



# A plug-and-play real-time architecture for MPSoC-FPGAs targeting interventional Computed Tomography

Dissertation

zur Erlangung des akademischen Grades

**Doktoringenieur**

**(Dr.-Ing.)**

von M. Sc. Daniele Passaretti

geb. am 23. Mai 1992 in Gaeta (Italien)

genehmigt durch die Fakultät für Elektrotechnik und Informa-  
tionstechnik der Otto-von-Guericke Universität Magdeburg

Gutachter:

Prof. Dr.-Ing. Thilo Pionteck

Prof. Dr. rer. nat. Nicola D'Ascenzo

Promotionskolloquium am 10. September 2024





## Abstract

In recent years, new image-guided interventional procedures, such as interventional Computed Tomography (iCT), have been explored to tackle the increasing number of tumors. During these interventional procedures, surgeons track the needle used for tumor ablation within the patient's body with the support of Computed Tomography (CT) or multi-modal CT/Positron Emission Tomography (PET)/Magnetic Resonance Imaging (MRI) techniques. While CT scanners for tumor diagnosis have been typically designed as custom closed systems, new scanners for interventional procedures aim to be adaptable and configurable for multimodality scanning, providing real-time images. Therefore, from the system designer perspective, these scanners are considered Cyber-Physical System (CPS) devices, where components must be controlled, and data must be acquired and processed in real time.

The “**K**onfigurierbarer, **I**nterfaceoffener, **D**osissparender **C**omputertomograph” (KIDS-CT) scanner is the first open-interface CT scanner assembled by Academia aiming to provide new features: extension for adding new components in a plug-and-play fashion, and user-accessible sensor/actuator parameters (e.g., individual settings of detector, X-ray tube voltage, Time-of-flight cameras). These features allow researchers and/or physicians to explore new multimodality techniques and interventional procedures with the aim of optimizing the X-ray dose and enhancing reconstruction algorithms.

This thesis addresses the problem of real-time data acquisition and processing in CPSs and their extension capabilities for adding components in a plug-and-play fashion. It focuses on the KIDS-CT scanner, for which multimodality functionalities and real-time support must be provided in order to conduct and explore iCT procedures. To address these problems at the system and hardware design level, this research work firstly contributes in the design process of the CT scanner, modeled as a CPS device, by proposing a System Architecture and the associated Communication Infrastructure; Secondly, it proposes a new Control-Data Acquisition System (CDAS) architecture for Multi-Processor System-on-Chip Field Programmable Gate-Array (MPSoC-FPGA) platforms. Although the proposed work has been implemented and validated targeting the KIDS-CT scanner, it is configurable for various CPS applications where data must be collected and processed on the fly, while components must be controlled in real time. In fact, the CDAS plays a crucial role in controlling CPS components at the device level, collecting and processing data in real time, and providing plug-and-play capability for the target application, such as the KIDS-CT scanner.

In order to reach these aims, various methodologies have been proposed: Real-time and non-real-time tasks are properly mapped between the Programmable Logic and the Processing System of the MPSoC-FPGA; The Communication Infrastructure has been modeled in layers and classes that contain different protocols on the base of the task

type; A dataflow-module and a data-processing module have been proposed to collect, and preprocess data on the fly, without using external memory. In addition, data can be pre-processed in different formats, making it suitable for exploring the design space by tuning data formats to determine the most appropriate design configuration to pre-process data for interventional procedures.

As part of this work, a guideline for designing an open-interface CT scanner has been provided at the system level and digital signal processing level. The proposed communication protocols for the plug-and-play capability and the real-time support have been described. Furthermore, the hardware/software CDAS architecture has been described through its three main components: the Control-synchronization Module, the Data-flow Module, and the Data-processing Module. Moreover, a novel hardware isolation method to enable isolation support on MPSoC-FPGAs has been proposed; the proposed isolation method solves the problem of isolation between hardware modules inside the MPSoC-FPGA.

Finally, this work describes the realization of the System Architecture, the Communication Infrastructure and the CDAS architecture in the specific case of the KIDS-CT scanner. For this purpose, the XC7Z045 MPSoC-FPGA has been used for implementing the CDAS architecture. Here, the proposed optimizations have permitted to achieve an efficient solution, which use only 7.81% of Look-Up Tables, 5.82% of Flip-Flops, 5% of Digital Signal Processors, and 7.89% of Block RAMs, and collect and process pixel data in an estimated time of 467.8 *ns*. Since pixel data are processed on the fly during the acquisition of each projection, and this processing is faster than the “integration period” required to acquire a projection, the proposed pre-processing solution adds zero latency to the acquisition time. Therefore, the Graphics Processing Unit (GPU) on the reconstruction system only needs to perform the rest of the processing, resulting the entire acquisition and reconstruction time much faster than before. Such a solution could not be achieved with the standard approach, as it would exceed the capacity of the available Digital Signal Processors in the selected MPSoC-FPGA. Furthermore, the proposed optimized solution is 6.4 times faster than the standard approach. In conclusion, this thesis answers to the problem of how to provide real-time support and plug-and-play capability within complex CPSs such as the KIDS-CT scanner, and enables this scanner to explore new multimodality techniques and interventional procedures.

## Zusammenfassung

Um die zunehmende Zahl von Tumorerkrankungen zu bekämpfen, wurden in den letzten Jahren neue bildgesteuerte interventionelle Methoden erforscht, wie z. B. das Interventional Computed Tomography (iCT)-Verfahren. Bei diesen interventionellen Verfahren verfolgen die Chirurgen die bei der Tumorentfernung verwendete Nadel im Körper des Patienten mit Computed Tomography (CT) oder multimodalen CT/Positron Emission Tomography (PET)/Magnetic Resonance Imaging (MRI)-Techniken. Während diagnostische CT-Scanner in der Regel als zugeschnittene, geschlossene Systeme konzipiert werden, sollen neue Scanner für interventionelle Verfahren anpassungsfähig und für multimodales Scannen konfigurierbar sein sowie Echtzeitbilder liefern. Daher funktionieren diese Scanner auf der Ebene des Systemdesigns als Cyber-Physical System (CPS)-Geräte, bei denen die Komponenten gesteuert und die Daten in Echtzeit erfasst und verarbeitet werden müssen.

Der “Konfigurierbare, Interfaceoffene, Dosissparende Computertomograph“ (KIDS-CT)-Scanner ist der erste von der akademischen Welt fertiggestellte CT-Scanner mit offener Schnittstelle, der neue Funktionen bietet: Erweiterungen für das Hinzufügen neuer Komponenten im Plug-and-Play-Verfahren und für den Benutzer zugängliche Sensor/Aktor-Parameter (z. B. individuelle Einstellungen des Detektors, der Spannung der Röntgenröhre und der Time-of-Flight-Kameras). Diese Funktionen ermöglichen es Forschenden und/oder Ärzten, neue multimodale Techniken und interventionelle Verfahren mit dem Ziel zu erforschen, die Röntgendosis zu optimieren und die Rekonstruktionsalgorithmen zu verbessern.

Diese Arbeit befasst sich mit der Problematik der Echtzeit-Datenerfassung und -Verarbeitung innerhalb von CPSs und deren Erweiterungsmöglichkeiten für das Hinzufügen von Komponenten in Plug-and-Play-Weise. Der Fokus liegt auf dem KIDS-CT-Scanner, für den multimodale Funktionalitäten und Echtzeitunterstützung bereitgestellt werden müssen, um iCT-Verfahren durchzuführen und zu erforschen. Um diese Probleme auf der System- und Hardware-Entwurfsebene anzugehen, trägt diese Forschungsarbeit erstens zum Entwurfsprozess des als CPS-Gerät modellierten CT-Scanners bei, indem sie eine Systemarchitektur und die zugehörige Kommunikationsinfrastruktur vorschlägt; zweitens schlägt sie eine neue Control-Data Acquisition System (CDAS)-Architektur für Multi-Processor System-on-Chip Field Programmable Gate-Array (MPSoC-FPGA)-Plattformen vor. Obwohl die Arbeit für den KIDS-CT-Scanner implementiert und validiert wurde, ist sie für verschiedene CPS-Anwendungen konfigurierbar, die unter Verwendung der entwickelten Systemarchitektur entwickelt werden können. Tatsächlich spielt das CDAS eine entscheidende Rolle bei der Steuerung von CPS-Komponenten auf Geräteebene, bei der Erfassung und Verarbeitung von Daten in Echtzeit und bei der Bereitstellung von Plug-and-Play-Fähigkeiten für die Zielanwendung, wie z. B. den KIDS-CT-Scanner.

Um diese Ziele zu erreichen, werden verschiedene Methoden angewandt: Echtzeit- und Nicht-Echtzeit-Aufgaben werden auf geeignete Art und Weise zwischen der programmier-

baren Logik und dem Verarbeitungssystem des MPSoC-FPGA abgebildet; die Kommunikationsinfrastruktur wird in Ebenen und Klassen modelliert, die verschiedene Protokolle auf der Grundlage des Aufgabentyps enthalten; ein Datenflussmodul und ein Datenverarbeitungsmodul werden implementiert, um Daten zu sammeln und vorzubehandeln, ohne externen Speicher zu verwenden. Darüber hinaus kann das Datenverarbeitungsmodul Daten in verschiedenen Formaten vorverarbeiten.

Diese Flexibilität ermöglicht die Erkundung des Designraums durch Anpassung der Datenformate. Eine solche Abstimmung hilft bei der Bestimmung der am besten geeigneten Designkonfiguration für die Vorverarbeitung von Daten bei interventionellen Verfahren.

Im Rahmen dieser Arbeit wird ferner ein Leitfaden für die Entwicklung eines CT-Scanners mit offener Schnittstelle auf Systemebene und auf Ebene der digitalen Signalverarbeitung bereitgestellt. Die integrierten Kommunikationsprotokolle für die Plug-and-Play Fähigkeit und die Echtzeitunterstützung werden beschrieben. Darüber hinaus wird die Software-/Hardware-Architektur von CDAS mit ihren drei Hauptkomponenten beschrieben: das Steuerungs-/Synchronisationsmodul, das Datenflussmodul und das Datenverarbeitungsmodul. Darüber hinaus wurde eine neuartige Hardware-Isolationsmethode entwickelt, um die Isolationsunterstützung auf MPSoC-FPGAs zu ermöglichen; die dazugehörige Isolationsmethode löst das Problem der Isolation zwischen Hardware-Modulen innerhalb des MPSoC-FPGA.

Insgesamt beschreibt diese Arbeit die Realisierung der Systemarchitektur, der Kommunikationsinfrastruktur und der CDAS-Architektur im speziellen Fall des KIDS-CT-Scanners. Zu diesem Zweck wurde der XC7Z045 MPSoC-FPGA für die Implementierung der CDAS-Architektur verwendet. Mit den hier implementierten Optimierungen wurde eine effiziente Lösung erreicht, die nur 7,81% der Look-Up Tables, 5,82% der Flip-Flops, 5% der Digitalen Signalprozessoren und 7,89% der Block RAMs verwendet und Pixeldaten in einer geschätzten Zeit von 467,8 ns erfasst und verarbeitet. Da die Pixeldaten während der Erfassung jeder Projektion “on-the-fly” verarbeitet werden und diese Verarbeitung schneller ist als die für die Erfassung einer Projektion erforderliche Integrationszeit, fügt die vorgestellte Vorverarbeitungslösung der Erfassungszeit keine Latenz hinzu. Daher muss die Graphics Processing Unit (GPU) auf dem Rekonstruktionssystem nur den Rest der Verarbeitung durchführen, wodurch die gesamte Erfassungs- und Rekonstruktionszeit deutlich schneller ist als bei bisherigen Lösungen. Eine solche Lösung könnte mit dem Standardansatz nicht erreicht werden, da sie die Kapazität des verfügbaren Digitalen Signalprozessoren in dem ausgewählten MPSoC-FPGA übersteigen würde. Außerdem ist die vorgeschlagene optimierte Lösung 6,4-Mal schneller als der Standardansatz. Zusammenfassend wird die Frage beantwortet, wie Echtzeit-Unterstützung und Plug-and-Play-Fähigkeit innerhalb komplexer CPSs, wie dem KIDS-CT-Scanner, bereitgestellt und wie damit neue multimodale Techniken und interventionelle Verfahren erforscht werden können.

# Publications

## Publication As First Author

### Original works in peer-reviewed international journals

- [DP 1] **D. Passaretti**, M. Ghosh, S. Abdurahman, M. L. Egito, and T. Pionteck. “Hardware Optimizations of the X-ray Pre-Processing for Interventional Computed Tomography Using the FPGA”. In: *Applied Sciences* 12.11 (2022). ISSN: 2076-3417. DOI: 10.3390/app12115659. URL: <https://www.mdpi.com/2076-3417/12/11/5659>.
- [DP 2] **D. Passaretti**, M. Steiger, and T. Pionteck. “Enabling Plug-and-Play in Cyber-Physical Systems Using MPSoC-FPGAs”. In: *IEEE Access* 11 (2023), pp. 116219–116234. DOI: 10.1109/ACCESS.2023.3325742.

### Original works in peer-reviewed international conferences

- [DP 3] **D. Passaretti**, J. M. Joseph, and T. Pionteck. “Survey on FPGAs in Medical Radiology Applications: Challenges, Architectures and Programming Models”. In: *2019 International Conference on Field-Programmable Technology (ICFPT)*. 2019, pp. 279–282. DOI: 10.1109/ICFPT47387.2019.00047.
- [DP 4] **D. Passaretti** and T. Pionteck. “Hardware/Software Co-Design of a control and data acquisition system for Computed Tomography”. In: *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. 2020, pp. 1–4. DOI: 10.1109/MOCAST49295.2020.9200273.
- [DP 5] **D. Passaretti** and T. Pionteck. “Configurable Pipelined Datapath for Data Acquisition in Interventional Computed Tomography”. In: *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2021, pp. 257–257. DOI: 10.1109/FCCM51124.2021.00044.
- [DP 6] **D. Passaretti**, F. Boehm, M. Wilhelm, and T. Pionteck. “Hardware Isolation Support for Low-Cost SoC-FPGAs”. In: *Architecture of Computing Systems*. Ed. by M. Schulz, C. Trinitis, N. Papadopoulou, and T. Pionteck. Cham: Springer International Publishing, 2022, pp. 148–163. ISBN: 978-3-031-21867-5.

- [DP 7] **D. Passaretti** and T. Pionteck. “A Control Data Acquisition System Architecture for MPSoC-FPGAs in Computed Tomography”. In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Ed. by F. Palumbo, G. Keramidas, N. Voros, and P. C. Diniz. Cham: Springer Nature Switzerland, 2023, pp. 361–365. ISBN: 978-3-031-42921-7.

## Patent

- [DP 8] T. Hoffmann, **D. Passaretti**, R. Frysch, T. Pfeiffer, and G. Rose. *MEDICAL IMAGING SYSTEM AND COMPUTER PROGRAM*. US Patent US 2023/0377720 A1 (Pub. Date: Nov. 23, 2023), EP EP4225148A1 (Pub. Date: Aug. 16, 2023), JP2023544700A (Pub. Date: Oct. 25, 2023), WO2022073958A1 (Pub. Date: Apr. 14, 2022). Assignee: Otto-von-Guericke-Universität Magdeburg, Magdeburg (DE). URL: <https://patents.google.com/patent/US20230377720A1/en>.

## Publication With Authors In Alphabetic Order

### Original works in peer-reviewed international journals

- [DP 9] A. Cilaro, M. Gagliardi, and **D. Passaretti**. “Hardware Support for Thread Synchronisation in an Experimental Manycore System”. In: *Int. J. Grid Util. Comput.* 11.1 (2020), 62–71. ISSN: 1741-847X. DOI: 10.1504/ijguc.2020.103970. URL: <https://doi.org/10.1504/ijguc.2020.103970>.

### Original works in peer-reviewed international conferences

- [DP 10] A. Cilaro, M. Gagliardi, and **D. Passaretti**. “NoC-Based Thread Synchronization in a Custom Manycore System”. In: *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*. Ed. by F. Xhafa, S. Caballé, and L. Barolli. Cham: Springer International Publishing, 2018, pp. 673–682. ISBN: 978-3-319-69835-9.

# Contents

<b>I</b>	<b>Introduction</b>	<b>xi</b>
<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Motivation . . . . .	2
1.1.1	Why interventional and multimodality CT matter? . . . . .	3
1.1.2	Why KIDS-CT? . . . . .	3
1.1.3	Why a CT scanner for multimodality/interventional like KIDS-CT is a CPS? . . . . .	3
1.1.4	Why use MPSoC-FPGAs? . . . . .	3
1.2	Research Questions . . . . .	4
1.3	Research Contributions . . . . .	4
1.4	Thesis Outline . . . . .	5
<b>2</b>	<b>Technical Background</b>	<b>7</b>
2.1	Cyber-Physical Systems . . . . .	7
2.1.1	Physical environment . . . . .	8
2.1.2	Embedded system . . . . .	8
2.1.3	Physical architecture . . . . .	9
2.2	Multi-Processor System-on-Chip Field-Programmable Gate Array . . . . .	10
2.2.1	On-chip communication architecture . . . . .	11
2.3	Mixed-Criticality Systems . . . . .	13
2.3.1	Shared resources in MCSs . . . . .	14
2.3.2	CPSs and MCSs . . . . .	14
2.3.3	MPSoC-FPGAs for MCSs . . . . .	14
2.4	Interventional Computed Tomography . . . . .	15
2.4.1	CT scanner fundamentals . . . . .	15
2.4.2	KIDS-CT scanner . . . . .	16
2.4.3	CT reconstruction theory . . . . .	19
2.5	Design Space Exploration . . . . .	27
2.5.1	Data formats for number representation . . . . .	28
<b>3</b>	<b>Related Works</b>	<b>30</b>
3.1	System Architecture In CPSs . . . . .	30
3.2	Control And Data Acquisition Systems . . . . .	32

3.3	Task And Peripheral Isolation . . . . .	36
3.3.1	Isolation in AMD-Xilinx architectures . . . . .	36
3.3.2	Protection units solutions . . . . .	38
3.4	Computed Tomography . . . . .	41
3.4.1	Controlling and data acquisition systems for CT scanners . . . . .	41
3.4.2	The data format exploration in CT data processing . . . . .	43
 <b>II Concept</b>		 <b>46</b>
 <b>4 Problem Analysis</b>		 <b>48</b>
4.1	Weakness Of The Current Architectures . . . . .	48
4.2	Research Questions & Objectives . . . . .	50
 <b>5 Methodology</b>		 <b>52</b>
5.1	Requirement Definition For The Selected CPS Application . . . . .	52
5.2	System Architecture . . . . .	53
5.3	Communication Infrastructure . . . . .	53
5.4	Control-Data Acquisition System . . . . .	54
 <b>6 Cyber-Physical System Architecture</b>		 <b>55</b>
6.1	Requirement & Task Classification . . . . .	55
6.2	System Architecture . . . . .	56
6.3	Communication Infrastructure . . . . .	58
6.3.1	Communication interface layer . . . . .	60
6.3.2	Transport protocol layer . . . . .	61
6.3.3	Application protocol layer . . . . .	61
 <b>7 Control-Data Acquisition System</b>		 <b>66</b>
7.1	Task Partitioning . . . . .	66
7.2	Hardware/Software Architecture . . . . .	67
7.3	Control-Synchronization Module . . . . .	69
7.3.1	Hardware layer . . . . .	69
7.3.2	Application layer . . . . .	73
7.3.3	Example . . . . .	75
7.4	Data-Flow Module . . . . .	77
7.4.1	Architecture reconfigurability . . . . .	78
7.4.2	Inter-clock domains . . . . .	80
7.4.3	Architecture description . . . . .	81
7.5	Data-Processing Module . . . . .	84
7.5.1	Architecture description . . . . .	84



7.6	Isolation Support For MPSoC-FPGAs . . . . .	86
7.6.1	LPU architecture . . . . .	88
7.6.2	Example . . . . .	89
 <b>III KIDS-CT</b>		<b>92</b>
 <b>8 System Architecture For The KIDS-CT Scanner</b>		<b>94</b>
8.1	CT Requirement Classification . . . . .	94
8.2	System Architecture . . . . .	95
8.3	Communication Infrastructure . . . . .	97
8.4	Optimization Of The Acquisition And Processing Datapath . . . . .	99
 <b>9 Control-Data Acquisition System For The KIDS-CT Scanner</b>		<b>102</b>
9.1	Hardware/Software Architecture . . . . .	102
9.2	Control-Synchronization Module . . . . .	103
9.2.1	Software architecture . . . . .	104
9.2.2	Hardware architecture . . . . .	104
9.3	Data-Flow Module . . . . .	106
9.4	Data-Processing Module . . . . .	109
9.5	Pixel Processing Optimization . . . . .	110
9.5.1	I0-correction step . . . . .	111
9.5.2	Cosine weighting and redundancy weighting steps . . . . .	112
9.6	Design Space Exploration . . . . .	115
9.6.1	Selection of input parameters . . . . .	116
9.6.2	Selection of metrics . . . . .	117
9.7	Component Isolation . . . . .	118
 <b>IV Validation &amp; Evaluation</b>		<b>120</b>
 <b>10 Validation</b>		<b>122</b>
10.1	Validation methodology . . . . .	122
10.2	CDAS Design Phase . . . . .	123
10.3	CDAS Post-Implementation Phase . . . . .	125
10.4	KIDS-CT Post-Integration Phase . . . . .	126
 <b>11 Performance Evaluation</b>		<b>127</b>
11.1	CDAS Architecture For The KIDS-CT Scanner . . . . .	127
11.2	Data-Flow Module . . . . .	130
11.2.1	Timing Analysis . . . . .	132

11.3 Data-Processing Module . . . . .	134
11.4 Lightweight Protection Unit . . . . .	136
<b>12 Design Space Exploration</b>	<b>139</b>
12.1 Image Quality Prerequisites . . . . .	139
12.1.1 CT scanning configuration . . . . .	140
12.1.2 Phantom selection . . . . .	141
12.1.3 Calculation of the image quality metrics . . . . .	143
12.2 Image Quality Analysis . . . . .	144
12.3 Hardware Cost & Computing Performance . . . . .	149
12.4 Design Space Exploration Considerations . . . . .	151
<b>13 Evaluation Of Functionalities</b>	<b>152</b>
13.1 Plug-and-Play Capability . . . . .	152
13.2 Real-Time Support . . . . .	154
13.3 Comparison With Related Work . . . . .	155
<b>V Finale</b>	<b>158</b>
<b>14 Conclusion</b>	<b>160</b>
14.1 Summary . . . . .	160
14.2 Discussion Of Results . . . . .	162
14.3 Future Work: Adaptive Computing Acceleration Platforms For CDAS . . .	164
<b>List of Figures</b>	<b>165</b>
<b>List of Tables</b>	<b>168</b>
<b>List of Acronyms</b>	<b>169</b>
<b>Bibliography</b>	<b>176</b>

# **Part I**

## **Introduction**



# 1 Introduction

Interventional Computed Tomography (iCT) procedures are increasingly being used to treat the growing number of tumors. To perform such medical procedures, surgeons need Computed Tomography (CT) scanners capable of acquiring and reconstructing images in real time. In such an environment, these devices can be modeled as Cyber-Physical Systems (CPSs), where different sensors/actuators need to be controlled and synchronized while data are acquired and processed in real time [1]. Due to the interoperability difficulties between components from different vendors that often acquire and store data offline, supporting plug-and-play capability and real-time controlling/synchronization and data processing is still an open challenge for many CPS applications, like in CT scanners [2–4]. Often, these tasks are distributed across different Control System (CS) and Data Acquisition System (DAS) architectures [5].

