOS technology. It combines all three different effects to reach the maximum speed. On the other side, due to the same reasons, it presents the highest power consumption and area occupation.

Differently from optimization 1, optimization 2 does not present any path with negative slack during synthesis. Despite this, it still does not reach the timing closure of the design. The critical path is still the same which involves the 32-bit multiplier. Nevertheless, the worst negative slack at the end of the implementation reduces from -59 ps, presented in optimization 1, to -39 ps. This value is small compared to the constraint on clock uncertainty of 200 ps. By reducing this value, the design might reach the timing closure but the margin related to clock jitter will decrease too, and this can create problems later on.

Optimization 3

The last optimization consists of using only one standard cells library for the complete implementation flow with the following characteristics:

- gate length: 30 nm
- threshold voltage: ultra-low (ULVT)
- track: 12-track

This library is equivalent to the second library presented in optimization 2 but this time it is used from the beginning. This allows more freedom for the implementation to optimize final critical paths not only by exchanging standard cells but also by rearranging cells and routing wires differently to reach the timing closure.

Indeed, using optimization 3, the design finally presents the timing closure. The critical path is still the same which involves the 32-bit multiplier. At the end of the implementation, the worst slack is positive and equal to 43 ps.

Clock [MHz]	333	666	1000
Standard cells library	35 nm, SVT, 9-track	35 nm, SVT, 9-track	30 nm, ULVT, 12-track
Instances	12152	17465	18317
Area [µm^2]	12231	18549	26619
Density [%]	51.61	56.70	65.39
Total power consumption [mW] (at 300 MHz and 30% input activity)	0.81	2.91	12.81

Table 5.5: Summary table of standard cell library, total instances, area, density and total power consumption after PnR for Trv32p3 microprocessor implemented at 333 MHz, 666 MHz and 1 GHz of clock frequency.

On the other hand, higher clock frequency impacts both area and power consumption. The final area of the Trv32p3 microprocessor implemented at 1 GHz is 26619 µm^2 with an occupation density of 65.39 % and counts 18317 total instances.

The area occupation increases by 118 % compared to the implementation at 333 MHz and by 44 % compared to the implementation at 666 MHz. Similarly, the total number of instances increases by 51 % and 5 % respectively compared to the implementation at 333 MHz and 666 MHz.
Moreover, total power consumption rises abruptly. At an input activity of 30 % and a dominant frequency of 300 MHz, total power consumption is 12.81 mW. This implementation consumes 1481 % and 340 % more power respectively compared to the design implemented at 333 MHz and 666 MHz. All these information are well summarized in table 5.5.

5.4 Final considerations

Several implementations have been presented in this chapter with related optimizations. The choice of an implementation strictly depends on the application. In this case study, these microprocessors need to be small enough to be contained in a Super-Pixel (SP) of a pixel detector. Moreover, they should be fast enough to perform data processing of pixels contained in a SP. By placing a Processing Element (PE) on a larger SP, the required clock frequency increases. At the same time, by placing PEs on larger SPs, the number of PEs on the pixel array decreases. Moreover, power consumption locally increases by increasing clock frequency but it overall decreases by decreasing the total number of PEs.
For instance, neglecting Network-on-Chip routers and considering pixel size of 55×55 μm^2, Trv32p3 microprocessor with a clock frequency of 333 MHz requires SPs with at least five pixels each. At the same clock frequency, the Tmicro microprocessor without trace buffer requires SPs with at least three pixels. To halve the time needed for data processing, by using Trv32p3 at a clock frequency of 666 MHz, the minimum number of pixels per SP increases up to eight. This reduces the maximum number of PEs on the pixel array but increases the power consumption for each of them. Furthermore, to perform data processing three times faster using 1 GHz of clock frequency, the Trv32p3 microprocessor should be placed at least every nine pixels. Nevertheless, this design presents extremely high power consumption. Finally, a good trade-off must be found between frequency, area, and power consumption of a PE. These figures of merit strongly depend on the size of the pixel array.

Chapter 6

Conclusions

This work presents a novel approach based on an embedded processor for data processing and readout for HEP applications. Two different proposals are presented: the first one consists of integrating an embedded processor at the periphery of the chip while the second one consists of placing Processing Elements directly on the pixel array. The latter one involves the division of the pixel array into Super-Pixels and the placement of Processing Elements on each Super-Pixel to perform a preliminary programmable data processing. This requires an ad-hoc communication architecture to allow the interaction among neighboring Processing Elements which is fulfilled by a Network-on-Chip. Each node of the Network-on-Chip is placed on each Super-Pixel to connect the corresponding Processing Element to the general network for information sharing. Network-on-Chip and Processing Elements have been analyzed separately.