This thesis addresses these challenges and proposes a solution considering the CPS System Architecture, its Communication Infrastructure, and the Control-Data Acquisition System (CDAS) architectures. A Centralized System Architecture not only enhances interoperability for plug-and-play capabilities but also improves the estimation of Worst-Case Execution Time (WCET) for communication tasks which are typically distributed across various components. The Communication Infrastructure employs distinct “layers” and “classes” to segregate control/data and non-real-time/real-time tasks. In contrast to other systems where CS and DAS exist in separate architectures, all these elements permit to join Control and Data Acquisition architectures in a novel CDAS architecture based on MPSoC-FPGA platforms. In this way, the various components are coordinated within a single chip where also data are processed, while the plug-and-play capability and the real-time support are provided in the target CPS. Indeed, this new hardware/software architecture is responsible for controlling/synchronizing CPS device components while data are being acquired and processed on the fly. Finally, in order to demonstrate the impact of the proposed solution, this work focuses on the iCT application and the “**K**onfigurierbarer, **I**nterfaceoffener, **D**osissparender **C**omputertomograph (KIDS-CT)” scanner, where it has been realized, validated and evaluated.

## 1.1 Motivation

The motivation for this thesis is manifold. Starting from the relevance of the targeted application and its impact, the motivation can be condensed to the following questions:

### **1.1.1 Why interventional and multimodality CT matter?**

The increasing number of tumors is pushing researchers to explore new medical procedures where single or combined radiological images are used during diagnosis and surgery [6]. Since CT imaging is one of the most effective support in cancer diagnosis, surgeons started to exploit it during tumor ablation [7]. Due to the different application requirements associated to interventional procedures, different radiology scanners can be used. For this purpose, these must be synchronized to generate images simultaneously and obtain useful combined images. So, this CT multimodality imaging became a key element during interventional procedures, contrasting tumors [8]. These factors motivate researchers and underscore the importance of these CT procedures for health and human life. In addition, to fulfill the requirements of this new CT application, the real-time support and the plug-and-play capability became essential requirements for their realization.

### **1.1.2 Why KIDS-CT?**

“KIDS-CT” is the first open-interface CT scanner assembled by and in Academia [9]. It is an open-interface CT platform where it is possible to add and exchange components such as X-ray tubes and detector systems in a plug-and-play fashion. Moreover, it has been designed to exploit and explore multimodality techniques. For instance, it can be combined with other devices like Time-of-Flight cameras and Ultrasound scanners. These features allow researchers to explore and test new reconstruction algorithms and new sensors/actuators suitable for new diagnostic and interventional procedures [10].

### **1.1.3 Why a CT scanner for multimodality/interventional like KIDS-CT is a CPS?**

A CT scanner for diagnostic procedures acquires and reconstructs images offline without real-time interactions between the physical and the cyber world [11]. In contrast, scanners for iCT procedures provide real-time images used to control the needle insertion and its position inside the patient’s body. These real-time images guide surgeons in the tumor ablation intervention [12, 13]. In addition, based on the needle position, the scanner parameters are manually or automatically adjusted to save X-ray doses. This medical procedure is a mere example of real-time interaction between the cyber and the physical world, given by the CT scanner and surgeon/patient, respectively. This real-time interaction defines the distinction between an embedded system and a CPS, such as the KIDS-CT scanner.

### **1.1.4 Why use MPSoC-FPGAs?**

As explained above, a CT scanner for interventional procedures can be modeled as a CPS, where tasks are executed in real time, and the plug-and-play capability should

be provided for enabling multimodality techniques. To fulfill these strict requirements, powerful reprogrammable architectures are the most suitable platforms [14]. In recent years, MPSoC-FPGAs have become the de-facto architecture for reprogrammable architectures [15]. They offer the computation capability of an accelerator and the hardware/software reprogrammability capability that gives the flexibility to extend the architecture for new components and use cases, such as in the targeted medical application [10, 16]. Furthermore, these platforms are suitable for ensuring the security and the dependability level of these Mixed-Criticality Systems (MCSs) such as the KIDS-CT scanner, providing different levels of security and isolation between tasks and peripherals associated with different application domains.

## 1.2 Research Questions

Pursued by the importance of providing an open-interface CT scanner such as the KIDS-CT scanner for interventional and multimodality exploration and by the offered potentiality of the MPSoC-FPGA platform in providing real-time support and plug-and-play capability in CPS applications, this work will answer the following research questions:

- How can a CS and a DAS be combined into a CDAS architecture for MPSoC-FPGAs targeting CPS applications such as the KIDS-CT scanner?
- How can CDAS tasks be partitioned on MPSoC-FPGAs to provide real-time support and plug-and-play capability?
- How can a CDAS architecture be designed for an MPSoC-FPGA to acquire and process data on the fly?
- Which data formats optimize computing performance and hardware cost of the CT pre-processing steps while keeping the image quality suitable for iCT?
- How can task isolation be achieved in low-cost MPSoC-FPGAs while adhering to the specific requirements of CPSs as MCSs?

## 1.3 Research Contributions

In relation to the above questions, this work provides the following research contributions:

- A Communication Infrastructure with the related centralized control unit that supports custom and standard vendor-protocols. It also proposes various application protocols for real-time and non-real-time commands and data communication tasks. Here, the centralized control unit is part of the proposed CDAS. This solution, facilitates the interoperability between CPS components and the support for plug-and-play capability within CPS devices [DP 2].

- A hardware/software architecture for CDAS within CPS applications. The proposed architecture targets MPSoC-FPGAs. It provides support for real-time data processing and plug-and-play capability for adding components to the CPS. It is also optimized for iCT applications [DP 3, DP 4].
- A system architecture for diagnostic and interventional CT that guides hardware and system designers in identifying design requirements of a Medical Cyber-Physical System (MCPS) like the different CT scanners [DP 3].
- A task partitioning methodology for MPSoC-FPGAs, where control, communication, and processing tasks are distinguished and mapped on Processing System (PS) and Programmable Logic (PL) in relation the timing requirements and criticality [DP 4].
- A lightweight dataflow architecture that can be configured at design time and run time for collecting data from various devices with different acquisition requirements (e.g., various protocols and data rates) [DP 5].
- A processing core architecture for real-time data processing, configurable for standard and custom data formats. In this processing core, a hardware optimization of the algorithm for the I0-correction CT pre-processing step is proposed [DP 1].
- A Design Space Exploration (DSE) of data formats applied to CT data in the pre-processing steps of the Feldkamp, Davis, and Kress (FDK) algorithm [DP 1].
- An isolation method for MPSoC-FPGAs that allows guaranteeing temporal and spatial isolation [DP 6].
- An MPSoC-FPGA prototype of the proposed architecture, integrated into the KIDS-CT scanner. Here, it is responsible for controlling and synchronizing the various internal components and collecting and pre-processing data in real time [9].

## 1.4 Thesis Outline

Part I, containing this Chapter, introduces the research area of this thesis. Chapter 2 explains the technical background on which this work is based on. Subsequently, Chapter 3 provides an overview of the related work, focusing on hardware/software architectures for controlling components and acquiring data in CPSs for medical applications.

Part II explains the process that led the author to the plug-and-play real-time architecture. Here, Chapter 4 systematically defines the problems of actual CPS devices and in the specific case of the iCT application, and Chapter 5 illustrates the proposed methodology for this research work. Then, Chapter 6 and Chapter 7 describe the proposed System Architecture with the Communication Infrastructure, and the CDAS architecture for



CPS devices, respectively. This last Chapter describes the different aspects of the CDAS architecture that contribute to achieving the research objectives.

Part III focuses on the targeted CPS use case: the KIDS-CT scanner. Here, Chapter 8 defines the requirements and describes how the proposed methodology and System Architecture for CPSs fits to model the KIDS-CT scanner. Chapter 9 then illustrates the CDAS architecture implemented on the XC7Z045 MPSoC-FPGA and integrated into the KIDS-CT scanner. This includes all the proposed mapping solutions, the modules for controlling-synchronizing components, and for collecting and processing data in real time. Furthermore, this Chapter focuses on the specific optimization proposed for the pixel processing part of the FDK algorithm, which is implemented in the CT reconstruction system. In addition, it describes how metrics and parameters have been selected to explore the design space related to the pixel processing part.

Part IV discusses the validation and evaluation of the proposed work, focusing on its realization and implementation for the KIDS-CT scanner use case. Here, Chapter 10 explains how the CDAS architecture was validated in the design, in the post-implementation, and in the post-integration phases, along with the System Architecture and the Communication Infrastructure. Chapter 11 evaluates the computing performance and the hardware cost of the implemented CDAS architecture implemented for the KIDS-CT scanner; it also considers other configurations of the different modules to show how the proposed architecture can also be extended to support additional components. Chapter 12 reports the results of the image quality analysis applied to the reconstructed images, where different data formats have been used in the reconstruction algorithm; it explores the design space, considering which data format is more suitable for interventional procedures. Chapter 13 evaluates and compares the proposed functionalities with the related work, focussing on the plug-and-play capability and real-time support.

Finally, in Part V, Chapter 14 concludes this thesis by discussing the achieved research objective and presenting open problems for future research work.

## 2 Technical Background

This Chapter introduces the fundamental concepts on which this thesis is based on. Initially, it defines a CPS device, clarifying the difference with an Embedded System (ES). In this context, it reports the various aspects of designing CPSs and introduces the case of MCSs, which are computing systems like KIDS-CT that can run tasks of different criticality. Then, it describes the MPSoC-FPGA, which is the ES platform selected for the proposed CDAS architecture. It also discusses the hardware isolation challenges focusing on the targeted platform. After that, it describes the background of the CT application, focusing on the KIDS-CT scanner that is the selected CPS use case of this thesis. Finally, it explains the fixed-point and floating-point representations used in this work as input parameters of the DSE for the CT pre-processing steps. The contents of this Chapter have also been presented by the author in Ref. [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7, DP 8].

### 2.1 Cyber-Physical Systems

From the computing and communication perspectives, various definitions of Cyber-Physical Systems (CPSs) [17] can be found in the literature. The closest to this work is given by A.E. Lee, who focuses on the computational aspects and defines CPSs as follows: “*Cyber-Physical Systems are integrations of computation and physical processes*” [18]. This definition takes into account two elements: the computation and the physical environment (i.e., physical processes). The link between these two elements allows CPSs to be distinguished from Embedded Systems (ESs), such as smartphones, where there is no physical environment to consider in the modelling and designing process [18]. In fact, “a CPS comprises an embedded system (the information processing part) and a (dynamic) physical environment, or **CPS = ES + (dynamic) physical environment**” [17], as shown in Fig. 2.1.

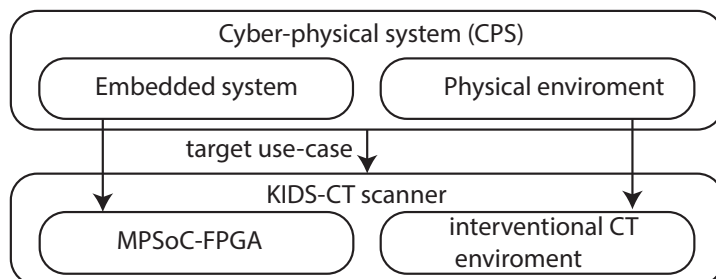


Figure 2.1: Integration of computation and physical environment

Furthermore, in order to understand the concept of CPS, it is important to define the meaning of “system” E.A. Lee defines it as follows: “*A system is simply a combination of parts that is considered as a whole*” [18]. In fact, a CPS device can be defined as a system composed of physical subsystems with their interconnections and computing units. For instance, the CT scanner contains several components that are standalone systems; the interaction of these interconnected components with the physical environment constitutes the CT scanner as CPS. Although, there are several definitions of CPSs, which also emphasize the concept of interaction between devices connected to the Internet, this thesis considers CPSs as a device made up of components (sensors/actuators/processing systems) interconnected through a point-to-point connection or a local area network where the hardware platform selected for the ESs becomes crucial. In fact, this research work analyses and proposes solutions to solve problems such as the plug-and-play capability and the real-time support through the use of MPSoC-FPGA platforms, independently of the physical environment of the application.

### 2.1.1 Physical environment

As mentioned above, a CPS comprises a physical environment, also called physical system. This refers to the real-world settings in which the CPS operates, including everything from the physical machines and sensors to the natural environment around them [18]. Different models can be used for describing a physical environment based on the interesting aspects to represent the physical environment. For example, designers can use data-driven models, mathematical models, formal models et cetera. Modeling the physical environment accurately is crucial for the effective design, implementation, and operation of CPSs, as it directly impacts decision-making, system efficiency, and safety.

This thesis targets the CT scanner for interventional procedures, which is a medical CPS where the surgeon interacts with the patient through the ablation needle and is guided by real-time images from the CT scanner. These elements and their interactions represent the physical environment, which is strictly application-dependent, as well as the input parameters for the proposed plug-and-play real-time architecture for MPSoC-FPGAs. Defining modelling solutions for the physical environment of the target application is out of the scope of this thesis, instead a System Architecture and Communication Infrastructure models are proposed. These permit to realize existing physical environments such as the autonomous CT scanner for interventional procedures proposed by the author in [DP 8].

### 2.1.2 Embedded system

An Embedded System (ES) can be defined as follows: “*An embedded system is a microprocessor-based system that is built to control a function or set of functions and is not designed to be programmed by the end user in the same way as a Personal Computer (PC)*”.[19]. This

definition immediately points to the problem of control, which is an essential element of ESs. Further, it highlights the impossibility for the end user to program an ES as a PC, since an ES requires an external system to be programmed. This makes a difference with general purpose systems like PCs. An ES consists of three parts:

- **Sensor:** It is a device that measures a physical quantity from the physical environment [18]. Here, the physical quantity is “*sampled and hold*” as an analog signal and converted into a digital signal to be processed.
- **Actuator:** It is a device that modifies a physical quantity of the physical environment [18]. Here, a digital signal is properly converted into an analog signal that is responsible for modifying the physical quantity.
- **Processing Unit:** It collects and processes sensor data and controls actuators. It also implements the functionalities of the ES (i.e., use cases of the target CPS device). The proposed “*plug-and-play real-time architecture*” for MPSoC-FPGA is part of the Processing Unit in the CPS.

### 2.1.3 Physical architecture

Modern CPSs contain many sensor/actuator components that generate a vast amount of data that cannot be sent and processed in time using the Cloud [20]. Therefore, these data must be processed on Edge at the device level [20]. For instance, Heinrich [21] reports that car sensors can generate about 22 GB for an event of 35 seconds. It means that a car may generate over 18 TB of data in 8 hours of driving. The same problem also exists for the KIDS-CT scanner that can generate about 24 GB of data in 30 seconds [22], only from the detector sensors.

Furthermore, to coordinate and synchronize these complex systems and to provide the control and processing algorithms at the device level, different control architectures exist [23–25]. As shown in Fig. 2.2, the various solutions can be classified into three types of control architectures: centralized, distributed, and decentralized along sensors and actuators on single or multiple chips [24, 25].

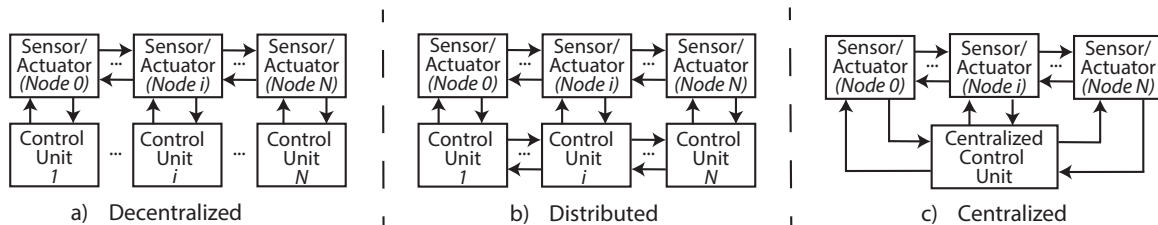


Figure 2.2: Control architectures

In CPS applications with reliable and hard real-time requirements, such as automotive systems, CPS vendors tend to choose the centralized solution [26]. For instance, AMD-

Xilinx [27] and Tesla [28] have introduced a Full-Self Driving (FSD) system for Advanced Driver-Assistance Systems (ADASs) that implements a centralized control architecture. Following the same trend, the system architecture proposed in this thesis is based on the centralized control architecture [DP 3].

## 2.2 Multi-Processor System-on-Chip Field-Programmable Gate Array

In recent years, technological advances have allowed the development of ESs on Multi-Processor System-on-Chips (MPSoCs) [29]. These platforms have multiple Central Processing Units (CPUs) and peripherals that permit data processing and sensor/actuator communication in a single integrated circuit [29].

Multi-Processor System-on-Chip Field Programmable Gate-Arrays (MPSoC-FPGAs) go one step further, providing programmable logic into MPSoCs allowing the implementation of custom peripherals for custom protocols and application-specific functions with great performance and power efficiency [16]. These capabilities make MPSoC-FPGAs appealing architectures for CPSs, compared to traditional processing unit platforms [30]. Following the AMD-Xilinx taxonomy, MPSoC-FPGAs have the PS and PL parts, as shown in Fig. 2.3. The former comprises the Application Processing Unit (APU) and peripherals implemented in the fixed silicon logic. The latter comprises reprogrammable logic capable of implementing Processing Elements (PEs) [31].

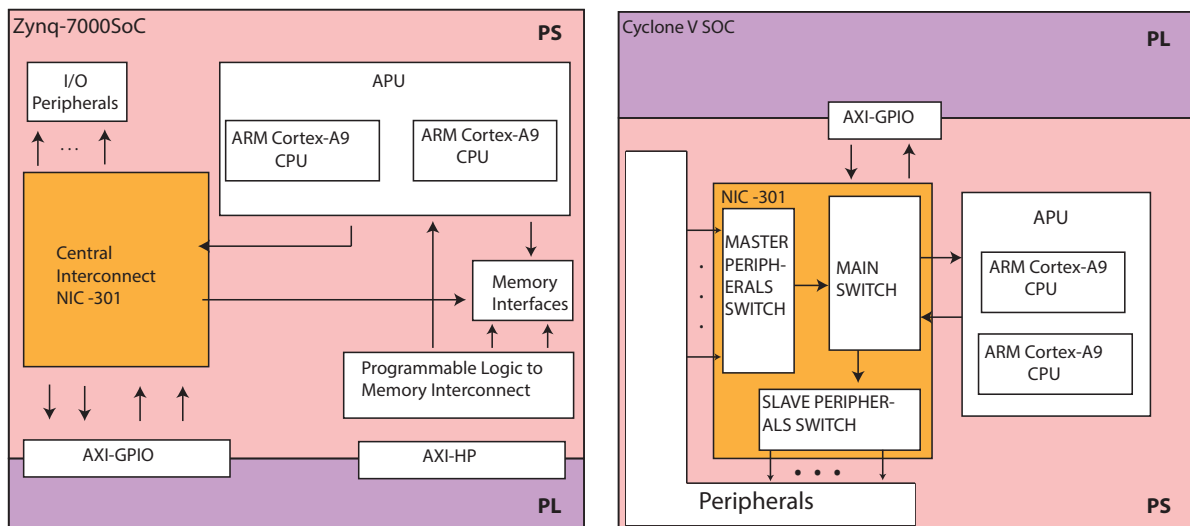


Figure 2.3: MPSoC-FPGA Micro-architecture: Zynq-7000 SoC and Cyclone V SoC

Unlike Application-Specific Integrated Circuits (ASICs), the Field Programmable Gate Array (FPGA) part (i.e., PL) is configured at boot time by the PS, and can also be reconfigured at run time [31]. This technological capability permits upgrading the hardware/software architecture at boot and at run time for possible expansion or replacement of components [32], which has been essential for realizing the proposed plug-and-play

capability.

### 2.2.1 On-chip communication architecture

In modern MPSoC-FPGAs, we can have hundreds of PEs, peripherals, and storage elements, providing flexibility and performance. In order to establish communication between them, designers define on-chip communication technologies and architectures [33]. In the past, each manufacturer defined its communication infrastructure and protocol, which made it difficult to integrate third-party elements, such as soft Intellectual Property (IP) cores. During these years, various bus standards have also been proposed to solve this problem, and the Advanced Microcontroller Bus Architecture (AMBA) was one of the most used. ARM proposed this bus to support the communication between their cores.

Nowadays, the de-facto standard interface for on-chip communication is Advanced eXtensible Interface (AXI), which is part of the AMBA 4 specification [34]. In contrast to most other communication standards, it does not specify a bus infrastructure but a communication interface. In this way, the interfaces may be connected by direct connection, bus, or Network-on-Chip (NoC) architectures also using different technologies and clock frequencies, facilitating the extension of the PS bus to the PL. It describes a point-to-point communication between two complementary master and slave interfaces, offering the following two protocols [35]:

- **Memory-mapped protocol:** Master and slave interfaces are connected by an interconnect component that routes or manipulates transactions between the master and slave. All transactions are associated with a destination address within a system memory space and data to be transferred. For the memory-mapped protocol, AXI4 provides the Full and the Lite versions. While AXI4-Lite performs single transactions using few logic signals, AXI4-Full supports 256 burst transactions in a single address phase.
- **Stream protocol:** It consists of a direct unidirectional connection between a master and a slave interface, without the use of addresses.

While the stream communication between two components uses a dedicated physical link, the memory-mapped communication uses an interconnect component, potentially permitting communication between a master and a slave interface and causing isolation problems. For this reason, designers and researchers consider only the memory-mapped protocol for the isolation issue.

In order to read/write data in memory-mapped mode, AXI4 interface uses five channels: Read Address channel (RA), Read Data channel (RD), Write Address channel (WA), Write Data channel (WD) and Write Response (WR) [36]. Thanks to the separation between read and write channels and between address and data channels, the protocol

allows data to be moved between master and slave simultaneously. In the read transaction, the master requests data through the RA channel. Then, the slave responds through the RD channel as shown in Fig. 2.4.

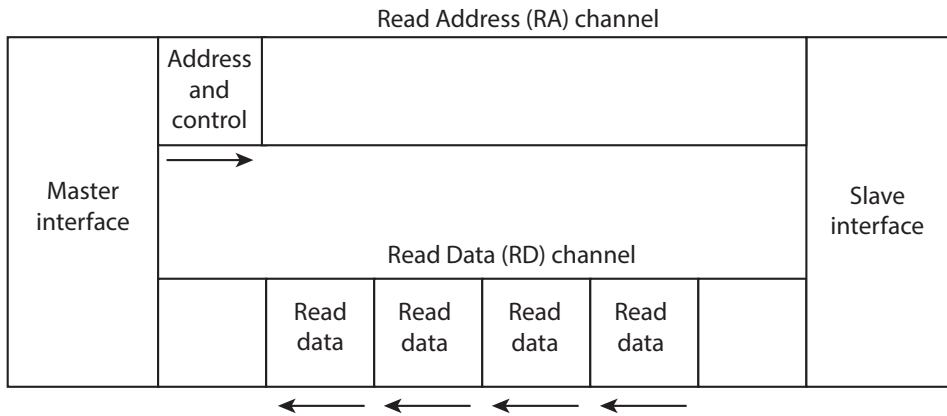


Figure 2.4: AXI4: channel architecture of read operations [36]

In the write transaction, the master first sends the address to write through the WA channel as shown in Fig. 2.5. Then, it sends the data to write through the WD channel. Finally, the receiver sends the acknowledgment message through the WR channel.

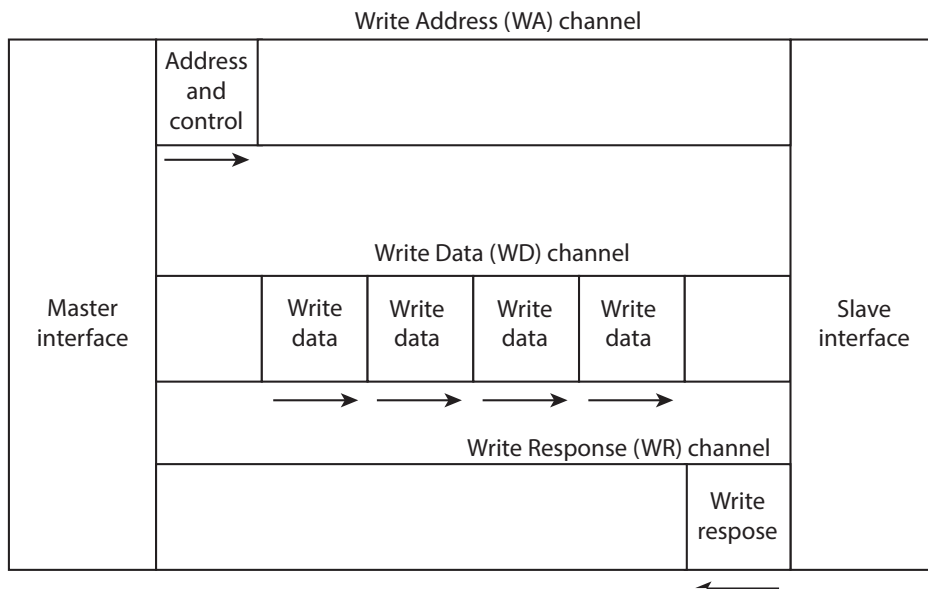


Figure 2.5: AXI4: channel architecture of write operations [36]

At the microarchitecture level, each channel consists of multiple signals, some of which differ from channel to channel. To ensure correct communication, all channels implement a handshake mechanism between the transmitter and the receiver using the following signals:

- **READY** - driven by the receiver to indicate to the sender that it is ready to receive data.

- VALID - driven by the sender to signal the validity of a transmission data/address.

The handshake is established when both signals are high. Furthermore, to avoid deadlocks, the master sets the VALID signal high without waiting for the READY signal and sets it low only when the handshake is established and data are sent. To exploit the communication infrastructure for isolation purposes, it is important to consider the following signals of the address channels:

- Identifier (ID) - used to differentiate in-flight transactions; a master can be identified by a fixed Section of the ID.
- Address (ADDR) - address of the first data to transfer
- Length (LEN) - number of burst data to send through the data channel for the related transaction
- Size (SIZE) - the size of each burst to send

The signals ADDR, LEN, and SIZE describe which data will be transferred. Since AXI-Lite does not support burst transmission, it does not have ID, LEN, and SIZE signals.

The WD and RD channels transport the data through the data signals (DATA). Finally, when a transaction is completed, the WRs channel uses the Response (RESP) signal to indicate the success or the failure of the transaction [36].

### 2.3 Mixed-Criticality Systems

In Mixed-Criticality Systems (MCSs), tasks are associated with criticality levels on a shared platform [37]. These levels range from low criticality, where failure or misbehavior may not affect the physical environment, to high criticality, where failure or misbehavior could lead to catastrophic events. Therefore, each criticality level must specify a required level of assurance against failure. In this context, the failure is a consequence of a fault trigger [38]. In fact, the fault trigger is defined as the set of conditions that activate a fault and propagate the resulting errors into a failure [39]. There are two types of faults: temporal and functional. The former usually arise from the long task execution times, while the latter include permanent or temporary defects. Various techniques exist for fault tolerance in CPSs [40]. Most works that are focused on temporal faults determine the upper bounds to the execution time of the provided tasks, commonly called WCET [41]. In MCSs, to reach a high level of assurance, different confidence of the WCET estimation is associated with the criticality levels [42].



### 2.3.1 Shared resources in MCSs

Modern hardware platforms, such as Multi-Cores and MPSoCs, can run multiple tasks in parallel. On the one hand, they offer performance optimization; on the other hand, they raise the problem of *interference* between tasks of different applications that share resources [43].

The interference problem opens two critical issues in MCSs: the WCETs estimation and the lack of security. For example, an MPSoC-FPGA may run a high-critical task on one core and a low-critical task on another core, sharing the same peripherals; in this scenario, the low-critical task may cause a functional failure by manipulating the shared resource, or it may cause a temporal failure by accessing the shared resources longer than expected.

For this reason, tasks in such systems are usually associated with *criticality domains*. Here, tasks of different domains are isolated from each other, and a shared resource can exclusively be accessed only by tasks of a granted domain. There are two types of isolation: temporal and spatial isolation. The former guarantees that tasks accessing a shared resource cannot interfere with other tasks in the time domain. The latter guarantees that tasks have exclusive access within the same shared resource.

### 2.3.2 CPSs and MCSs

In parallel with the evolution of separate research branches for CPSs and MCSs, it has been noticed that many CPSs are also MCSs [37]. In this context, Schneider et al. observed that numerous CPS operations must perform deadline-critical and Quality of Service (QoS)-critical tasks [44]. For example, CT scanners have critical tasks (e.g., High Voltage (HV) control and synchronization tasks) where safety and reliability are essential requirements. In order to meet these requirements and to avoid errors from propagating throughout the system while critical and non-critical tasks are running, isolation is the primary solution [45]. In addition, critical tasks can be part of two different applications, which must be isolated to improve security and dependability levels.

### 2.3.3 MPSoC-FPGAs for MCSs

When MPSoC-FPGAs are used in CPS applications, which are also MCSs, tasks with a different criticality and/or application domain must be isolated, as explained above. Here, tasks can also be implemented in dedicated hardware modules on the PL part. In order to guarantee isolation in MPSoC-FPGAs, which use the memory-mapped input-output, first, the tasks or components are associated with one or more domains. Second, the resources refer to peripherals or specific memory regions. Finally, the policies which grant/deny communication transactions are defined. For this reason, the communication infrastructure is crucial to guarantee isolation. The following two main approaches are utilized to provide isolation in MPSoC and platforms:

- **Protection Unit (PU):** It is a hardware unit that usually is part of a CPU architecture or an IP core implemented on the PL. For instance, Arm-M processors [46] integrate an Memory Protection Unit (MPU) that is supported by e. g. FreeRTOS [47]. It controls the access to memory regions that are configured by privileged software, giving access rights only to certain applications. The MPU is similar to the Memory Management Units (MMUs), but does not provide address translation, making it suitable for embedded use cases.
- **Hypervisor:** It is an additional management and security system that is implemented between the Operating Systems (OSs) and the hardware layer, enabling the parallel running of several OSs in a secure way (i.e., it guarantees the isolation between the resources accessed by the different OSs). Furthermore, it offers and manages virtualization, where a single physical resource is divided into multiple virtual resources. Virtualized resources can include CPUs, memory, I/O peripherals, timers, and interrupts. To minimize the performance degradation associated with virtualization, hardware platforms are designed to support this functionality. In particular, CPU virtualization is enhanced by the introduction of additional CPU modes [48]. Moreover, hypervisors and virtualization facilitate the deployment of legacy applications [49]. The isolation mechanism proposed in this thesis lays the foundation for supporting hypervisors in low-cost MPSoC-FPGA platforms.

## 2.4 Interventional Computed Tomography

Interventional Computed Tomography (iCT), also known as CT-guided intervention, is a medical procedure that uses the CT technique to guide a minimally invasive intervention in a specific area of the body. It is mainly used as a standalone system or in combination with PET, MRI, and/or Ultrasound for guiding tumor ablation [50]. During an interventional CT procedure, the patient lies on a table that slides into the CT scanner, which uses X-rays to create detailed images of the body's internal structures. CT scanners for iCT procedures aim to acquire and display reconstructed images in real time, allowing the surgeon to precisely locate the Region Of Interest (ROI) and guide the needle or other medical instruments to the desired location [13]. The following Sections describe the basics of CT scanners, the KIDS-CT scanner, and the theory of CT reconstruction.

### 2.4.1 CT scanner fundamentals

A CT scanner is a medical imaging device that uses X-rays and computer technology to create detailed images of bones and soft tissues. To do so, an X-ray detector and an X-ray tube rotate around the patient and multiple X-ray images are taken from different angles [11]. A reconstruction unit then processes these projected images called *projections*

to create cross-sectional images of the body that can be viewed on a monitor.

Nowadays, CT scanners come in a variety of sizes and configurations, ranging from small portable units to large, multi-slice machines that can produce high-resolution images of the entire body taking several minutes [51]. The latest generation of CT scanners also incorporates advanced technologies such as low-dose radiation techniques [52], iterative reconstruction algorithms [53], and dual-energy imaging [54], which can improve image quality while minimizing radiation exposure to the patient. In addition, different scanners with specific shapes and rotation mechanisms are used depending on the body part of interest: mammography scanners for the breast, dental cone beam CT scanners for teeth, C-arm and helical CT scanners for scanning bones and soft tissues [55].

Although distinct scanners are used for different body parts, physicians initially used the same scanners for diagnostic and interventional procedures. However, due to the different timing and X-ray dose required for the two applications, companies have begun to design scanners with identical shape and geometry parameters but different control mechanisms, dose modulation techniques, and data acquisition and reconstruction components for the two procedures. For example, Philips and Siemens have proposed two platform systems for different interventional procedures [56, 57]. Furthermore, various scanners enabling multimodality techniques have also been proposed, such as the "MIYABI Angio-CT" [58], by Siemens, which is also used in surgery rooms for iCT procedures.

#### 2.4.2 KIDS-CT scanner

This work focuses on the KIDS-CT platform, which is an open-interface CT scanner [9]. This scanner has been designed to integrate additional components in a plug-and-play fashion and to run various CT reconstruction algorithms. It can perform axial and helical CT scans of the whole body, which are the most common modes for clinical applications.

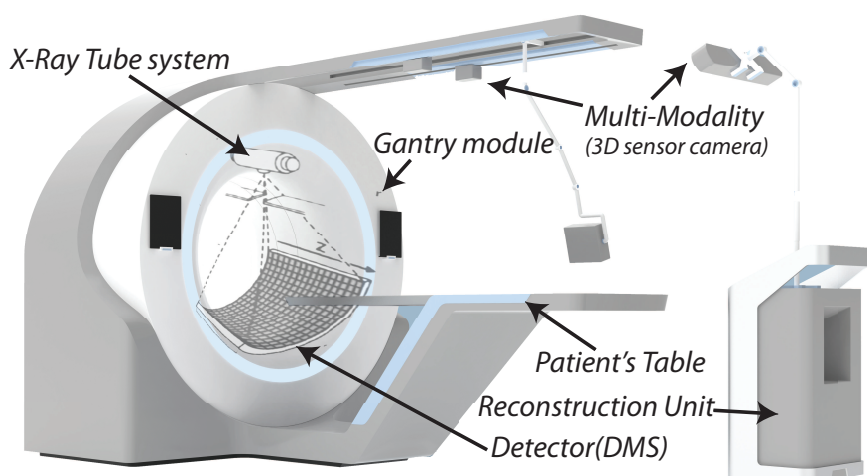


Figure 2.6: Representation of the KIDS-CT scanner [DP 3]. Copyright 2019, IEEE

In its basic configuration, the KIDS-CT scanner consists of the main components shown in Fig. 2.6: X-ray tube system, Detector Management System (DMS), gantry module, patient table, collimator module, and reconstruction system. In order to define the requirements for the KIDS-CT platform the main internal components are described below.

### **X-ray tube system**

The X-ray tube system consists of the X-ray source, the cooling unit, and the power distribution system. The KIDS-CT platform mounts the “Xpert bundle with CT6500” in Ref. [59]. Typically, this generates an X-ray beam with multi-focal spot jumping at the nominal voltage of 140 kV [59]. Here, the position of the X-ray source is identified by a single point called focal spot  $f$ , as shown in Fig. 2.9.

From a system design point of view, this component has several interfaces for real-time and non-real-time control tasks and signals. It is mounted on the rotating side of the gantry, where it is directly connected to the proposed CDAS.

### **Detector Management System**

The Detector Management System (DMS) contains the matrix of the detectors, which are irradiated by the attenuated X-ray flux. The acquired signal is converted into digital information and sent to a DAS, which collects and processes all data. In the KIDS-CT platform, the DAS responsible for collecting detector data is part of the proposed architecture for CDAS. The DMS mounted on the KIDS-CT platform is the “Philips - 64 row CD300” in Ref. [22]. This is a cylindrical detector with 43008 detection elements arranged along an arc. The detector elements are distributed as rounded rectangular. The geometry distribution, which is essential for reconstructing the 3-Dimensional (3D) voxel pixel, is explained in Section 2.4.3.

### **Patient table**

For making the scan, a patient lies on a table called the patient table. During the scan, it is used for positioning the patient between the DMS and the X-ray source [11]. When the scanner is set to the helical mode for a full body scan, the table is moved towards or away from the gantry while the DMS, the collimators, and the X-ray tube systems rotate. In this application, the patient table, the gantry, the DMS, the collimators, and the X-ray tube system must synchronize their respective positions and speeds in order to acquire useful images for the reconstruction. Furthermore, to center the body part to be irradiated with the X-ray source at the iso-center, the KIDS-CT patient table has been designed with three degrees of movement that must be controlled in real time. The patient table has

real-time and non-real-time interfaces to communicate with the other CPS components for controlling and synchronizing issues.

### **Collimator system**

In CT scanners, collimators perform two primary roles: limiting unnecessary patient exposure to radiation and guaranteeing high-quality imaging. This component is situated between the X-ray source and the patient. This mechanical device can be manually set before the scanning or controlled by an engine that changes the direction of the X-ray beam on the patient. In the KIDS-CT platform, the collimator system uses a real-time interface to communicate with the CDAS.

### **Gantry**

The gantry is the mounting framework that surrounds the patient [11]. It mainly contains the X-ray tube system, the DMS, and the collimators. These components are mounted on a rotating disk (rotating side), that must fulfill positional and angular accuracy. For high positional accuracy, the gantry has to limit the vibration in all directions. For high angular accuracy, the disk speed rotation must be constant. For example, in the medical application, the error for a fraction of millimeters for each collected image has an upper bound of 500-mm radius [11].

The data are acquired on the rotating side but must be sent to the reconstruction system on the scanner's stationary side. Slip-ring technology is used to power devices mounted on the gantry's rotating disk and establishes communication with devices on the stationary side. It includes an electromechanical device consisting of concentric circular rings aligned with the gantry axes. Brushes connect both ends of the gantry to these rings, ensuring an electrical link between the rotating and stationary parts of the device [60]. The KIDS-CT scanner uses a Schleifring Gantry [61].

### **Reconstruction system**

The reconstruction system is situated on the stationary side. It collects the acquired data streamed via the slip-ring communication link, reconstructs them, and displays them to the doctor. It consists of a workstation PC having an MPSoC-FPGA for the synchronization and the data collection and a GPU accelerator for the image reconstruction. Both these platforms are connected to the PC via a Peripheral Component Interconnect Express (PCI-E) interface. The collected data are sent from the MPSoC-FPGA through the PC main memory to the GPU accelerator. Furthermore, to collect data with the MPSoC-FPGA, the proposed CDAS architecture has been reused and configured only for this purpose.

### User interface system

The user interface system is situated on the stationary side. It is used by surgeons or radiologists to control the scanner for acquiring and visualizing the reconstituted 3D images. It can also be implemented in the same workstation of the reconstruction system.

### Controlling and data acquisition system

The Controlling and Data Acquisition System is the core of the KIDS-CT platform. The proposed architecture controls and synchronizes all the KIDS-CT components in real time, realizing a centralized CDAS for MPSoC-FPGA. Usually, commercial CT scanners use various DAS and CDAS architectures, which are custom-designed for the specific device to implement. For this reason, there is no generic CDAS architecture defined in the literature. In Chapter 3, the author will describe the significant existing solutions for CT and other CPS applications as related work of this thesis. Part II of the thesis will describe the proposed architecture that realizes the CDAS.