The considered Network-on-Chip is characterized by a mesh topology and a wormhole packet switching technique. Several configurations have been analyzed by varying the Super-Pixel dimension and the operating frequency. Results show that latency always grows when the packet size increases or the available buswidth decreases. Moreover, latency explodes at high levels of particle rate and the Network-on-Chip starts to fail. The maximum sustainable particle rate improves by reducing the Super-Pixel dimension and it increases proportionally with the operating frequency.

Given the limited power, area, and latency budget available for High Energy Physics applications, an Application Specific Instruction-set Processor (ASIP) has been chosen as Processing Element since it ensures hardware efficiency with good software programmability. This work has been done for the first time in the Micro-Electronics group at CERN. Due to the high-level language complexity of ASIPs, two microprocessor examples have been chosen as a starting point characterized

by 16-bit non-RISC-V and 32-bit RISC-V instruction sets. Both microprocessors have been customized with an AMBA APB system bus protocol interface toward the external world for possible future integration in a SoC environment. Moreover, additional load and store instructions have been added to their instruction sets. An application-specific test code performing filtering and clustering of particles verified their correct functionality with real physics data as input. Results are strongly dependent on the application and input data set. Considering an input set of three events, the 16-bit non-RISC-V microprocessor requires 4284 cycles while the 32-bit RISC-V one requires 2560 cycles. The number of cycles is strongly dominated by read and write accesses to the central register file and Data Memory. Approximately 70 % of the instructions are used in the processor model which allows up to 30 % of instruction set reduction. Nevertheless, in the same conditions, the former microprocessor occupies 6147 µm^2 and consumes 0.50 mW while the latter one occupies 12231 µm^2 and consumes 0.81 mW when both are implemented with a clock frequency of 333 MHz in a 28 nm CMOS technology. For the 32-bit RISC-V core, higher performance can be achieved by doubling the clock frequency at the expense of a larger area ($\times$1.5) and higher power consumption ($\times$3.5). To achieve an operating frequency of 1 GHz, high-performance standard cells are required to reach the timing closure of the design. Despite the high frequency, the area only doubles, but the power consumption increases by more than 10 times compared to an increase of $\times$3 of the frequency.

Further improvements are necessary to better specify the Network-on-Chip and the processor model to be fully compliant with the application. Novel instructions and functional units are required to decrease the cycle count, especially for register and memory accesses. Moreover, the removal of unused instructions, operations, and functional units might further optimize the physical implementation of the ASIP.