## 2.4.3 CT reconstruction theory

### Fundamentals of CT reconstruction

As explained above, a CT scanner measures the X-ray flux passing through an object from different angles and reconstructs the 3D voxels. If there is no object between the X-ray source and the detector sensors, the measured values in the ideal case are constant because there is no attenuation [11]. In clinical applications, bones and soft tissues are considered to be non-uniform objects, with different attenuations measured at different angles. Furthermore, the generated X-ray flux is mono-energetic, so the relation between the input X-ray photons  $I_0$  (generated by the X-ray source) and the output X-ray photons  $I$  (measured by the DMS) is expressed by the Beer-Lambert law, in the equation 2.1.

$$I = I_0 e^{-\mu_1 \Delta x} e^{-\mu_2 \Delta x} e^{-\mu_3 \Delta x} \dots e^{-\mu_n \Delta x} = I_0 e^{-\sum_{n=-1}^N \mu_n \Delta x} \quad (2.1)$$

In the equation 2.1,  $\Delta x$  and  $\mu$  are the thickness and linear attenuation coefficient of each element inside the object, respectively. This relates the generated and measured intensity of the X-ray photons to the density of the elements crossed by them. In fact, reconstruction algorithms aim to find the  $\mu$  value for each element (e.g., soft tissue, bone) inside the object, starting from projected images, where each pixel represents a measured X-ray photon intensity.

A practical example is shown in Fig. 2.7, where a simple non-uniform object with four elements has been considered. Each element has a different  $\mu$ , and the reconstruction algorithm aims to compute them, starting from the four acquired projections  $p_1$ ,  $p_2$ ,

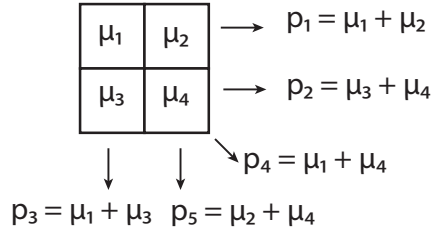


Figure 2.7: Example of a 4 voxel object and its projections

$p_3$ , and  $p_4$ . In this simple case, the algebraic system has a unique solution because there is the same number of equations and unknowns. Therefore, it can be solved with a mathematical approach using the direct matrix inversion. The same mathematical approach was used for the first CT scanner in 1967 [11], but in modern CT scanners, this solution is computationally not applicable due to the complexity of the system.

Nowadays, various methods are used in the state-of-the-art, such as back-projection, statistical, and machine-learning methods [62]. However, the most successful algorithm is the FDK, which uses the Filtered Back-Projection (FBP) method. Due to its low computational cost, the FDK algorithm has also been selected for the KIDS-CT platform and this thesis.

### FDK algorithm

The FDK algorithm was published in 1984 [63]. It is an analytical 3-D reconstruction method for flat panel and cylindrical detectors, usually implemented with a filtered back-projection scheme.

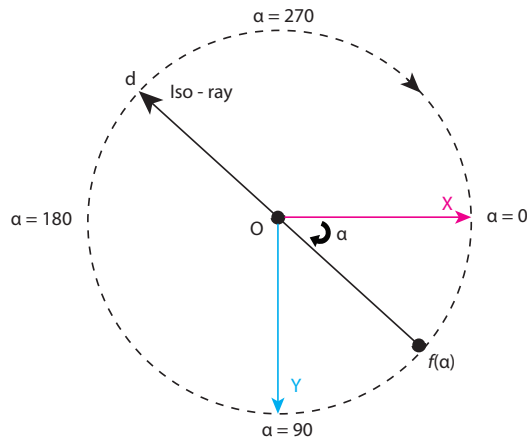


Figure 2.8: CT circular trajectory. “O” represents the iso-center, and “d” is the orthogonal distance between each point  $f(\alpha)$  and the corresponding detector plane

This algorithm considers scanners with a circular trajectory. As shown in Fig. 2.8, the DMS and the X-ray source rotate along the circle in the direction of  $\alpha$ .

The FDK algorithm reconstructs the 3D volume of the scanned object, starting from the

2-Dimensional (2D) projected images. For matching the object, the X-ray source, and the DMS position between the 2D pixels and the 3D voxel to reconstruct, different geometry parameters are required. These parameters are associated with the following coordinate systems:

- **World Coordinate System (WCS):** It is a Cartesian coordinate system that is fixed during the acquisition.

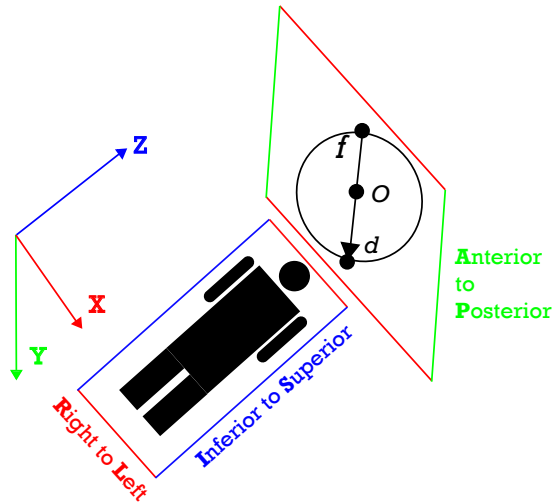


Figure 2.9: World Coordinate System. (X,Y,Z) refer to the Left, Posterior, Superior (LPS) orientation

The WCS is used to indicate the specific locations in 3D global space where the CT system geometry and object projections are positioned [64]. The WCS has two types of orientations: LPS and Right, Anterior, Superior (RAS).

In medical imaging, LPS is commonly used to represent the WCS. As shown in Fig. 2.9, the LPS orientation of the WCS is illustrated, with the positive X, Y, and Z axes positioned from right to left, anterior to posterior, and inferior to the superior of the object body, respectively. The reference point of iso-center ( $O \in \mathbb{R}^3$ ) is located at the center of the WCS in the CT scanning system.

- **Voxel Coordinate System (VCS):** The 3D position vector of voxels, which are discrete units in a 3D space, is defined with discrete indices  $x^i \in \mathbb{Z}^3$ . The volume is represented as a discretized grid of  $N_x \times N_y \times N_z$  voxels along the x-direction, y-direction, and z-direction, respectively. Here,  $N_x$ ,  $N_y$ , and  $N_z$  denote the number of columns, rows, and slices in the volume. The terms  $\Delta x$ ,  $\Delta y$ , and  $\Delta z$  represent the width, height, and depth of each voxel, respectively.
- **Detector Coordinate System (DCS):** It is used to locate the position of the pixel in relation to the center of the projection; as shown in Fig. 2.10, it is orthogonal to the X-ray source.



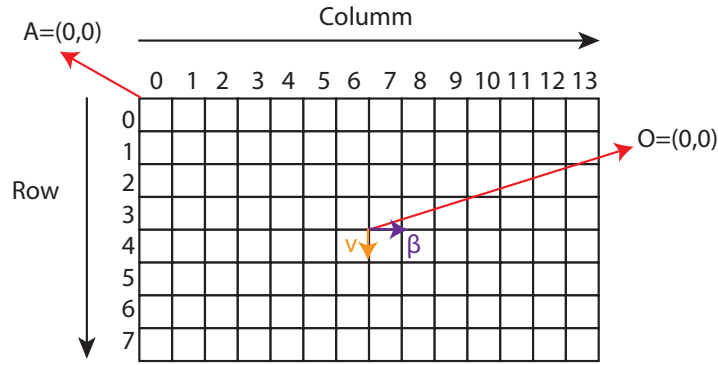


Figure 2.10: Pixel Coordinate System (PCS) and DCS. The rows and columns refer to the PCS, and point the pixel index in the collected projection; the point A has coordinate (0,0) in the PCS. Instead, the point O has coordinates (0,0), following the DCS

- PCS:** It is used to locate a specific point in an image by combining rows and columns to construct an image. As shown in Fig. 2.10, the PCS is ordered from left to right and from top to bottom (rows increment downward, while columns increment to the right). For instance, in the KIDS-CT PCSs represents the data array points of the cylindrical detector plane. As described in Tab. 2.2, the total number of rows and columns of the data array are defined by  $N_v$  and  $N_u$ , while the coordinates are defined as  $v^{pix} \in \mathbb{R}$ :  $v^{pix} \in [1, N_v]$  and  $u^{pix} \in \mathbb{R}$ :  $u^{pix} \in [1, N_u]$ .

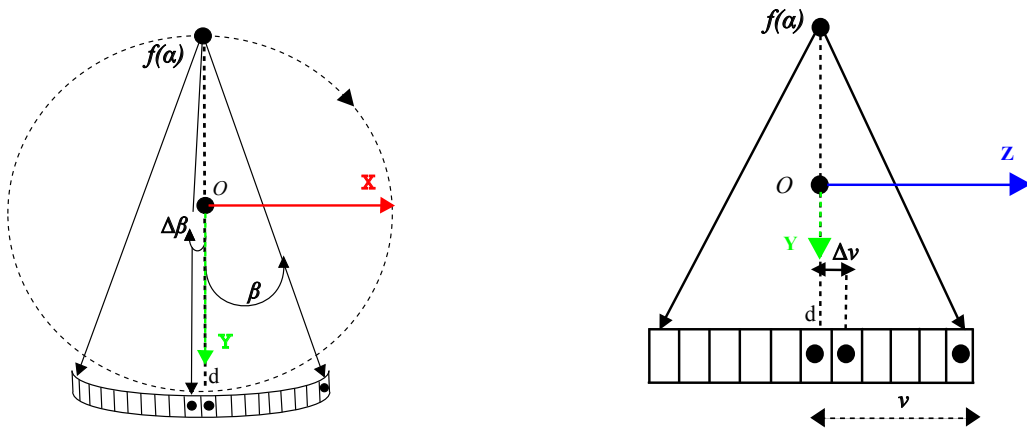


Figure 2.11: Fan Arc Angle (left) and Fan line height (right). The arrows X, Y, and Z refers to the WCS with the LPS orientation

$\Delta v$  and  $\Delta\beta$  are the sampling intervals used to calculate the physical position of the pixel, starting from the indices  $v^{pix}$  and  $u^{pix}$ , respectively. In the case of cylindrical detectors such as the KIDS-CT scanner,  $\Delta v$  and  $\Delta\beta$  represent the pixel height increment and the fan angle increment, respectively. As shown in Fig. 2.11, they are sampled equally in angular and column and row distances, respectively.

### Geometry of CT reconstruction

In CT systems, various geometry parameters are required. As explained above, these parameters are associated with the various coordinate systems and permit the matching between projected data and voxels of the 3D volume. The geometry parameters are divided into three main groups:

- **Detector geometry parameters:** They describe the structural information of the detector, and the values are constant for all projections. These parameters depend on the selected DMS and are described in Tab. 2.1.

Parameter name	Symbol	Unit
Total number of column	$N_u$	1
Total number of rows	$N_v$	1
Each fan angular increment	$\Delta\beta$	radian
Each pixel height	$\Delta v$	mm
Row index	$v^i$	1
Column index	$u^i$	1
Fan arc angle	$\beta$	radian
Fan line height	$v$	mm

Table 2.1: Detector Geometrical Parameters [65]

- **Projection geometry parameters:** They describe the geometric information of the data acquisition and 2D cross-sectional image (projection). The variables are constant for one image because the projection angle  $\alpha$  changes position during the circular scan. These parameters are listed in Tab. 2.2.

Parameter name	Symbol	Unit
Total number of Projection	$N_\alpha$	1
Projection angle	$\alpha$	radian
Source to iso-center distance	$r_f(\alpha)$	mm
Source to detector distance	$r_{fd}(\alpha)$	mm
Beginning rotation angle	$\alpha_0$	radian
Last rotation angle	$\alpha_{N_\alpha-1}$	radian
Angular increment	$\Delta\alpha$	radian
Focal spot position	$\mathbf{f}(\alpha)$	mm
Central row index	$v_0^{pix}(\alpha)$	1
Central column index	$u_0^{pix}(\alpha)$	1
Vector direction along orthogonal to the detector	$\mathbf{X}$	1
Vector direction along detector column direction	$\mathbf{Y}$	1
Vector direction along detector row direction	$\mathbf{Z}$	1

Table 2.2: Projection Geometry Parameters [65]

- **Volume geometry parameters:** They illustrate the geometry of the reconstructed volume after the reconstruction. These parameters depend on the 3D image to be reconstructed and are reported in Tab. 2.3

Parameter name	Symbol	Unit
Total number of columns in the reconstructed volume	$N_x$	1
Total number of rows in the reconstructed volume	$N_y$	1
Total number of slices in the reconstructed volume	$N_z$	1
Reconstructed voxel width	$\Delta x$	mm
Reconstructed voxel height	$\Delta y$	mm
Reconstructed voxel depth	$\Delta z$	mm
Reconstruction center	$x_{rc}$	mm
Volume offset (The 3D position vector of first voxel in WCS)	$x_0$	mm
Voxel index	$x^i$	mm

Table 2.3: Volume Geometry Parameters [65]

### Algorithm steps

In order to reconstruct the 3D volume of the target object, the FDK algorithm preprocesses the 2D projected images, then applies the back-projection, as shown in Fig. 2.12.

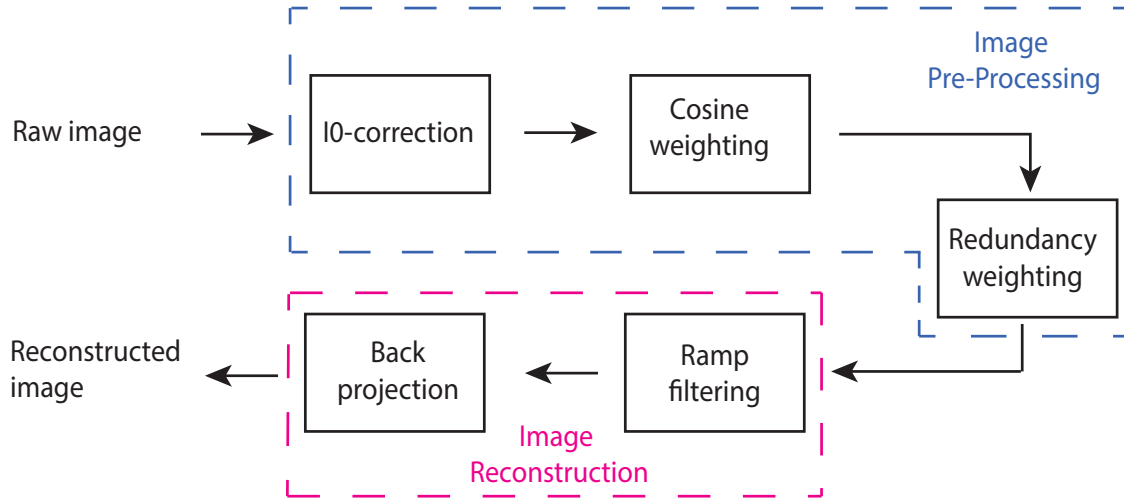


Figure 2.12: FDK algorithm steps

The pre-processing phase is applied to each pixel. Since there is no dependency between pixels of different projections, it can be easily parallelized. The pre-processing phase consists of at least the following three steps:

1. **I0-correction:** The projected images are acquired in the attenuation domain and are denoted by  $d(u, v, \alpha)$  where  $u$  and  $v$  define the detector row and column indices, and  $\alpha$  is the projection angle that indicates the single projection. The data in

the attenuation domain are represented as Natural numbers, like the greyscale pixels. Before processing, these 2D images must be transformed into X-ray intensity projections using the following equation:

$$I(u, v, \alpha) = c \cdot e^{-d(u, v, \alpha)} \quad (2.2)$$

where  $c$  is a scaling factor.  $I(u, v, \alpha)$  represents the intensity pixel data of the 2D projection. When the image is converted to the intensity domain, the total attenuation value in the path of the photon propagation is represented as Real number and it can be calculated with the following equation:

$$g(u, v, \alpha) = \ln \left( \frac{I_0(u, v, \alpha)}{I(u, v, \alpha)} \right) \quad (2.3)$$

Where  $I_0$  represents the projected images without an object acquired during the scanner calibration. To summarize, this step has the aim to convert the raw data from attenuation to intensity domain and to remove scatter X-ray photons which are acquired also without the object.

2. **Cosine weighting:** It weights the data based on the pixel position [64]. In addition, this step considers the shape of the matrix detector, which is cylindrical in the KIDS-CT scanner. This specific shape results in a different attenuation/intensity weight for each projection pixel, which in the case of the KIDS-CT scanner is calculated by the equation 2.4:

$$g_c(u, v, \alpha) = \cos(u \cdot \Delta\beta) \cdot \frac{D}{\sqrt{D^2 + (v \cdot \Delta v)^2}} \cdot g(u, v, \alpha) \quad (2.4)$$

The value  $D$  is constant in the equation 2.4, representing the distance from the focal spot to the detector. In addition,  $g(u, v, \alpha)$  is the intensity image data, which results from the equation 2.3. The rest of the equation represents the angle between the iso-ray and the ray connecting source and detector point.

3. **Redundancy weighting:** In the redundancy weighting, the sum of the weights corresponding to the same line-integral must equal one [66]. To reduce the intensity  $g_c(u, v, \alpha)$  based on its weight, the following equation is used:

$$g_{c,r}(u, v, \alpha) = w_r(u, v, \alpha) g_c(u, v, \alpha) \quad (2.5)$$

In the equation 2.5,  $g_c(u, v, \alpha)$  is the cosine weighting of the projection, and the  $w_r$  is the weighted value of redundancy weighting. In the case of KIDS-CT (full

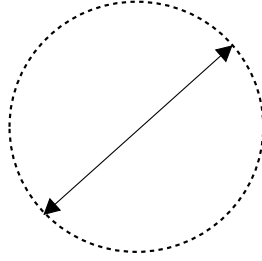


Figure 2.13: X-ray intersects the circular focal spot twice in full scanning

circular scanning), the X-ray flux intersects the circular focal spot twice, as shown in Fig. 2.13. Therefore, the redundancy weighting value will be for full scan [66]:

$$w_r(u, v, \alpha) = \frac{1}{2} \quad (2.6)$$

After the pre-processing phase, there is the filtering step, which is applied to each projection. In this step, there is a dependency between the nearest pixels, depending on the applied filter, but there is no dependency between projections. In the FDK, the **Ramp filtering** is used. It consists of a high-pass filter to reduce low-frequency noise.

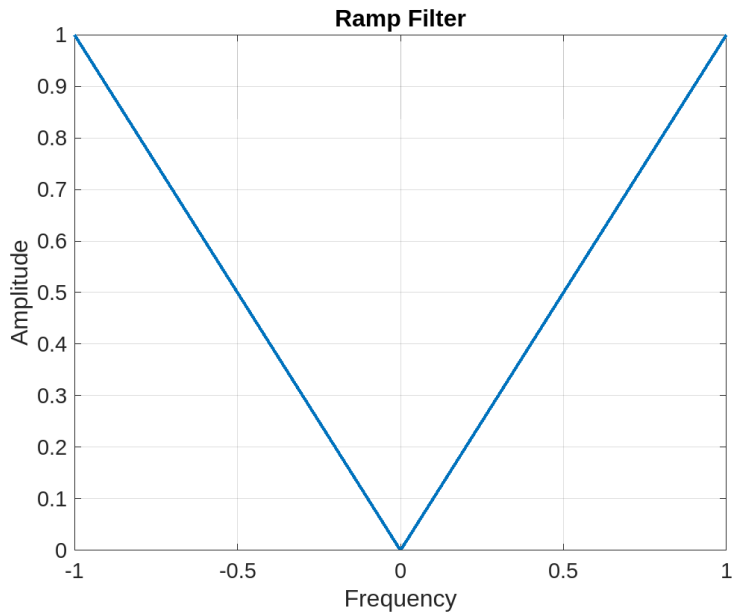


Figure 2.14: Illustration of the ramp filter  $h_{ramp}$  in the frequency domain

The filtering is performed on a row-by-row basis within the projection; A single row is taken from the projection and subjected to a 1D convolution using the filter kernel value. This 1D convolution is applied to all the projections. The results of ramp filtering include a sharper image and less blurring. The ramp filter equation can be expressed as follows:

$$g_{c,r,f}(u, v, \alpha) = g_c(u, v, \alpha) * h_{ramp}(u) \quad (2.7)$$

In the equation 2.7,  $h_{ramp}$  represents the ideal ramp filter shown in Fig. 2.14, and “\*” represents the convolution operator.

The **Back-projection** takes the pixel-filtered projection  $g_{c,r,f}(u, v, \alpha)$  into the image space of the VCS to obtain an approximate voxel  $\mu(x)$  according to the following equation:

$$\mu(x) = \int_{\alpha_0}^{\alpha_{N\alpha-1}} \frac{R_f}{\|x - f(\alpha)\|^2} g_{c,r,f}(u, v, \alpha) d\alpha \quad (2.8)$$

In the equation 2.8, the  $R_f$  is the source to iso-center distance,  $f(\alpha)$  is the focal spot position. Furthermore,  $\mu(x)$  is the linear attenuation coefficient, which represents the reconstructed voxel of the 3D image. The back-projection permits to calculate the total attenuation of each projection and the sum of all the attenuation values from the reconstructed image in volume geometry [64].