Appendix A

Network on Chip analysis on MATLAB

```
bd = 64; % Bandwidth delay, number of columns
pr = 200; % Particle Rate, number of rows
bandwidth_delay = linspace(1,bd,bd); % packet_size(S)/buswidth(W)
particle_rates = logspace(0,11,pr); % [cm^{-2}*s^{-1}]

%% first case for 1x1 & 320 MHz
m = 1;
op_freq = 320;
L_sd = Latency_Function(m,op_freq);

%% Plot settings
A = axes;
mesh(bandwidth_delay,particle_rates,L_sd)
set(A,'YScale','log');
zlim([8 200]);
title("Latency VS Bandwidth delay & Particle rate");
xlabel('Bandwidth delay [cycle]');
ylabel('Particle rate [cm^{-2}*s^{-1}]');
zlabel('Latency [cycle]');

hold on
%% second case for 1x1 & 80 MHz
m = 1;
freq = 80;
L_sd = Latency_Function(m,freq);
mesh(bandwidth_delay,particle_rates,L_sd)

%% third case for 4x4 & 320 MHz
```

```
m = 4;
freq = 320;
L_sd = Latency_Function(m,freq);
mesh(bandwidth_delay,particle_rates,L_sd)

%% fourth case for 4x4 & 80 MHz
m = 4;
freq = 80;
L_sd = Latency_Function(m,freq);
mesh(bandwidth_delay,particle_rates,L_sd)

%% fifth case for 32x32 & 320 MHz
m = 32;
freq = 320;
L_sd = Latency_Function(m,freq);
mesh(bandwidth_delay,particle_rates,L_sd)

%% sixth case for 32x32 & 80 MHz
m = 32;
freq = 80;
L_sd = Latency_Function(m,freq);
mesh(bandwidth_delay,particle_rates,L_sd)

%% seventh case for 256x256 & 320 MHz
m = 256;
freq = 320;
L_sd = Latency_Function(m,freq);
mesh(bandwidth_delay,particle_rates,L_sd)

%% eighth case for 256x256 & 80 MHz
m = 256;
freq = 80;
L_sd = Latency_Function(m,freq);
mesh(bandwidth_delay,particle_rates,L_sd)
hold off

%% FUNCTION
function Latency_Output = Latency_Function(general_m,general_op_freq)
bd = 64; % Bandwidth delay, number of columns
pr = 200; % Particle Rate, number of rows
bandwidth_delay = linspace(1,bd,bd); % packet_size(S)/buswidth(W)
particle_rates = logspace(0,11,pr); % [cm^{-2}*s^{-1}]
n = 256; %pixel matrix
pixel_matrix = n*n;
pixel_dim = 55*55; % Pixel dimension, 1 um^2 = 1e-8 cm^2

m = general_m; %NoC matrix - this can be 1, 4, 32, 256 or any other
    value
NoC_dim = m*m;
```

```
77
78 op_freq = general_op_freq*1e6; %operating frequency, in this case 320
       or 80 MHz
79 Latency_Output = ones(pr,bd); %initialization of Latency matrix
80
81 Hs = 2; %Router service time for the header flit [cycle]
82 C = [1 2/9 2/9 2/9; %contention matrix
83       2/9 1 2/9 2/9;
84       2/9 2/9 1 2/9;
85       2/9 2/9 2/9 1];
86 for j = 1:pr %filling of rows
87     particle_rate = particle_rates(j);
88     for i = 1:bd %filling of columns
89                                     % S_W means S/W
90         S_W = bandwidth_delay(i); %[cycle] sweep over bandwidth delay
91                                     %so both S and W might vary
92         T = Hs + S_W; %service time of the packet
93         Ws = 1; %queuing delay the packet experiences at the source
       node s
94         x_sd = (particle_rate*(pixel_dim*1e-8)*NoC_dim)/(op_freq); %
       [packet/cycle]
95         %"pixel matrix" at numerator and denominator get simplified
96
97         lambda_i_j = x_sd;
98          lambda_j = 3*lambda_i_j; %Since a PE requests info from
       adjacent super-pixels and packets move only by maximum 2 positions
99         LAMBDA = [lambda_j 0 0 0; %traffic arrival rate and for a
       symmetric and isotropic case all the diagonal values are equal
100                  0 lambda_j 0 0;
101                  0 0 lambda_j 0;
102                  0 0 0 lambda_j];
103        I = eye(4); %identity matrix
104         R_bar = [1; %residual time and R = 1
105                  1;
106                  1;
107                  1];
108        N = (inv(I - T*LAMBDA*C))*LAMBDA*R_bar; %average number of
       packets at each input buffer of the router
109        W_i_j = N/lambda_i_j; %average waiting time for each channel
       buffer of each router
110        lat = Ws + W_i_j + T; %Latency
111        if (lat(1) < 4) %To stop when it starts to diverge
112            break; % (no actual value will be lower or equal to 4
       becuase they all start from higher values)
113        else
114        Latency_Output(j,i) = lat(1); %Since all the lat values are
       the same, just take one for plotting
115        end
116
```

```
117         end
118 end
119 end
```

Bibliography

[1] CERN. *CERN website*. URL: https://home.cern/ (cit. on p. 1).

[2] CERN. *ATLAS Experiment*. URL: www.atlas.ch (cit. on p. 2).

[3] CERN. *CMS Experiment*. URL: https://cms.cern/ (cit. on p. 2).

[4] CERN. *LHCb Experiment*. URL: http://lhcb.web.cern.ch/ (cit. on p. 2).

[5] CERN. *ALICE Experiment*. URL: https://alice-collaboration.web.cern.ch/ (cit. on p. 2).

[6] Philippe Mouche. «Overall view of the LHC. Vue d'ensemble du LHC». In: (2014). General Photo. URL: https://cds.cern.ch/record/1708847 (cit. on p. 2).

[7] Stefan Ulmer. *BASE Annual Report 2018*. Tech. rep. Geneva: CERN, 2019. URL: https://cds.cern.ch/record/2654098 (cit. on p. 3).

[8] Oliver Sim Brüning, Paul Collier, P Lebrun, Stephen Myers, Ranko Ostojic, John Poole, and Paul Proudlock. *LHC Design Report*. CERN Yellow Reports: Monographs. Geneva: CERN, 2004. DOI: 10.5170/CERN-2004-003-V-1. URL: https://cds.cern.ch/record/782076 (cit. on p. 3).