## 2.5 Design Space Exploration

CPSs pose significant challenges to embedded designers because they must meet stringent and conflicting requirements [67]. For instance, interventional CT scanners must acquire and process data in real time and provide high-quality images with a low X-ray dose. At the same time, CT scanners should be reliable, flexible, and capable of supporting multiple medical procedures [68]. On the one hand, MPSoC-FPGAs offer a heterogeneous architecture that permits addressing these challenges. On the other hand, the complexity and the increasing number of design options offered by these platforms can lead designers to make wrong decisions and fail with the final product [69]. For this reason, designers have defined methods to find optimal design options in the very early design stages, such as performing a Design Space Exploration (DSE) [70].

The concept of Design Space Exploration initially emerged in the context of logic synthesis [71]. In this area, designers playing with the constraints observed that it is possible to reduce the design costs by considering the delay-area trade-off in the design space. This process of systematically adjusting design parameters has been acknowledged as DSE [71]. It considers the various design options (e.g., mapping of application tasks either on a programmable core or a hardware block, communication infrastructure, and protocols). These design options increase with the number of tasks and the complexity of the hardware platform, which is generally considered an NP-hard problem [72].

To meet the real-time requirements and to reduce the design space, this thesis focuses on the design exploration of the task mapping problem and on data formats that enhance the image pre-processing phase. In the next section, the basics of data formats for number representation are explained; these concepts give the basis for the proposed exploration of the data format design space options.

### 2.5.1 Data formats for number representation

As explained above, CT sensor data are acquired in the attenuation domain, represented with positive natural numbers ( $\mathbb{N}^+$ ). In order to reconstruct the 3D volume, the data are converted into the intensity domain, represented with real numbers ( $\mathbb{R}$ ). In computer science, there are several data formats for representing real numbers, depending on the accuracy of the representation. The data formats typically utilized for real numbers are either *float* or *double*. These two formats refer to the Standard for Floating-Point Arithmetic (IEEE 754) [73]. This standard specifies arithmetic representations, conversions, and arithmetic operations. As shown in Fig. 2.15, three fields represent floating-point formats: sign (S), exponent (E), and mantissa (M) bits. Moreover, a value can be represented as a function of S, E, and M, as follows:

$$f(S, E, M) = \begin{cases} (-1)^S \cdot 2^{E+1-2^{e-1}} \cdot (1 + M \cdot 2^{-m}) & \text{for } 0 < E < 2^{e-1}, \\ (-1)^S \cdot (1 + M \cdot 2^{-m}) & \text{else} \end{cases}$$

In the above function,  $e$  and  $m$  represent the data width of the exponent and mantissa fields, respectively (e.g., in 32-bit floating point  $e$  is equal to 8 and  $m$  is equal to 23). According to the IEEE 754 standard, there are four different floating-point encoding formats with varying bit lengths: 16 bits for half-precision, 32 bits for single-precision, 64 bits for double-precision, and 128 bits for quad-precision. As illustrated in Fig. 2.15, S, E, and M are organized in the same order for all the encoding configurations, but they have different lengths, varying the subset size of the representable real number.

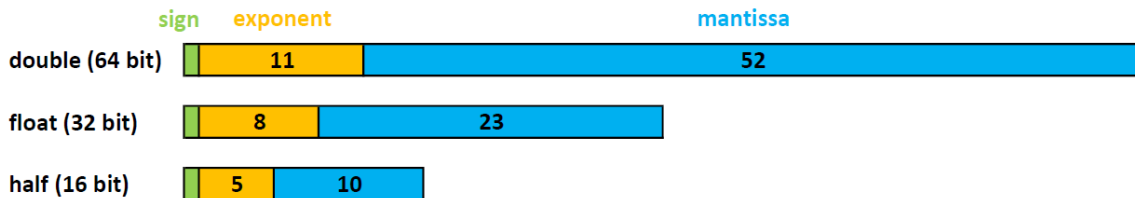


Figure 2.15: Encodings of the floating-point standard

Furthermore, the various encodings require different arithmetic processing units, having different performances in execution time, memory utilization, power consumption, and chip area.

For instance, 32-bit floating point encoding represents numbers within the range of  $2^{-149}$  to  $2^{128}$ , with a relative error of  $2^{-23}$  resulting from the truncation of digits.

In recent years, in order to meet stringent computational requirements, designers have begun to use other standard data formats, such as fixed point, and propose custom representations for performing operations on real numbers. Although these other formats can usually represent a smaller subset than IEEE 754, the accuracy of the results may be sufficient to meet the application requirements. For instance, the fixed-point representation

may perform better than IEEE 754, using the same data width. In fact, Arithmetic Logic Units (ALUs) for fixed-point operations are usually faster and uses fewer resources than IEEE 754 to perform the same operations [74].

The fixed-point representation [75, 76], as shown in Fig. 2.16, has the following two parts: *Integer (I)* and *Fractional (F)*.



Figure 2.16: Fixed-point representation

Different from IEEE 754, it has no standardized encoding for the various lengths. In fact, hardware designers represent the Integer part either with the signed format including the sign bit or with the unsigned format where sign bit is represented with an additional field, as shown in Fig. 2.17.



Figure 2.17: Fixed-point representation methods

In this work, the author represents the fixed-point representation with the first solution, where the format length or *data Width (W)* is equal to  $I_{\text{SIGNED}} + F_{\text{UNSIGNED}}$ . The data width of I and F will depend on the subset of  $\mathbb{R}$  to represent. For the results of this work, the two solutions are independent, so it has been chosen in according to the solution proposed by AMD-Xilinx for their High-Level Synthesis (HLS) libraries, where the sign bit is included in the I part [77].



## 3 Related Works

Besides this work, other architectures for MPSoC-FPGAs exist that focus on controlling components, acquiring and processing data within CPSs [27, 28, 78–85]. Comparing them reveals strengths and limitations that the proposed work aims to address. Therefore, this Chapter begins with an overview of system architectures for CPSs, addressing issues to provide a literature review where plug-and-play capability and real-time support have been pointed out. This highlights the importance of solving this problem at the CPS device level, where FPGAs and MPSoC-FPGAs are broadly used to realize CSs and DASs. Furthermore, related work also highlights the challenges of task mapping on MPSoC-FPGA platforms within CPS applications. Moreover, the crucial aspect of security in CPSs that have MCS requirements, such as the CT application, is explored in the context of MPSoC-FPGAs. Finally, related works that consider these issues in the specific case of the CT application are considered. Due to the fact that academic researchers usually focus on image processing problems, and CT scanner manufacturers do not provide these informations, only a few works have been found on the system architecture. This lack of information also limits the exploration of multimodality and interventional procedures in Academia, that an open-interface CT scanner such as the KIDS-CT aims to fulfill. Thus, system architectures of CT scanners have been examined by the author, pointing to the limits to provide real-time support and plug-and-play capability. In this context, a deep analysis of existing work on data formats for CT imaging is also presented. The content of this Chapter is covered by the articles in Ref. [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7].

### 3.1 System Architecture In CPSs

Several system architectures have been proposed in the literature for CPSs, including the communication infrastructure [78–81, 86]. These works mainly address architectures that model the interconnection of CPS devices over the network and include the cognitive layer where decisions are also made over the network. However, this thesis considers CPSs at the device level, following the definition of CPS given by A.E. Lee in Ref. [18]. Therefore, the focus of this work mainly addresses real-time support and plug-and-play capability within the CPS device [2, 3].

The National Institute of Standards and Technology (NIST) has established the Cyber-Physical Systems Public Working Group (CPS PWG) [3], which operates as an open public forum supporting designers. Its objective is to develop a generic framework for

CPS. This process begins by examining the three primary aspects of CPSs: conceptualization, realization, and assurance. Pertaining to these, the framework suggests activities to define various CPS models. In these stages, the framework highlights challenges in interoperability and extensibility among different CPS components. These challenges stem from the incompatibility of various protocols used, which are crucial for enabling plug-and-play capabilities. In this context, the Medical Device Plug-and-Play (MDPnP) initiative has been promoted. It seeks to extend plug-and-play capabilities to medical devices by developing new models, architectures, and standards [87–89].

The Open Platform Communications (OPC) Foundation has proposed Open Platform Communications United Architecture (OPC UA), which is a service-oriented architecture that integrates the OPC standard, emphasizing open and interoperable Machine to Machine (M2M) communications [90]. Based on the OPC UA and the OPC standard, different architectures for CPS devices have also been proposed [4, 91–93]. Between them, García et al. [4] presented a Cyber-Physical Production System architecture that aims to facilitate the collection of data related to “plant floor processes” and the transmission of commands to control devices in the targeted application. For this purpose, they propose an OPC UA server for Modbus/TCP industrial networks. Although, OPC UA offers the possibility to integrate different components in a CPS, allowing extensibility and interoperability in the CPS architecture, it is limited to legacy protocols and can not be implemented in small microcontrollers due to its memory requirements [94]. Also Graube et al. [94] highlight these limitations, which are critical in small CPS components such as sensors/actuators.

In a systematic mapping study, Hofer [2] also identifies component interoperability and system extensibility as common challenges of CPS architectures for Industry 4.0. The challenges of interoperability and extensibility arise from the need to integrate different components within a CPS, such as sensors, actuators, and communication architectures, which often use different protocols and technologies. However, legacy systems and the use of components from different vendors make it difficult to design an architecture that fits all possible application scenarios.

All the aforementioned works proposing frameworks and architectures for CPS have mainly pointed out the challenges and solutions focusing on communication standards, without considering CPSs where sensors/actuators with custom standards and custom architectures without OS are involved, such as the targeted CT applications. Instead, Hofer [2] has addressed these challenges at the device level, where several solutions have also been proposed [27, 28, 95] and they will be discussed in Section 3.2. These works show that besides the different domains and their applications to provide interoperability and

extensibility, a major challenge should be solved at the device level: designing control units that provide plug-and-play capability for vendor components using custom and standard protocols.

### 3.2 Control And Data Acquisition Systems

As mentioned above, CSs and DASs are responsible for controlling components and acquiring/collecting data within a CPS device. Several CS and DAS architectures for MPSoC-FPGAs and FPGAs have been proposed in literature [82, 85, 86, 96–102].

Marjanovic et al. [96] presented an integrated DAS for MRI, based on an MPSoC-FPGA. The architecture is divided into modules that acquire and process data. Each module is written in VHSIC Hardware Description Language (VHDL) and is interconnected via the AXI4 memory-mapped bus interface. The *High-Speed Data Link* modules acquire and merge the data and then process them using a streaming interface. The acquisition protocol is based on the Xilinx Aurora IP core [103]. The bandwidth and the reconfigurability of the architecture are not specified by the authors.

Yang and Chen [97] present a high-speed embedded platform for image acquisition and processing. They combine FPGA and Digital Signal Processor (DSP) for image acquisition and processing tasks and use an ARMv9 processor to communicate with the host. It achieves a data rate of 36 Mbps. They have a main Finite State Machine (FSM) that controls the system with a non-pipelined datapath. This solution is only for fixed image data processing, and the implementation is not parametrizable or configurable for other use cases or application domains. Like Yang and Chen [97], Shi and Zhang [98] implement a dual-channel image acquisition system based on an FPGA. This system acquires images from multiple sensor cameras. It displays the images via an HDMI interface with a maximum bandwidth of 2.25 Gbps and simultaneously stores them in DDR3 SDRAM.

Xie et al. [99] present an FPGA implementation of a high-speed data acquisition system for high-resolution millimeter-wave radar. They achieve 7200 Mbps from the radar interface. The acquired data are stored in DDR3 SRAM and transmitted off-line to the external PC. The acquisition data capacity is 48 MB per event. This system does not support real-time processing, and it has a custom architecture that can only be used for this specific application. Flouzat et al. [100] introduce an acquisition and processing platform for automotive applications. It is an FPGA-based solution optimized for high bandwidth and low latency. It acquires data from 8 cameras in parallel and processes and stores them via PCI-E, with a throughput of up to 3.2 Gbps.

Salgaro et al. [101] proposed a DAS for PET. This CPS supports a high-speed commu-

nication protocol for a network of various DASs implemented on FPGAs. The authors propose a Ring-Chain architecture connecting multiple FPGAs in a daisy chain using Gigabit Transceivers. This architecture ensures data integrity, manages line congestion, and allows for high-speed data transfer of up to 250 million measures per second. However, the disadvantage of the distributed solution lies in the complexity of synchronizing multiple FPGA and effectively managing line congestion, which is crucial for the accurate functioning of the system.

Min et al. [82] proposed a low-cost C-shaped PET, where the DAS architecture has been designed and implemented on an FPGA. In this CPS shown in Fig. 3.1, the DMS can be configured before each scan to acquire and eventually process data before sending them to the host PC. The acquisition system and the host PC communicate with a USB connection which is the bottleneck of the system. Therefore, it can only be used for applications that reconstruct the image offline.

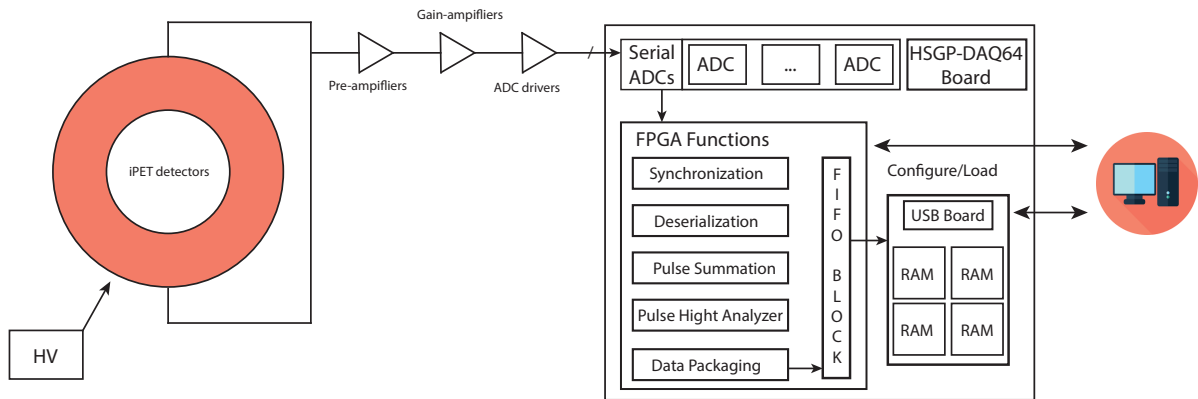


Figure 3.1: Low-cost C-shaped PET architecture

Furthermore, the system is optimized for PET, and can not be configured for additional sensors or different medical procedures. Since all tasks are hard-wired in custom hardware modules and the DAS lacks a software stack, the system can not be extended.

Traxler et al. [86] proposed a custom DAS for FPGAs targeting the PET application. They integrated this architecture into a real-time Jagiellonian-PET (J-PET) scanner, with three DAS units, each achieving a speed of 3.2 Gbit/sec. As part of the design of this J-PET scanner, the authors also proposed a Data Processing System (DPS) for MPSoC-FPGA platforms, allowing the real-time processing of collected data on a single chip that includes an FPGA, a CPU, and a GPU [83, 84]. This architecture consists of eight parallel pipelines that acquire data. Within the proposed processing system, data are decoupled, processed, and then displayed to the doctor. The different architectures and detectors are coordinated by a CS using a configuration set sent by the host PC. However, the DAS and the processing system are only reprogrammable for different J-PET

configurations and cannot be adapted for other CPS applications or to extend the J-PET scanners for multimodality procedures.

A generic DAS architecture for nuclear medical imaging applications has been proposed by Fysikopoulos et al. [85]. It has been implemented on a AMD-Xilinx Spartan-6 FPGA and connected to 12-bit octal-channel high-speed Analog-to-Digital Converters (ADCs) for acquiring data from detectors, as shown in Fig. 3.2.

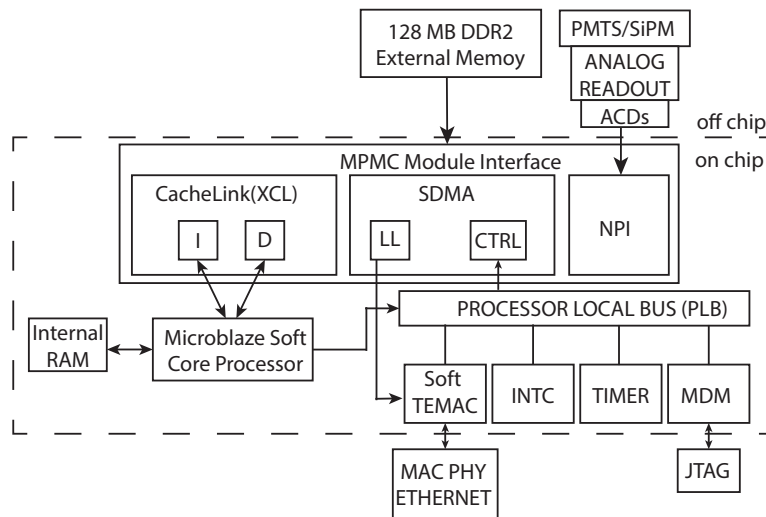


Figure 3.2: Configurable DAS architecture from Fysikopoulos et al. [85]

The DAS has been designed to be configured for different detectors. Due to the limited FPGA resources, only the calculation of coordinates corresponding to the position of an event is performed on the DAS, while the other data processing is performing externally. The DAS stores the data on the main memory during the acquisition, and sends them to a PC offline; the various control tasks are implemented on a Microblaze soft-core processor [104]. In order to communicate, this DAS uses an Ethernet connection with User Datagram Protocol (UDP) and a custom datagram protocol for data acquisition; the implemented system reaches a data rate of up to 60 Mbits/s. Although this architecture is configurable for various applications and detectors, it is not extensible in a plug-and-play fashion yet, and it does not provide real-time support.

From industry, Eltec has proposed a DAS, commonly referred to as a frame grabber, designed for utilization with cameras and Computed Tomography (CT) scanners. This system relies on FPGAs for the efficient processing of data [102]. Specifically, the model designated for CT applications is the PCEY-0600 PC\_EYE frame grabber. This particular model is characterized by its integration of multiple FPGAs on a singular circuit board. Since it can not send data to the reconstruction system in real time, it uses Dynamic Random Access Memory (DRAM) for data buffering purposes. The operational protocol

involves initially storing the acquired data on the DAS, followed by a subsequent offline transfer to the Reconstruction System post-acquisition.

In the context of Advanced Driver-Assistance Systems, Tesla introduced a FSD hardware architecture [28]. This is an example of dedicated centralized CDAS architecture for CPSs with a dedicated hardware architecture.

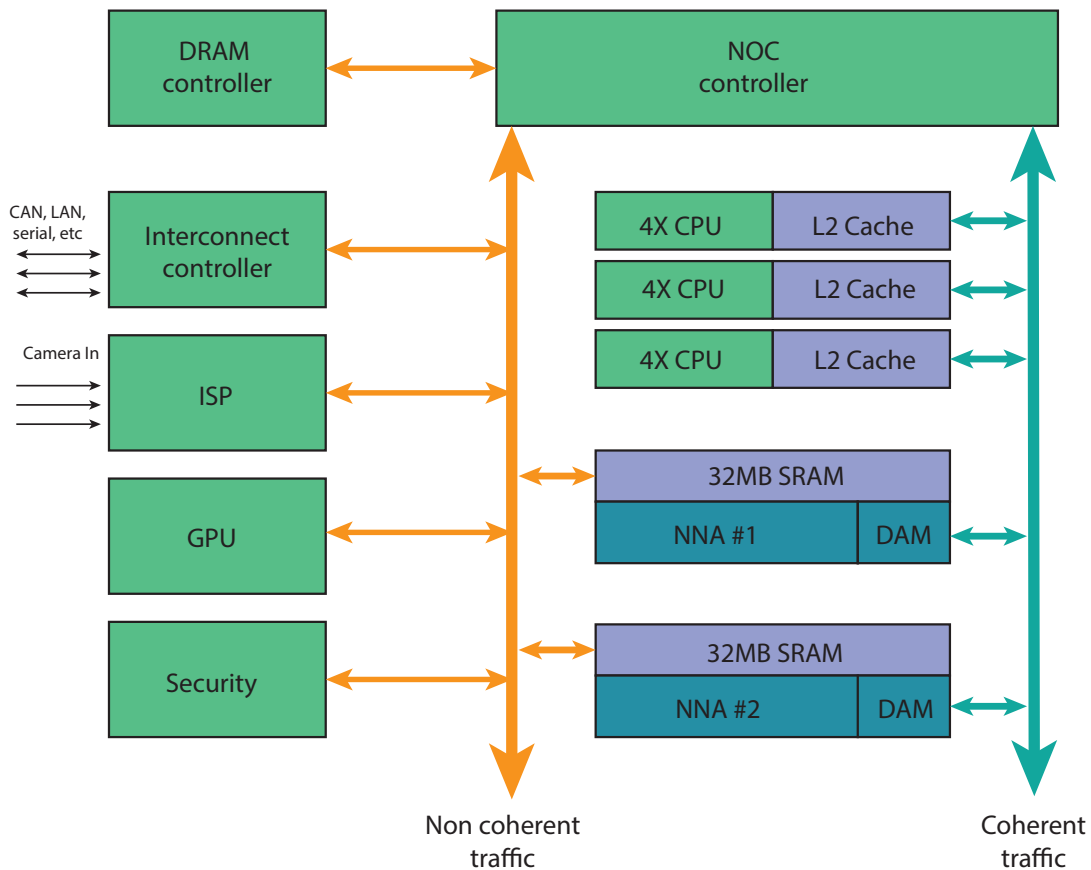


Figure 3.3: Tesla FSD: Block diagram architecture

As shown in the Block Diagram in Fig. 3.3, the camera data are collected by the ISP that pre-processes and stores them in main memory every few milliseconds. When data are ready to be processed the CPU activates the Neural-Network Accelerator (NNA) that processes them to detect objects such as lane lines, cars, pedestrians et cetera. In the case of post-processing algorithms, the GPU is used. All components and algorithms are coordinated by a CPU which also runs the autopilot algorithm. The architecture proposed by Tesla is designed for ASIC. Although the FSD proposed by Tesla acquires/collects data from different [27] sensors/actuators and processes them in real time, this architecture is not extensible for adding components in a plug-and-play fashion.