[9] Toshinori Mori. «Searches for standard model Higgs boson at LEP». In: *AIP Conference Proceedings* 272.2 (1992), pp. 1321–1325. DOI: 10.1063/1.43422. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.43422. URL: https://aip.scitation.org/doi/abs/10.1063/1.43422 (cit. on p. 3).

[10] G Apollinari, I Béjar Alonso, O Brüning, M Lamont, and L Rossi. *High-Luminosity Large Hadron Collider (HL-LHC): Preliminary Design Report*. CERN Yellow Reports: Monographs. Geneva: CERN, 2015. DOI: 10.5170/CERN-2015-005. URL: https://cds.cern.ch/record/2116337 (cit. on pp. 3, 15).

[11] Martin Aleksa et al. *Strategic R&D Programme on Technologies for Future Experiments*. Tech. rep. Geneva: CERN, 2018. URL: https://cds.cern.ch/record/2649646 (cit. on p. 4).

[12] Lucie Linssen, Akiya Miyamoto, Marcel Stanitzki, and Harry Weerts. *Physics and Detectors at CLIC: CLIC Conceptual Design Report*. 2012. DOI: 10.48550/ARXIV.1202.5940. URL: https://arxiv.org/abs/1202.5940 (cit. on p. 4).

[13] A Abada et al. «FCC Physics Opportunities: Future Circular Collider Conceptual Design Report Volume 1». In: *European Physical Journal C* 79 (2019) (cit. on p. 4).

[14] M. Tanabashi et al. «Review of Particle Physics». In: *Phys. Rev. D* 98 (3 2018), p. 030001. DOI: 10.1103/PhysRevD.98.030001. URL: https://link.aps.org/doi/10.1103/PhysRevD.98.030001 (cit. on p. 6).

[15] T Akesson et al. «Particle identification using the time-over-threshold method in the ATLAS Transition Radiation Tracker». In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 474.2 (2001), pp. 172–187. ISSN: 0168-9002. DOI: https://doi.org/10.1016/S0168-9002(01)00878-6. URL: https://www.sciencedirect.com/science/article/pii/S0168900201008786 (cit. on pp. 8, 21).

[16] Davide Ceresa, Gianmario Bergamin, Alessandro Caratelli, Jan Kaplon, Kostas Kloukinas, Simone Scarfì, and Yusuf Leblebici. «MPA-SSA, design and test of a 65nm ASIC-based system for particle tracking at HL-LHC featuring on-chip particle discrimination». In: *2019 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*. 2019, pp. 1–3. DOI: 10.1109/NSS/MIC42101.2019.9059989 (cit. on p. 8).

[17] Aleksandra Dimitrievska and Andreas Stiller. «RD53A: A large-scale prototype chip for the phase II upgrade in the serially powered HL-LHC pixel detectors». In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 958 (2020). Proceedings of the Vienna Conference on Instrumentation 2019, p. 162091. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2019.04.045. URL: https://www.sciencedirect.com/science/article/pii/S0168900219305029 (cit. on p. 9).

[18] X. Llopart et al. «Timepix4, a large area pixel detector readout chip which can be tiled on 4 sides providing sub-200 ps timestamp binning». In: *Journal of Instrumentation* 17.01 (2022), p. C01044. DOI: 10.1088/1748-0221/17/01/c01044. URL: https://doi.org/10.1088/1748-0221/17/01/c01044 (cit. on pp. 9, 21).

[19] Isiaka A. Alimi, Romil K. Patel, Oluyomi Aboderin, Abdelgader M. Abdalla, Ramoni A. Gbadamosi, Nelson J. Muga, Armando N. Pinto, and António L. Teixeira. «Network-on-Chip Topologies: Potentials, Technical Challenges, Recent Advances and Research Direction». In: *Network-on-Chip*. Ed. by Isiaka A. Alimi, Oluyomi Aboderin, Nelson J. Muga, and António L. Teixeira. Rijeka: IntechOpen, 2021. Chap. 3. DOI: 10.5772/intechopen.97262. URL: https://doi.org/10.5772/intechopen.97262 (cit. on pp. 15, 16).

[20] Hyung Gyu Lee, Naehyuck Chang, Umit Y. Ogras, and Radu Marculescu. «On-Chip Communication Architecture Exploration: A Quantitative Evaluation of Point-to-Point, Bus, and Network-on-Chip Approaches». In: *ACM Trans. Des. Autom. Electron. Syst.* 12.3 (2008). ISSN: 1084-4309. DOI: 10.1145/1255456.1255460. URL: https://doi.org/10.1145/1255456.1255460 (cit. on p. 16).

[21] Umit Y. Ogras and Radu Marculescu. *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*. Springer Publishing Company, Incorporated, 2013. ISBN: 9400739575 (cit. on pp. 16, 18–23, 25, 26).

[22] José Duato, Sudhakar Yalamanchili, and Lionel Ni. «CHAPTER 1 - Introduction». In: *Interconnection Networks*. Ed. by José Duato, Sudhakar Yalamanchili, and Lionel Ni. The Morgan Kaufmann Series in Computer Architecture and Design. San Francisco: Morgan Kaufmann, 2003, pp. 1–41. DOI: https://doi.org/10.1016/B978-155860852-8/50004-3. URL: https://www.sciencedirect.com/science/article/pii/B9781558608528500043 (cit. on p. 16).

[23] Benoit De Lescure. «Why network-on-chip has displaced crossbar switches at scale». In: (2021). URL: https://www.edn.com/why-network-on-chip-has-displaced-crossbar-switches-at-scale/ (cit. on p. 16).

[24] Tuomas Sakari Poikela. «Readout Architecture for Hybrid Pixel Readout Chips». Presented 15 Jun 2015. 2015. URL: https://cds.cern.ch/record/2042198 (cit. on p. 17).

[25] Zhonghai Lu. «Using wormhole switching for networks on chip : feasibility analysis and microarchitecture adaptation». In: 2005 (cit. on p. 18).

[26] T. Poikela et al. «The VeloPix ASIC». In: *Journal of Instrumentation* 12 (Jan. 2017), pp. C01070–C01070. DOI: 10.1088/1748-0221/12/01/C01070 (cit. on p. 21).

[27] M De Gaspari et al. «Design of the analog front-end for the Timepix3 and Smallpix hybrid pixel detectors in 130 nm CMOS technology». In: *Journal of Instrumentation* 9.01 (2014), pp. C01037–C01037. DOI: 10.1088/1748-0221/9/01/c01037. URL: https://doi.org/10.1088/1748-0221/9/01/c01037 (cit. on p. 21).

[28] M Raymond, D Braga, W Ferguson, J Fulcher, G Hall, J Jacob, L Jones, M Pesaresi, and M Prydderch. «The CMS binary chip for microstrip tracker readout at the SLHC». In: *Journal of Instrumentation* 7.01 (2012), pp. C01033–C01033. DOI: 10.1088/1748-0221/7/01/c01033. URL: https://doi.org/10.1088/1748-0221/7/01/c01033 (cit. on p. 21).

[29] X. Llopart, R. Ballabriga, M. Campbell, L. Tlustos, and W. Wong. «Timepix, a 65k programmable pixel readout chip for arrival time, energy and/or photon counting measurements». In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 581.1 (2007). VCI 2007, pp. 485–494. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2007.08.079. URL: https://www.sciencedirect.com/science/article/pii/S0168900207017020 (cit. on p. 21).

[30] W.S. Wong et al. «Introducing Timepix2, a frame-based pixel detector readout ASIC measuring energy deposition and arrival time». In: *Radiation Measurements* 131 (2020), p. 106230. ISSN: 1350-4487. DOI: https://doi.org/10.1016/j.radmeas.2019.106230. URL: https://www.sciencedirect.com/science/article/pii/S1350448719305165 (cit. on p. 21).

[31] T Poikela et al. «Timepix3: a 65K channel hybrid pixel readout chip with simultaneous ToA/ToT and sparse readout». In: *Journal of Instrumentation* 9.05 (2014), pp. C05013–C05013. DOI: 10.1088/1748-0221/9/05/c05013. URL: https://doi.org/10.1088/1748-0221/9/05/c05013 (cit. on pp. 21, 24, 60).

[32] J Varela. *Timing and synchronization in the LHC experiments*. Tech. rep. Geneva: CERN, 2000. DOI: 10.5170/CERN-2000-010.77. URL: https://cds.cern.ch/record/478248 (cit. on p. 25).

[33] John Little. «OR FORUM - Little's Law as Viewed on Its 50th Anniversary.» In: *Operations Research* 59 (June 2011), pp. 536–549. DOI: 10.2307/23013126 (cit. on p. 25).