### 3.3 Task And Peripheral Isolation

In CPSs, such as the KIDS-CT scanner, which must meet MCS requirements, tasks associated with different applications are run within isolated environments. This isolation, based on criticality level and/or application domain, helps to prevent interactions between tasks that could affect timing and dependability issues.

Various approaches have been proposed in the literature for providing isolation within high-performance computing MPSoC-FPGAs [48, 105–116].

#### 3.3.1 Isolation in AMD-Xilinx architectures

AMD-Xilinx has proposed several isolation mechanisms for their high-performance MPSoC-FPGA platforms, such as the Ultrascale+ and the Versal architectures [117]. Due to the complexity of these architectures and the large number of attacks that need to be mitigated, they consider the Swiss cheese model, shown in Fig. 3.4, which has multiple layers with holes through which attacks can pass individually, but the same attacks cannot pass all layers if they are stacked [117].

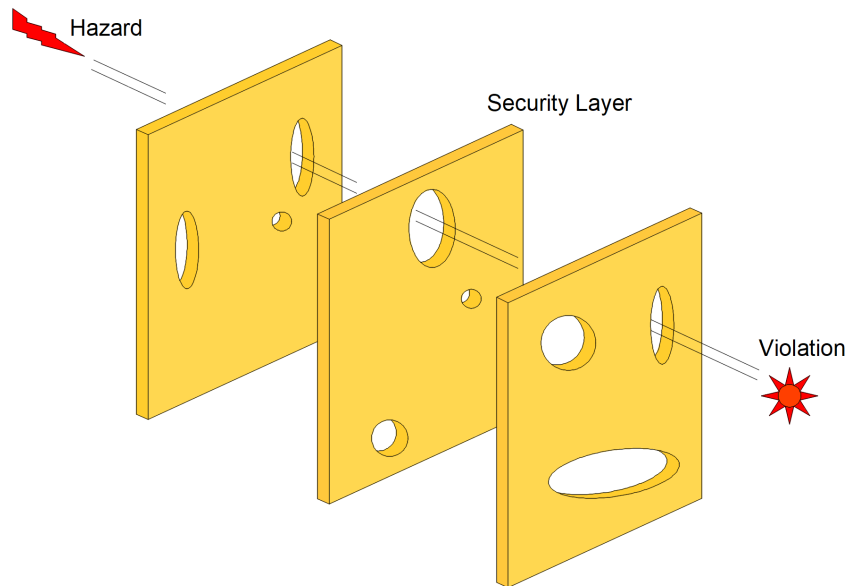


Figure 3.4: Swiss cheese model

Applying this model to provide isolation on their architectures, AMD-Xilinx proposes the following solutions: Trust Zone, AXI isolation blocks (AIB), Platform Management Unit (PMU), System Memory Management Unit (SMMU), Xilinx Memory Protection Unit (XMPU) and Xilinx Peripheral Protection Unit (XPPU) [106], which are mainly not supported by the low-cost MPSoC-FPGAs targeted by this thesis, such as the Zynq-7000 architecture [118]. Here, the authors have focused on the description of the TrustZone and the XMPU/XPPU solutions, which will be compared with the proposed isolation mechanism in part IV.

## TrustZone

TrustZone is a security feature within the Arm architecture that divides the system into secure and non-secure worlds, affecting both software and hardware aspects. While tasks running in the secure world have access to all memory regions and peripherals, tasks running in the non-secure world are restricted to their own domain. This dual structure is realized by two separate processor states, one for each world.

Transactions are marked with the current processor state using the AxPROT[1] bit of the AXI interface [36]. In addition, compliant IP cores such as the Xilinx AXI Interconnect [119] can be configured to grant access only to transactions associated with the secure world. Although widely adopted in systems with Arm processors, this extension primarily manages only two asymmetric domains [105]. Furthermore, TrustZone technology is not limited to AMD-Xilinx architectures, but it is also implemented in platforms with ARM processors such as the Intel Cyclone V [120].

## XMPU/XPPU

Xilinx Memory Protection Unit (XMPU) and Xilinx Peripheral Protection Unit (XPPU) consist of PUs architecture. Unlike traditional PUs that are integrated into the processor, these units are placed in front of the DDR memory controller and the On-Chip-Memory (OCM), as well as in front of the peripherals [48].

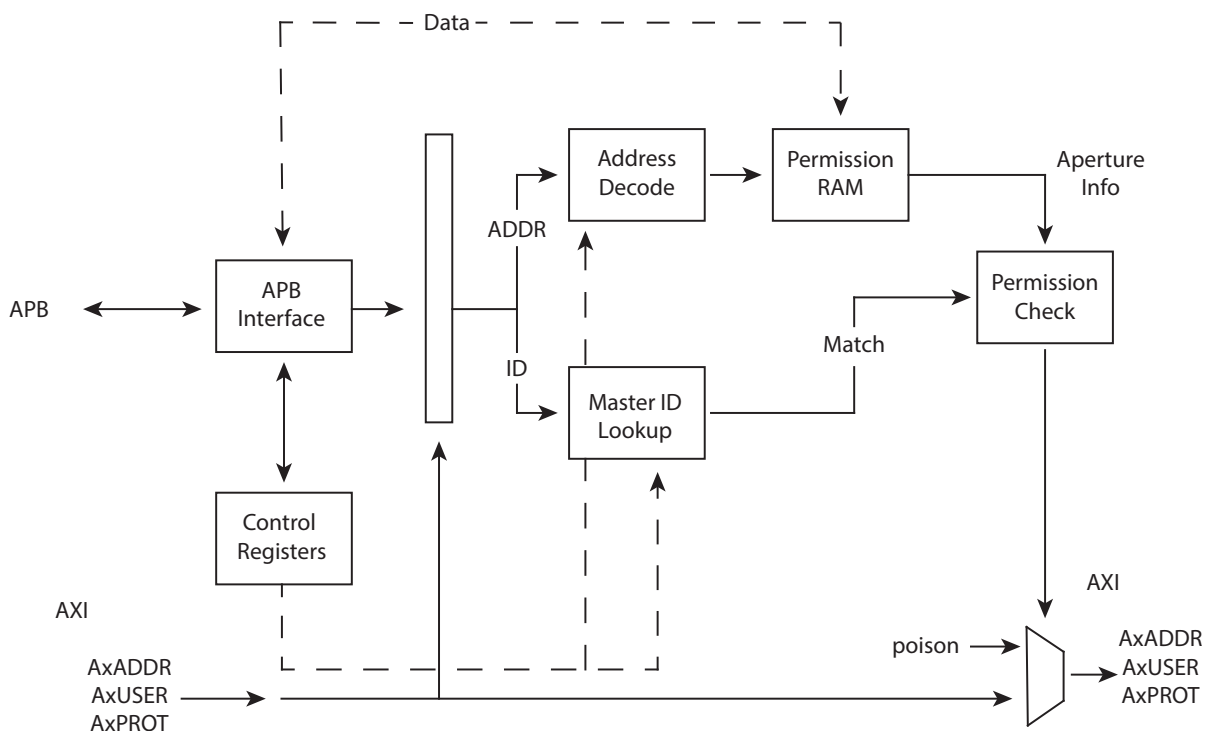


Figure 3.5: XPPU functional diagram

Both units limit the access of certain 'masters' to certain memory regions, thus pro-



viding isolation for masters/slaves using PS-PL communication. However, they do not provide isolation for master/slave components that are both implemented in the PL and communicate via a PL-PL bus. In addition, they are only supported by high-performance MPSoC-FPGAs produced by AMD-Xilinx. In order to block AXI transactions, they act as pass-through components, implementing an AXI slave interface to receive incoming transactions and a master interface to forward them; for run-time configuration, they also have an Advanced Peripheral Bus (APB) interface [121], as shown in Fig. 3.5. Since peripherals use the same memory address space (i.e., memory-mapped Input/Output (I/O)) the XMPU and the XPPU have the same functionality and similar architecture, but the former is optimized for a larger memory address space than the latter, and the XMPU for peripherals with finer-grained address spaces than XMPU. Therefore, the rest of this Section will refer to both architectures as a single architecture called XPPU/XMPU.

When an AXI transaction attempts to cross the XPPU/XMPU, it checks the master ID and its address. If the address is authorized for this master, the transaction is granted and forwarded unchanged. If the transaction is denied, it is acknowledged with an error. This error can be caused by address or attribute poisoning. Address poisoning involves altering the transaction address to redirect it to an error-generating slave. In attribute poisoning, a flag is set in the USER-signals, which is detected by a downstream component responsible for generating error messages.

### 3.3.2 Protection units solutions

Several PU solutions have been proposed in the literature to provide isolation [107–114]. A taxonomy can also be found in [115], and a formal model that describes configurations is presented in [116]. Among them, this thesis focuses on the description of the isolation solutions suitable for low-cost MPSoC-FPGAs that will also be compared in part IV with the Lightweight Protection Unit (LPU) proposed in this research work.

However, none of these works combines the flexibility with the optimizations of the LPU that make it the best option for low-cost MPSoC-FPGAs. Nor do any of these works consider re-routing PS traffic through the PL to provide system-wide PU capabilities.

#### Network-on-chip firewall

LeMay et al. [112] introduce an isolation mechanism for AXI-based NoC called Network-on-Chip Firewall (NoCF). The NoCFs are instantiated at the connections between masters and the NoC interface. These firewalls operate based on a policy defined by allowed address ranges for both read and write transactions. To optimize performance, dedicated NoCF are instantiated in the design, handling read and write transactions. The key ideas of their solution is based on the Policy Decision Point shown in Fig. 3.6. It is implemented by a dedicated isolated soft microprocessor core that is configured at run time.

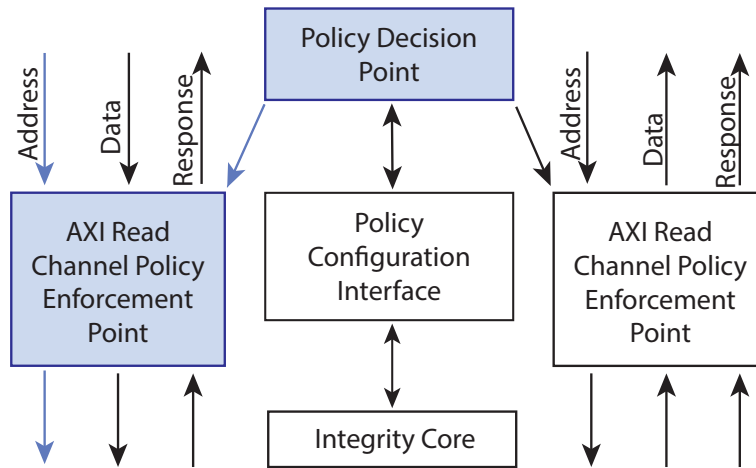


Figure 3.6: Network-on-Chip Firewall

### MPPU instance example

Kornaro et al. [113] propose an isolation mechanism called Memory Partition Protection Unit (MPPU). As shown in Fig. 3.7, the MPPU is implemented in the PL part and connected to the PS that control this. It is based on AXI and exploits the transactions ID and ADDR signals to identify the application domain and the memory region to access. In order to do it, a look-up table with the permission is implemented in the global global memory. This table contains information for read/write, data/instruction, secure/non-secure, and privileged/unprivileged access. This solution is suitable for low-cost

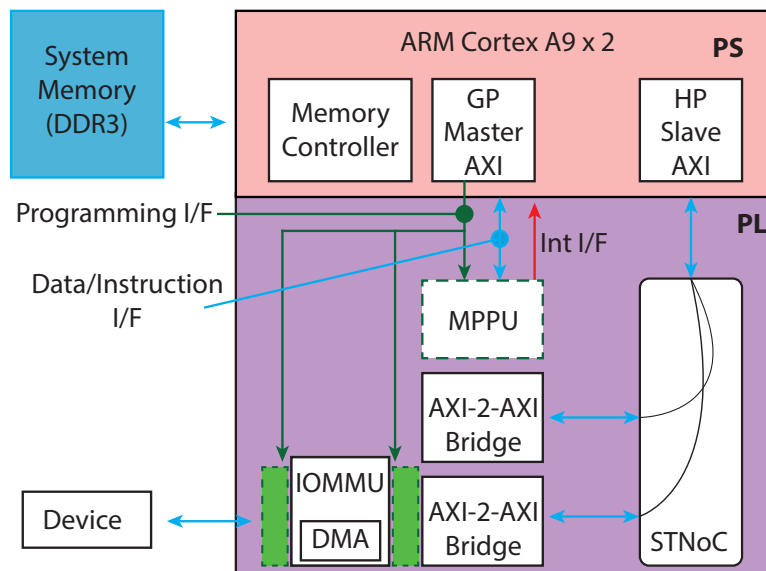


Figure 3.7: MPPU: example of deployment [113]

MPSoC-FPGA, but it is not efficient in terms of timing because policies are written in the global memory. Due to that, it is not suitable for real-time application because the access time to policy is not deterministic.

### Hardware/software IP management modules

Saha et al. [114] present an approach in which context information is added to AXI transaction. To do so, they propose a hardware/software architecture, where the Hardware IP management module (SIMM), running on the PS is responsible for updating the policy and managing them, while the Hardware IP management module (HIMM) checks the AXI transactions.

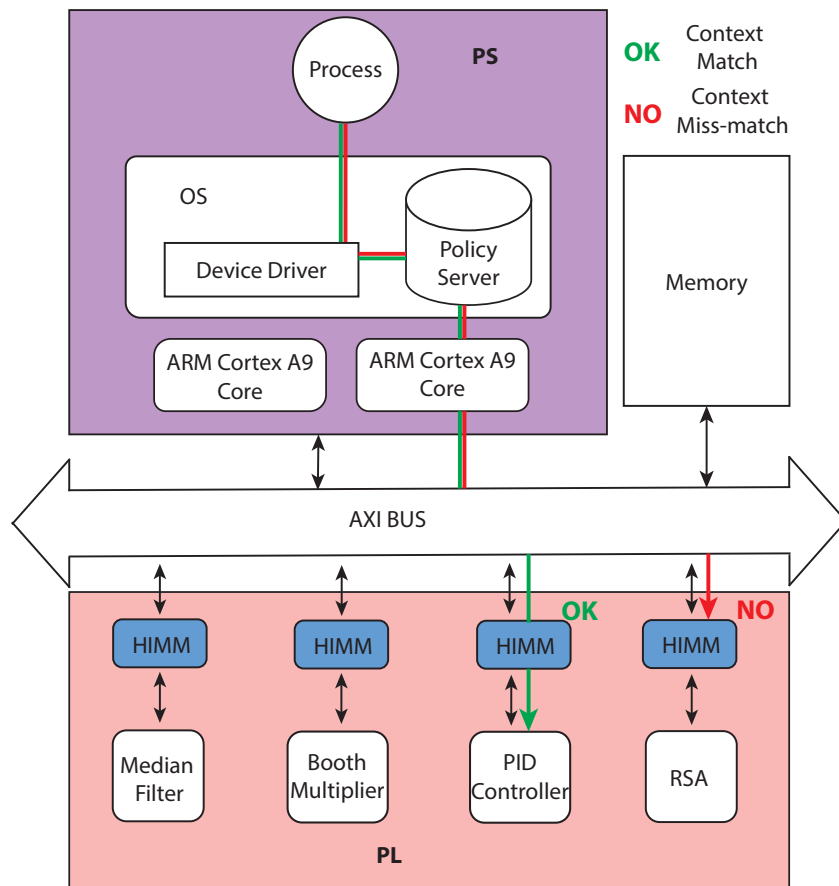


Figure 3.8: Hardware/Software IP management modules overview

Each slave has a HIMM in front that checks all the crossing transactions; it receives the context information from the SIMM that stores them in a Policy server. An overview of the resulting architecture for this approach is shown in Fig. 3.8, which shows granted and denied transactions related to two slaves. Furthermore, in this solution, the HIMM requests the policies to the SIMM when they are not in the HIMM cache. Due to the high deviation of timing access to the Policy server, this solution is also not suitable for real-time applications.

### 3.4 Computed Tomography

This Section analyzes related works on the targeted CT application. It mainly focuses on design solutions for CT scanners at the device level, considering the system architecture, the communication infrastructure, the task partitioning, and how components are controlled and data acquired in the existing solutions. Since, commercial scanners are closed systems, where it is not possible to know these details, only a few existing architectural models could be analyzed for the proposed solution. The lack of existing works and scanners built in Academia also pushed the author to propose a guideline for hardware designers that has been the starting point for the proposed architecture of the KIDS-CT scanner [DP 3].

#### 3.4.1 Controlling and data acquisition systems for CT scanners

Altera-Intel has proposed a System Architecture for CT scanners [122]. This architecture outlines which MPSoC-FPGA and/or FPGA are suitable for controlling internal components, as well as for collecting and processing data. They suggest a solution, as shown in Fig. 3.9, where the Control System, Data Acquisition System, and Data Processing System are designed as independent architectures. These are to be implemented on different cards (i.e., components) and platforms. In their CT scanner architecture, four key components are proposed: the *Data Acquisition Card*, the *Data Consolidation Card*, the *Data Processing Card*, and the *Control Card*.

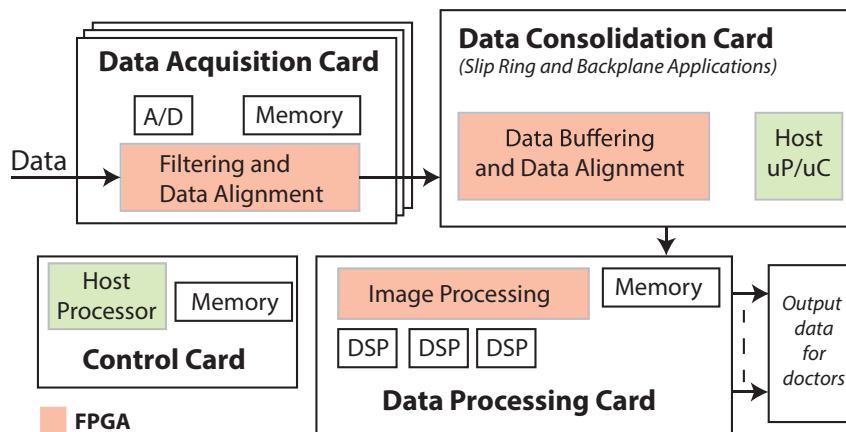


Figure 3.9: Altera-Intel CT scanner architecture

The Data Acquisition Card is responsible for acquiring analog signals from sensors, converting them, and sending them to the Data Consolidation Card. As shown in Fig. 3.9, there is typically more than one *Data Acquisition Card* in a CT scanner, and they are directly connected to the matrix of pixels inside the DMS. For instance, in the DMS mounted in the KIDS-CT scanner, signals are converted and sent out. However, unlike the Altera-Intel solution, no pixel processing (e.g., filtering) is performed in the DMS of the KIDS-CT scanner. As Data Acquisition Card, they suggest the use of FPGAs with a huge

number of Analog (A)/Digital (D) converters and DSPs for filtering such as platforms belonging to the Cyclone/Arria family [123, 124].

The Data Consolidation Card collects data from the Data Acquisition Cards, buffers, aligns, and merges them before forwarding it to the *Data Processing Card*. For this stage, it is suggested to utilize an MPSoC-FPGA platform from the Stratix family [125]; this allows for efficient management and combination of various control tasks and data transmissions. Regarding data processing, the implementation of an FPGA accelerator is recommended, as represented in Fig. 3.9 by the Data Processing Card. This accelerator should possess several features: numerous DSPs, Block RAM (BRAM), and DRAM memory, capable of reaching over 10,000 Giga Floating-point Operations Per Seconds (GFLOPSs) [126]. Such a configuration yields performance comparable to modern GPUs.