[34] Wikipedia: The Free Encyclopedia. *Data processing*. [Online; accessed 10-August-2022]. URL: https://en.wikipedia.org/wiki/Data_processing (cit. on p. 33).

[35] Wikipedia: The Free Encyclopedia. *Microprocessor*. [Online; accessed 10-August-2022]. URL: https://en.wikipedia.org/wiki/Microprocessor (cit. on p. 33).

[36] Synopsys, Inc. *ASIP Designer - Application-Specific Processor Design Made Easy*. URL: https://www.synopsys.com/dw/doc.php/ds/cc/asip-brochure.pdf (cit. on pp. 35, 38).

[37] Wikipedia: The Free Encyclopedia. *Application-specific instruction set processor*. [Online; accessed 27-July-2022]. URL: https://en.wikipedia.org/wiki/Application-specific_instruction_set_processor (cit. on p. 36).

[38] Dinkar Sitaram and Geetha Manjunath. «Chapter 5 - Paradigms for Developing Cloud Applications». In: *Moving To The Cloud*. Ed. by Dinkar Sitaram and Geetha Manjunath. Boston: Syngress, 2012, pp. 205–253. ISBN: 978-1-59749-725-1. DOI: https://doi.org/10.1016/B978-1-59749-725-1.00005-6. URL: https://www.sciencedirect.com/science/article/pii/B9781597497251000056 (cit. on p. 37).

[39] Wikipedia: The Free Encyclopedia. *Instruction-level parallelism*. [Online; accessed 27-July-2022]. URL: https://en.wikipedia.org/wiki/Instruction-level_parallelism (cit. on p. 37).

[40] Inc. Synopsys. *nML Manual - ASIP Designer*. English. Version S-2021.12. Synopsys, Inc. 2021 (cit. on p. 38).

[41] Inc. Synopsys. *PDG Manual - ASIP Designer*. English. Version S-2021.12. Synopsys, Inc. 2021 (cit. on pp. 41, 44).

[42] Inc. Synopsys. *Trv (RISC-V ISA) Models - ASIP Designer*. English. Version S-2021.12. Synopsys, Inc. 2021 (cit. on p. 42).

[43] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1*. Tech. rep. UCB/EECS-2016-118. EECS Department, University of California, Berkeley, 2016. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html (cit. on p. 42).

[44] Inc. Synopsys. *Tmicro Core - ASIP Designer*. English. Version S-2021.12. Synopsys, Inc. 2021 (cit. on p. 42).

[45] Arm. *AMBA®APB Protocol Specification*. English. Version 2.2. Arm. 2021 (cit. on pp. 44, 46).

[46] Inc. Synopsys. *Go HDL Generator Manual - ASIP Designer*. English. Version S-2021.12. Synopsys, Inc. 2021 (cit. on pp. 50, 56).

[47] MEDUNA Lukáš. «Detecting elementary particles with Timepix3 detector». Diplomová práce. Praha: Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra softwaru a výuky informatiky, 2019. URL: https://dspace.cuni.cz/bitstream/handle/20.500.11956/106938/120331567.pdf?sequence=1&isAllowed=y (cit. on p. 60).

[48] MÁNEK Petr. «A system for 3D localization of gamma sources using Timepix3-based Compton cameras». Diplomová práce. Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra softwarového inženýrství, 2018. URL: https://dspace.cuni.cz/bitstream/handle/20.500.11956/101404/120308522.pdf?sequence=1&isAllowed=y (cit. on p. 60).

[49] Benedikt Bergmann and Till Rehm. *Detector measures cosmic radiation on the Zugspitze*. 2021. URL: https://www.dlr.de/content/en/articles/news/2021/02/20210616_detector-measures-cosmic-radiation-on-the-zugspitze.html (visited on 08/15/2022) (cit. on p. 60).

[50] F. Zimmermann. «LHC/FCC-based muon colliders». In: *Journal of Physics: Conference Series* 1067 (2018), p. 022017. DOI: 10.1088/1742-6596/1067/2/022017. URL: https://doi.org/10.1088/1742-6596/1067/2/022017 (cit. on p. 61).

[51] IBM. *Program Profiling*. [Online; accessed 17-August-2022]. URL: https://www.ibm.com/docs/en/i/7.2?topic=techniques-program-profiling (cit. on p. 68).

[52] Inc. Synopsys. *Checkers Simulator Manual - ASIP Designer*. English. Version S-2021.12. Synopsys, Inc. 2021 (cit. on p. 68).