In the context of medical image processing and image-guided surgery, Altera-Intel also proposes a design framework featuring 18 video functions based on a streaming interface [127]. This approach allows designers to implement software functions in hardware, facilitating the exploration of architectural challenges related to memory and optimization of single functional units, such as Convolution and Fast Fourier Transform (FFT) architectures, resulting a suitable solution for implementing a CT reconstruction algorithm in the Data Processing Card. Finally, all the described cards are controlled and coordinated by the Control Card which can be implemented on a CPU architecture. This solution does not consider the problem of custom control protocols involved in this application, which can only be implemented using custom ASICs or FPGAs.

Another CT scanner architecture shown in Fig.3.10 has been proposed by AMD-Xilinx [128]. Here, the control and data acquisition systems have also been implemented over four sub-systems: the *HV Supply Control*, the *Data Acquisition & Gantry Control*, the *Image Reconstruction* and the *System Sequencer*.

The HV Supply Control system manages all tasks inside the X-ray tube: software parameters, high voltage power supply, and errors. AMD-Xilinx recommends using a Zynq Ultrascale+, which is an MPSoC-FPGA designed for high-performance computing [129]. This system can implement isolated modules in the PL for HV tasks, utilizing the XMPU/XPPU, and run the related software on the PS part. The Data Acquisition & Gantry Control system controls and coordinates the Gantry and the DMS while data are being acquired and forwarded for processing. For this system, the use of an MPSoC-FPGA from the 7series family is also recommended for implementing various tasks. The Image Reconstruction system and the System Sequencer respectively implement the reconstruction algorithm and synchronization tasks within the CT scanner architecture. For the reconstruction algorithm, AMD-Xilinx proposes to use an adaptive System-on-Chip (SoC) from the Versal family [130], a new heterogeneous architecture that includes FPGA, CPU, and DSP or Artificial Intelligence (AI) engine architectures for data processing.

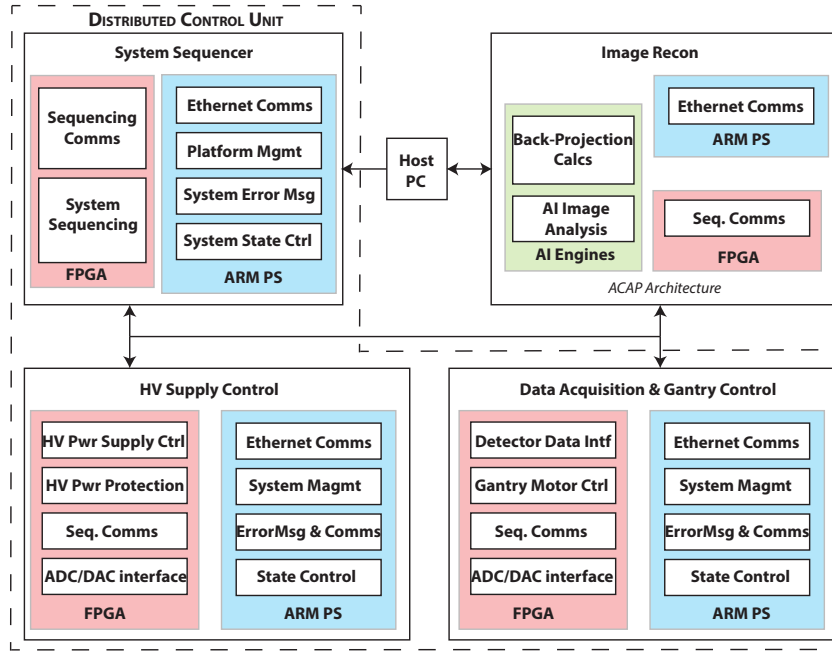


Figure 3.10: AMD-Xilinx CT scanner architecture

Both CT solutions analyzed above exemplify a distributed architecture that allows for the separation of control and processing tasks associated with different critical applications, thus preventing interference. However, they lack scalability, and components cannot be integrated in a plug-and-play fashion. In fact, to integrate a new component (e.g., a detector or an X-ray tube), tasks must be subdivided into subtasks and distributed across the various control units. In addition, they are designed for diagnostic procedures where real-time controlling and data processing are not requirements. In fact, a distributed architecture introduces additional delay in coordinating and synchronizing components due to the dynamic cross-interactions that a centralized system solves, as also experienced in automotive applications [131].

### 3.4.2 The data format exploration in CT data processing

In literature, various algorithms for CT image processing and reconstruction are discussed, targeting FPGAs[132–138] and GPUs [139–143] platforms, and exploiting their high level of flexibility and parallelism. This thesis does not aim to propose a new algorithm or hardware accelerator for CT image processing and reconstruction. Instead, its primary goal is to provide a CDAS where a selected algorithm can be integrated into a dataflow architecture, enabling on-the-fly data processing. In this context, this research aims to find the best data format for interventional CT applications. Therefore, related works that propose dataflow architecture for MPSoC-FPGAs and exploit mixed-data formats on GPU and FPGA are mainly considered for the comparison.

Dandekar et al. [144] proposed a reconfigurable architecture for real-time pre-processing in interventional CT. It consists of a dataflow architecture for FPGA that aims to optimize latency. They focused on the filtering steps for neighboring voxels. By exploiting the spatial locality of the data, they proposed a solution using a custom brick-caching schema to improve memory performance. They described their architecture in VHDL and explored different fixed-point formats for their solution: 8, 12, and 16 bits. Through a custom implementation of the proposed optimizations, they achieved a processing rate of 46 frames per second for images with dimensions of  $256 \times 256 \times 64$  voxels.

Nourazar and Goossens [145] developed an iterative CT reconstruction algorithm specifically optimized for the Tensor Cores in NVidia GPUs [146]. To boost performance by exploiting Tensor Cores, they, first, optimized the index ordering of the system matrix, which is a sparse matrix that describes the pixel-driven projections. Then, they used a mixed-precision computation approach in the reconstruction algorithm. The accuracy resulting from the mixed-precision computation was found to be almost equivalent to that of 32-bit floating-point computation [145]. Specifically, mixed-precision computation involves using different data formats within the reconstruction algorithm to represent the same real value.

Clemens Maaß et al. in [147] have investigated the impact of data formats on CT reconstructed images. They explored various encodings within the IEEE-754 standard, demonstrating that utilizing a 16-bit floating point improves the execution time in CT image reconstruction without compromising image quality [147]. In fact, the 16-bit floating-point reads and writes in memory half of the data compared to the 32-bit floating point. Therefore, it improves the throughput and the access to memory itself which is crucial in this application. This is particularly beneficial during the back-projection phase of CT image processing, which frequently accesses external memory [147].

Unlike related works, this research work is the first to perform a systematic DSE, where data formats can be tuned in the image processing steps of a selected CT reconstruction algorithm, to identify the optimal data format for interventional CT procedures. While other studies also investigating data format in this application have been limited to evaluating reconstructed images in terms of Mean Square Error (MSE), this work performs a hardware performance evaluation and a comprehensive quality analysis for each point in the design space, including metrics such as contrast and uniformity of the reconstructed image. This approach is crucial in assessing the impact of data formats on the accurate identification of tumors by surgeons.





**Part II**  
**Concept**



## 4 Problem Analysis

The KIDS-CT is the first open-interface scanner assembled in Academia with the aim of exploring multimodality and interventional procedures. The existing CDASs for CPSs described in Chapter 3 can not be used for the scopes of such a complex system. They do not provide extensibility capability (i.e., plug-and-play) and real-time support, which are essentials for the aforementioned medical procedures for which the KIDS-CT scanner is designed. This Chapter analyzes the limits given by the related work and discusses the research questions that arise from it. Finally, it presents the research objectives to address them in complex CPSs such as the KIDS-CT scanner. The content of this Chapter has been published by the author in Ref. [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7, DP 8].

### 4.1 Weakness Of The Current Architectures

Chapter 3 presented different solutions addressing real-time support, as well as extensibility and interoperability issues. This includes considerations for implementing CPS architectures in conjunction with the related CS and DAS. Related work particularly focuses on solutions for automotive and medical devices, which are CPS applications that have common requirements with the targeted CT application. By analyzing the related work, the following weaknesses have been identified in the current architectures.

1. It is observed that the FSD architecture [28], which is the only centralized system, is also the only one able to control components, acquire and process data, and provide real-time support for events with a one-millisecond response time. This centralized approach contrasts with other systems where these tasks are distributed across separate CS and DAS which do not offer similar real-time support. In addition, the distributed solutions are limited by additional communication and synchronization overhead, which negatively impacts the time efficiency of performing various tasks.
2. Although the Tesla solution provides real-time support, the proposed architecture is implemented on an ASIC, which limits the ability to add components. In fact, solutions targeting ASIC [28] or FPGA [82] platforms result in a less flexible architecture compared to those designed for MPSoC-FPGAs [27, 85]. Although FPGAs can be reprogrammed to extend the architecture for new devices, they do not natively provide a CPU with an OS and a software stack that can facilitate the control and integration of new vendor components. For example, many vendor components,

such as the X-ray tube in the KIDS-CT scanner, require a software stack with Unix libraries to communicate with them. In addition, MPSoC-FPGAs can implement architectures where control and data processing tasks can be optimized by strategically mapping them to PS and PL parts.

3. Several solutions have been proposed in the literature to guarantee temporal and spatial isolation, but they target only high-performance MPSoC-FPGAs. Although some of these solutions can be implemented in low-cost MPSoC-FPGAs, they are not suitable for real-time applications such as CPSs with MCS requirements. By requiring external memory, these solutions penalize the WCET estimation. Therefore, to ensure isolation in CPSs such as MCSs, it is necessary to find a solution suitable for real-time applications that can be implemented on a low-cost MPSoC-FPGAs.
4. DAS architectures proposed in the related work can not acquire and process data in real time, therefore data are stored in an intermediate memory (e.g., main or external memory), during the acquisition. In order to process them, data are sent to an external DPS only after the acquisition is completed. The limits of these solutions arise from the communication limits and how these architectures handle the collected data in the architecture itself.
5. Pixel processing and reconstruction algorithms are primarily implemented using 32-bit floating-point formats. Maas et al. [147] observed that by employing the half-precision floating-point data format, execution time improves while maintaining image quality. Although this initial investigation yielded positive results, the impact of different data formats on image quality metrics has not yet been fully explored. Furthermore, studies have been limited to custom solutions and have not considered the data format as a variable in the design space. This overlooks the potential to identify the optimal data format in terms of balancing image quality and performance.
6. The CT scanner architectures in the related work use a distributed solution, where tasks are associated with diverse CSs, DASs, and DPSs. These solutions have been designed for dealing with closed commercial systems that address diagnostic procedures. For instance, to explore multimodality procedures and, therefore, to add a new component such as a different detector in a CT scanner, the different tasks should be appropriately partitioned among the distributed CSs, DASs and DPSs. Dividing tasks among these systems requires an update of all of them, which can be a really complex process; a distributed solution may also determine synchronization and timing issues to meet the real-time requirements. Consequently, these closed systems are not suitable for exploring multimodality techniques and interventional procedures.

## 4.2 Research Questions & Objectives

Weaknesses in items [1-4] presented in the previous Section are related to the CPS architectures. Weaknesses in items [5-6] are related to multimodality techniques and interventional procedures and are combined with the KIDS-CT scopes. Different aspects should be considered for complex CPSs such as the KIDS-CT scanner due to the following requirements:

- Real-time data acquisition and processing of huge amounts of data.
- Real-time control and synchronization of device components.
- Extensibility of the CPS device for new components (i.e., plug-and-play capability).
- Task isolation between different critical applications running on the same CPS device.

Studies in Section 3.2 of CPSs in automotive applications have shown that a centralized approach is more efficient than a distributed one in meeting real-time requirements. However, these works did not consider the interoperability problem between components and, therefore, the extensibility of the whole system. Moreover, the various solutions did not exploit MPSoC-FPGAs platforms, which are heterogeneous architectures that can be reprogrammable at setup and run times. Based on the analysis of the weakness found in related works, this thesis aims to reach a main research object:

*"Find out how CS and DAS architectures can be combined into a CDAS architecture for MPSoC-FPGAs in order to provide real-time support and plug-and-play capability in complex CPSs such as the KIDS-CT scanner"*

In relation to this research objective, the questions addressed in this thesis are:

- How can a CS and a DAS be combined into a CDAS architecture for MPSoC-FPGAs targeting CPS applications such as the KIDS-CT scanner?
- How can CDAS tasks be partitioned on MPSoC-FPGAs to provide real-time support and plug-and-play capability?
- How can a CDAS architecture be designed for an MPSoC-FPGA to acquire and process data on the fly?
- Which data formats optimize computing performance and hardware cost of the CT pre-processing steps while keeping the image quality suitable for iCT?
- How can task isolation be achieved in low-cost MPSoC-FPGAs while adhering to the specific requirements of CPSs as MCSs?

In order to address these questions and the research objective, a CDAS can not be considered standalone because real-time support and plug-and-play capability involve the whole CPS device. Therefore, designing a new CDAS architecture implies the redesign of the System Architecture and the Communication Infrastructure of the CPS device. Furthermore, a good task partitioning between hardware and software modules in the MPSoC-FPGA should be done, taking advantage of the platform heterogeneity. Based on the open-interface KIDS-CT concept and the weakness of the existing CS and DAS, also minor research objectives should be achieved by the new CDAS for MPSoC-FPGAs in conjunction with the proposed System Architecture and Communication Infrastructure for CPS devices:

1. A Centralised System Architecture for CPSs that provides the plug-and-play support at system level.
2. A Communication Infrastructure that supports the integration of custom/standard protocols, facilitating the realization of the plug-and-play capability and the real-time communication between components.
3. A CDAS architecture for MPSoC-FPGAs that permits the integration of additional components in the CPS device.
4. A partitioning approach of real-time and non-real-time tasks that exploits the heterogeneity of the MPSoC-FPGA for providing real-time support and facilitating the integration of new components.
5. A lightweight dataflow architecture that provides support for on-the-fly data acquisition from sensors with different data rates that contribute to providing plug-and-play capability and real-time support.
6. A lightweight dataflow architecture that provides support for on-the-fly image processing, configurable for standard and custom data formats. In addition, it should offer the capability to explore the design space by tuning different data formats.
7. A selection approach to reduce the number of metrics and parameters of the design space that aim to find the best data format and computing performance in a targeted CPS application, such as interventional CT.
8. An isolation method for providing temporal and spatial isolation which results in a Lightweight Protection Unit suitable for low-cost MPSoC-FPGAs.
9. A prototype of the CDAS architecture should be implemented on an MPSoC-FPGA. It should be able to provide real-time support and plug-and-play capability for the KIDS-CT scanner, enabling the support for multimodality techniques and interventional procedures.

## 5 Methodology

After having analyzed the weaknesses of existing architectures for component control, data acquisition, and data pre-processing in CPSs and CT scanners, this Chapter defines the ideas and the methodology to achieve the defined objectives. Starting from the requirement definition, it introduces the methodology followed for designing the proposed System Architecture for CPSs, the CDAS architecture, and the optimization ideas that contribute to reaching the research objectives. The content of this Chapter has been published by the author in Ref. [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7, DP 8].

### 5.1 Requirement Definition For The Selected CPS Application

In order to define a generic architecture that provides the plug-and-play capability and fulfills real-time requirements, this work has started to define the *common* and the *specific* requirements for a target CPS application. The former permits the requirement matching between a selected CPS application and the proposed architecture. The latest considers the constraints for the specific application and functionality and eventual plug-and-play extensions. As shown in Fig. 5.1, the requirement definition consists of three steps: the Application modes/functionality, Application requirements, and Design requirements.

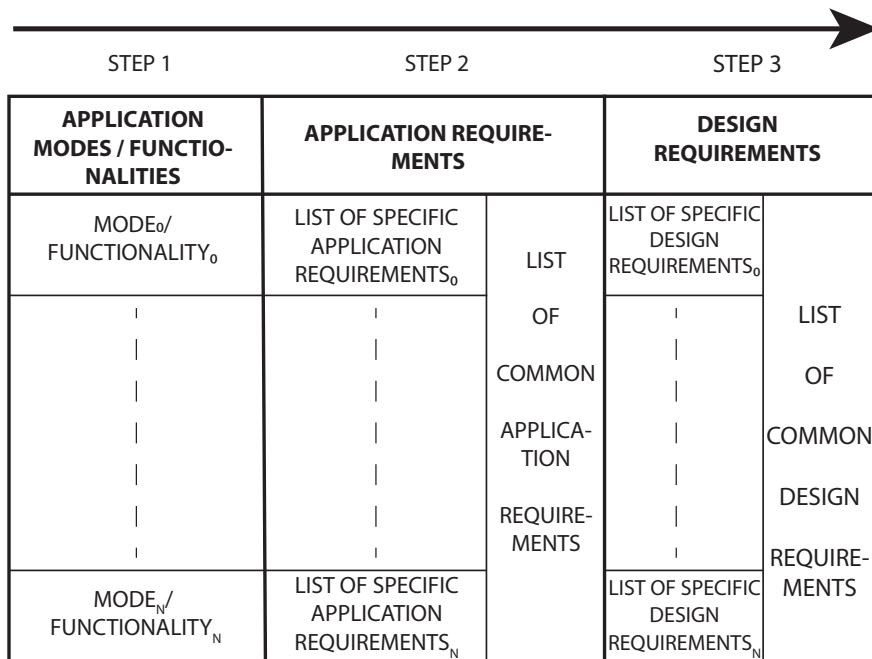


Figure 5.1: Application classification and requirement definition steps

In the first step, the designer analyzes the CPS functionalities from an application perspective (e.g., medical procedures in the case of a CT scanner). In the second step, the designer defines common and specific application requirements for the related application functionalities; at this step, the requirements are not related to any hardware or software implementation but only to the functionalities (e.g., image processing, real-time controlling, etc.). In the third step the designer defines design requirements for the hardware/software architecture of the ES within the CPS, based on the defined application requirements (e.g., support of custom protocols). The Section 8.1 explains the application of these steps in the case of the CT application, where the designer starts from the various CT procedures up to define the design requirements for the hardware/software architecture of the KIDS-CT scanner.

## 5.2 System Architecture

As discussed in Chapter 4, a CDAS cannot be considered as a standalone system to provide real-time support and plug-and-play capability in a CPS device. In order to have a CDAS that can communicate in real time with all components and that can be easily extended with new components, this work considers a centralized System Architecture where each component has a *physical level* and a *logical level*. The main idea behind this solution is to have a CPS where all components are connected to the CDAS. In addition, these levels allow the separation of the physical requirements, such as interconnection interfaces, from the logical requirements, where the functional and application requirements are considered.

## 5.3 Communication Infrastructure

A CPS may contain various vendor components that also have interfaces and protocols with different communication requirements: real-time and non-real-time communications, and small and huge amounts of data to be transmitted. Therefore, to facilitate the integration of new components into a running CPS and the interoperability between these, protocols and interfaces are grouped within classes and layers in the Communication Infrastructure. The key idea of such a solution is given by the fact that classes allow the separation of interfaces and protocols based on the data and time requirements, while the layers allow a systematic implementation of different vendor interfaces and protocols from the *communication interface layer* up to the *application protocol layer*, where a common protocol per class is defined. For realizing this Communication Infrastructure, the CDAS plays a fundamental role as it implements the master for each master-slave communication and the server for the client-server communication in the CPS device. In this way, the integration of new components into the CPS device determines the addition of new modules only in the CDAS, facilitating the plug-and-play capability.



## 5.4 Control-Data Acquisition System

As explained above, the CDAS architecture is the core of the centralized System Architecture and Communication Infrastructure. On the one hand, it simplifies the interoperability and synchronization problem between different components, exploiting the heterogeneity of the MPSoC-FPGA platform. On the other hand, the mapping methodology for the different task components and groups becomes crucial to meet the criticality and timing requirements. Based on this, the main idea for the CDAS architecture is to implement different modules for the application requirements, separating the related functionality into real-time and non-real-time tasks and mapping them on the PL and PS respectively.

To achieve real-time support, plug-and-play capability, and task optimization, various optimization approaches specific to MPSoC-FPGAs have also been proposed within the CDAS architecture:

- A lightweight re-configurable dataflow architecture that collects and prunes data on the fly. To do so, different clock domains have been defined in the datapath, which permits the buffering of collected data in on-chip memory and forwarding them fast enough to avoid external memory. In addition, the architecture is designed to be configured at design time, and the PS can re-configure it at run time, facilitating the integration of new components in the CPS.
- A data processing architecture that can be configured to integrate processing algorithms implemented in dataflow hardware units. It can also be configured for custom and standard data formats. This allows the design space to be explored by tuning the different data formats. The idea is to provide an architecture in which data processing algorithms can be integrated and then used to find the best data format for the target application.
- An isolation method for low-cost MPSoC-FPGAs that can be used to isolate the CDAS modules associated with different external components or criticalities. The idea behind this is to realize a Lightweight Protection Unit (LPU) that checks and grants/denies AXI4 transactions. While memory regions and protection domains must be configured at design time to optimize the execution time and resource utilization, the policies are set at run time to provide temporal isolation.

All these optimization solutions focus on a different part of the CDAS architecture to achieve the research objectives and to allow its configuration for a specific CPS application such as the KIDS-CT scanner.

## 6 Cyber-Physical System Architecture

To achieve the research objectives and to integrate the proposed CDAS architecture into a CPS device, it is also necessary to define a System Architecture and its related Communication Infrastructure. For this purpose, this Chapter initially classifies tasks based on requirements identified in the related work, where separate architectures are proposed for the CS and the DAS. Then, a System Architecture, including the CDAS, is proposed based on these classified task groups. Finally, a Communication Infrastructure is proposed to meet the different communication requirements of the task groups within a target CPS. The proposed solutions discussed in this Chapter have been published in Ref. [DP 2, DP 3, DP 4, DP 7].

### 6.1 Requirement & Task Classification

The task classification step allows the definition of the task requirements for the proposed CDAS, for the System Architecture and its Communication Infrastructure. This takes into account the interactions and the interoperability between the various components. Following the method presented in Section 5.1, it is crucial to identify the *common* and *specific* requirements from the various functionalities. Based on the analyzed state of the art on CSs and DASs for CPSs such as the KIDS-CT scanner, the tasks have been categorized into the following three groups:

- **Control and Synchronization:** This group includes all the control and synchronization tasks for the various CPS components. Each component must have at least one control task implemented in the CDAS to be integrated into a CPS; this task may be real-time or non-real-time. Unlike control tasks, synchronization tasks are not mandatory for each component and may involve more than one component. Since vendors may use custom or standard protocols associated with these control and synchronization tasks, custom peripherals must be supported by the CDAS. Furthermore, the communication tasks associated with the control and synchronization tasks involve a large number of messages containing small packets and/or control signals.
- **Data Acquisition and Collection:** This group includes all real-time and non-real-time tasks that directly acquire or collect data from a CPS component (i.e., sensor). They differ from control and synchronization tasks because of the large amount of

data to be considered. Due to the amount of data, they have been divided into separate task groups. This categorization is necessary because different optimization strategies are required in terms of task organization to meet real-time requirements. Consequently, communication in this context typically involves messages with large packet sizes.

- **Data Processing:** This group includes all tasks that process large amounts of data. While the *Data Acquisition and Collection* group includes tasks that consider communication issues related to sensors that stream data at different data rates (e.g., a different data alignment), this group focuses on the processing algorithms. The tasks of this group are strictly application-dependent; therefore, the CDAS architecture only aims to provide mechanisms to integrate them easily.

Different hardware/software modules are proposed to implement task groups within the CDAS architecture. In contrast with related works that implement different tasks on separated CSs, DASs, and DPSs, here all tasks are implemented in the CDAS. In addition, the proposed Communication Infrastructure separates the communication tasks associated with different groups into distinct interface and protocol classes. This classification simplifies the integration of new components into the CPS device by separating tasks with different requirements.

## 6.2 System Architecture

The System Architecture is the conceptual model of a CPS device, representing the components, the behavior, the connections, and interactions [148]. The proposed model consists of a *physical level* and a *logical level*, as shown in Fig. 6.1.

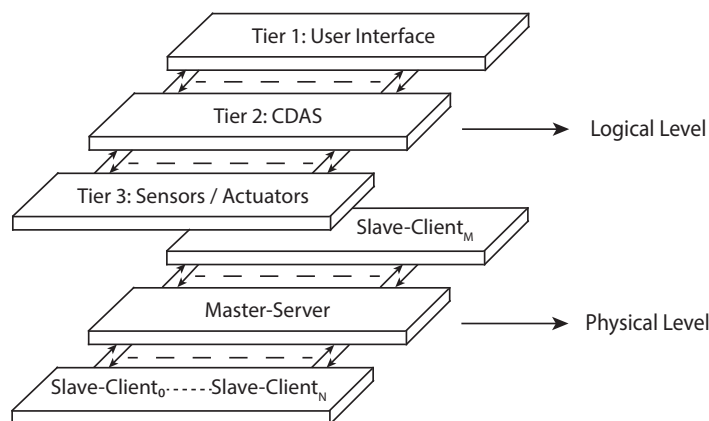


Figure 6.1: System Architecture of a CPS

At the **physical level**, as shown in Fig. 6.2, a centralized architecture has been proposed. It has two types of nodes: the *master-server node* and the *slave-client node*. The

former consists of the CDAS for MPSoC-FPGA. The latter consists of sensors/actuators, user interfaces, and additional DPSs, which can be implemented on workstation PCs, microcontrollers, MPSoCs, and FPGAs. All slave-client nodes are connected to the master-server node, which implements the masters for the master-slave communication and the server for the client-server communication. This centralized solution allows a more straightforward realization of real-time communication and synchronization than distributed architectures. It also simplifies the realization of the plug-and-play capability at the *physical level*, as integrating a new component into the system merely involves connecting a new node to the CDAS, which will control it.

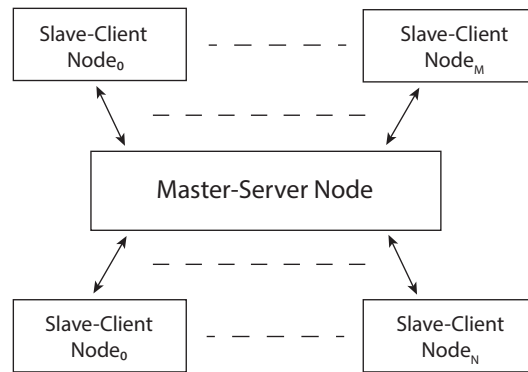


Figure 6.2: System Architecture: Physical level

At the **logical level**, as shown in Fig. 6.3, this work proposes a multi-tier architecture where the user sends commands from Tier 1 to Tier 2. Here the commands are interpreted, adapted for the different components, and sent to the Tier 3, where the sensors/actuators are updated. In the multi-tier architecture, each upper tier can only communicate with the lower tier and vice versa.

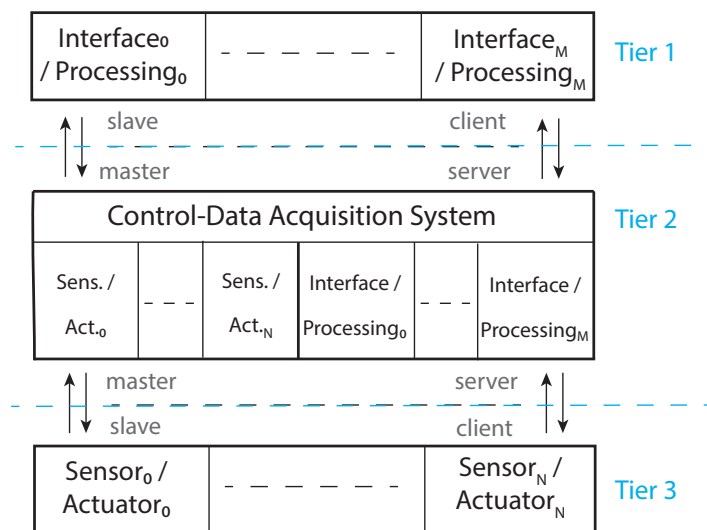


Figure 6.3: System Architecture: Logical level

The multi-tier architecture at *logical level* also facilitates failure avoidance while allowing

users to parameterize the various sensors/actuators. Although users do not have direct access to sensors/actuators located at Tier 3, they can configure them by sending commands from Tier 1, as explained above. In this way, users do not need to know which protocols are used to establish the connection between the CDAS at Tier 2 and sensors/actuators at Tier 3.

### 6.3 Communication Infrastructure

The Communication Infrastructure is the glue element between the nodes in the proposed System Architecture. It defines the internal communication within the CPS device, specifying component interfaces at the *physical level* and transport and application protocols at the *logical level*. In order to provide the plug-and-play capability, the Communication Infrastructure aims to be vendor-agnostic. In addition to related works that only support standard protocols such as OPC UA [149], the proposed solution also supports custom interfaces and protocols. For instance, this feature is essential in the KIDS-CT scanner, where slave-client nodes use different custom protocols defined by the related component-vendor.

In the Communication Infrastructure shown in Fig. 6.4, all slave-client nodes are connected to the master-server node, and each node is modeled with three layers: the *communication interface layer*, the *transport protocol layer*, and the *application protocol layer*. This layered solution permits the master-server node to communicate with components from different vendors and to add plug-and-play components. While the master-server node implements the interfaces and transport protocols based on the connected slave-client nodes, it unifies all the different protocols at the *application protocol layer*.

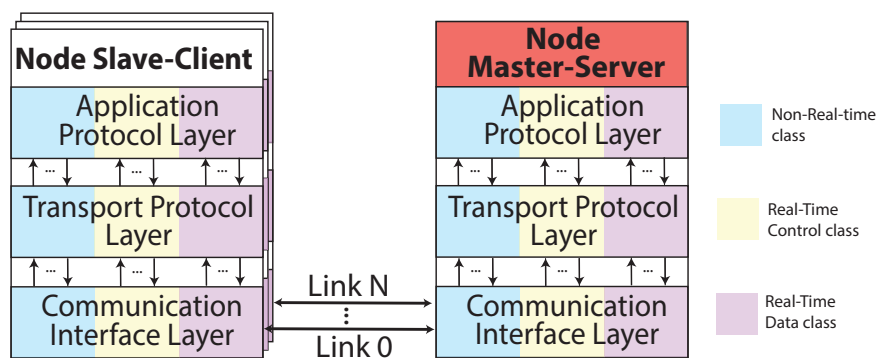


Figure 6.4: Communication Infrastructure

Furthermore, as shown in Fig. 6.4, each node also has three types of classes per layer: the **real-time control class**, the **real-time data class**, and the **non-real-time class**. These classes allow handling different communication requirements (e.g., real-time and non-real-time data and control packets). Therefore, each interface and protocol per link is associated with an appropriate class for each layer when a new component is added.

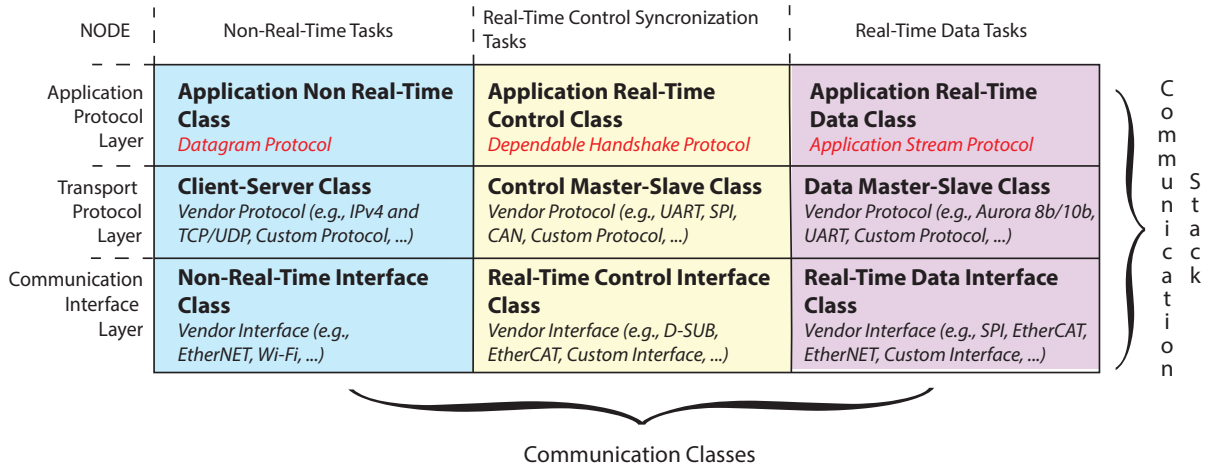


Figure 6.5: Example of the master-server node in the Communication Infrastructure

An example of a master-slave node with different interfaces and transport protocols per class (UART, IPv4, etc.) is shown in Fig. 6.5. However, there is only one protocol per class (written in red) at the *application protocol layer* that unifies the different vendor protocols. Consequently, each proposed application protocol has been proposed based on the requirements of the associated class.

Although the Communication Infrastructure is vendor-agnostic, the components must fulfill two requirements. The first states that each node must have its own control unit, which sets the node in safe mode if the connection is lost or an error is propagated. The second formulates that a node must have at least one control interface (real-time or non-real-time); if a node had only the *real-time data interface*, it could not be controlled or triggered from outside.

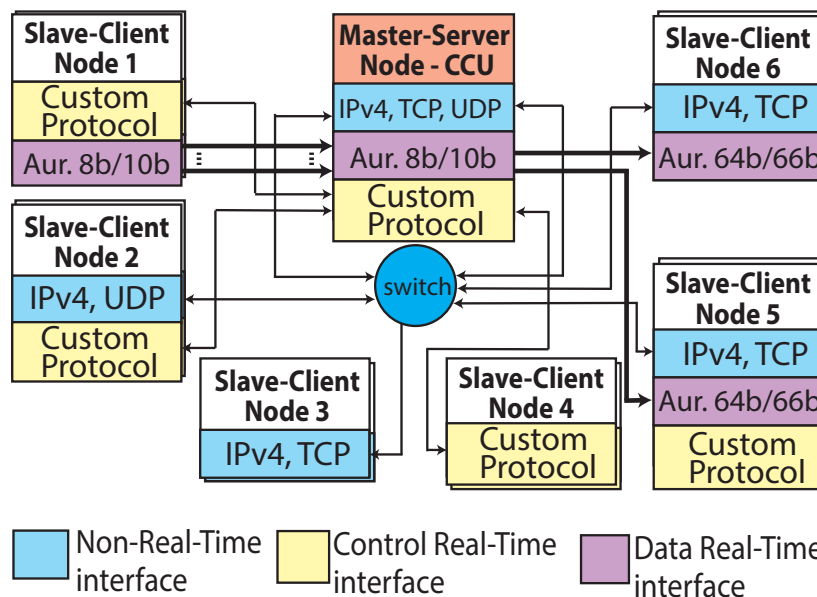


Figure 6.6: Example of node interconnection and interface layer

Fig. 6.6 shows a generic example where nodes have different protocol types and number of interfaces per node. This example highlights the solution for the interoperability problem. Although the protocols of two nodes differ at both the *communication interface layer* and the *transport protocol layer*, at the *application protocol layer* data are encapsulated in a unique, common protocol for each class. Furthermore, it reports the case where nodes have only one interface, such as nodes 3 and 4, which have only the *non-real-time interface* and the *real-time control interface*, respectively. At the *application protocol layer*, the master-slave node (i.e., CDAS) manages the communication between tasks of these different classes interfaces as separate communications. In this way, there are no timing issues between non-real-time and real-time tasks because they do not interfere with each other.

### 6.3.1 Communication interface layer

Each node may have one or more interfaces with different requirements. Therefore, the following three classes have been defined:

- **Non-real-time interface class:** This class groups the interfaces for non-real-time communication between different nodes. It contains all interfaces that will be associated to the *client-server class* in the upper layer (e.g., Ethernet and Wi-Fi connections). Non-real-time control and data processing tasks use the interfaces such as sensor/actuator configurations. Often, in MPSoC-FPGAs these interfaces are already implemented as peripherals in the PS part, and they are accessed by software modules running on the APU.
- **Real-time control interface class:** This class includes interfaces for real-time control data and synchronization signals. All of these interfaces use master-slave protocols in the upper communication layers. Examples of instances for this *communication interface class* are interfaces for Universal Asynchronous Receiver Transmitter (UART), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), EtherCAT protocols, and/or custom protocols using custom signals.
- **Real-time data interface class:** This class includes interfaces associated with real-time data-flow tasks, where large amounts of data are transferred from sensors to the collecting system or processing units. The main idea behind this class is to support streaming communication on the upper layer, independently of its data rate. The transceiver can require a reference clock and dedicated transceivers, such as the Gigabit Transceiver (GTX) [150], to support high-speed communication. For this reason, interfaces of this class within the CDAS are usually mapped with transceivers placed on the PL part of the MPSoC-FPGA.

### 6.3.2 Transport protocol layer

The *transport protocol layer* is above the *communication interface layer*. It also has three classes that are associated with the classes of the above layer:

- **Client-server class:** It groups all client-server protocols for non-real-time communication. Usually, each component uses a transport protocol for this class which is defined by the vendor. Therefore, the CDAS which is the master-server node should support different transport protocols for communicating with the various nodes at the same time.
- **Control master-slave class:** This class includes real-time protocols that send control/synchronization messages. In this class, custom protocols that use control signals are also encapsulated as packets within messages. The key idea is to separate small and large packets because packets of similar size can mitigate the *convoy effect* [151]. A uniform packet size allows an accurate estimation of the WCET, essential for real-time tasks. If a communication involves a real-time protocol with a large amount of data, it is not included in this class. Examples of protocols in this class are UART, I2C, SPI, and Ether-Cat.
- **Data master-slave class:** This class includes real-time protocols that transmit data as a stream. The PL part of the MPSoC-FPGA supports the implementation of transceivers that can stream data on-chip. This allows data to be processed on the fly, which is essential for real-time requirements. Aurora 8b/10b [103] and PCI-Express [152] (stream mode) are examples of protocols associated with this class.

### 6.3.3 Application protocol layer

On top of the communication stack is the *application communication layer*, consisting of three classes. Unlike the other communication layers that implement various vendor protocols, this unifies all of them with a unique protocol per class. In this way, it is possible to provide interoperability between nodes with different interfaces and transmission protocols. Furthermore, it facilitates the realization of the plug-and-play capability. This layer has the following classes:

- **Application non-real-time class:** This class groups non-real-time tasks that communicate with client-server protocols. Since messages for these tasks range in size from a few bytes to several megabytes, a generic **Application Datagram Protocol** has been proposed for this class.
- **Application real-time control class:** This class includes real-time control and synchronization protocols, which can be custom or standard. Although different



real-time protocols need to be used for the different requirements, this thesis proposes a **Handshake Protocol** that handles communication errors and lost messages in real time.

- **Application real-time data class:** This class groups communication real-time tasks that stream large amounts of data. It includes serial communication protocols having a variable message size. Further, messages can be sent in simplex mode between sender and receiver, which lacks synchronization signals. To solve this problem, this thesis proposes the **Application Stream Protocol** that uses various commands in and between messages to synchronize the sender and the receiver.

### Application Datagram Protocol

In the Application Datagram Protocol, data are sent as messages of variable size. Each message also consists of a variable number of packets with their own variable size and data format, as shown in Fig. 6.7. This flexible and scalable structure allows the same protocol to be used for small commands and large data messages. In addition, having packets of different sizes allows commands and large amounts of data with different formats and sizes to be encapsulated in two packets of the same single message. Packets are aligned to 16 bits, consequently, command messages smaller than this size have padding bits. As shown in Fig 6.7, the Application Datagram Protocol has the following three segments: the *Header Segment*, the *Data Segment*, and the *Tail Segment*.

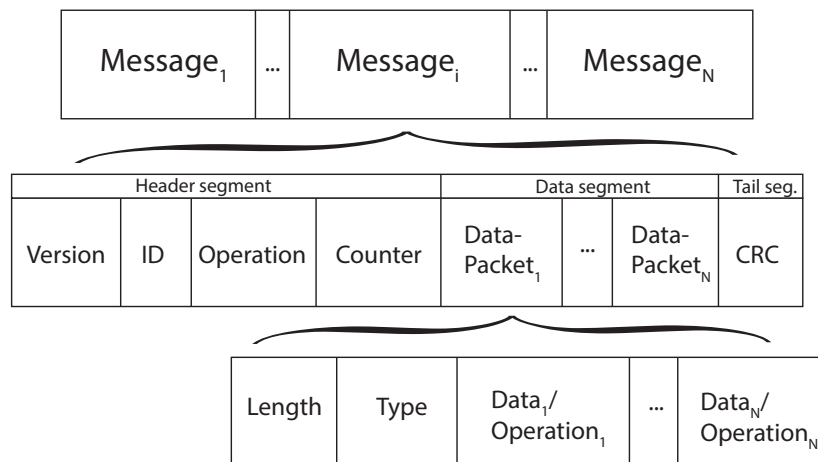


Figure 6.7: Message structure of the Application Datagram Protocol.

The **Header Segment** has four mandatory fields. The first is the *Version* field, which defines the actual client-server protocol version. It is used to verify that a client and the server are using the same protocol version, thus avoiding decoding errors. In the case of different versions, the communication is terminated. The second field is the *Unique Identifier (UID)* that identifies each application request. If the size of an application

message exceeds the Maximum Transmission Unit, it is split into several messages, and the UID is incremented by 1. The third field is the *Operation* field, which specifies the server operation to run for the received message and component. The fourth is the *Counter* field that defines the number of body segments within the message. The body segment is empty when the operation has no data information, and the *Counter* equals 0.

The **Data segment** contains commands and data. These are encapsulated in data packets of variable size. To decode such variable packets, there is the *Length* and *Type* fields that define the number of bytes and the data type of the sent data. A message ends with the **Tail segment** that contains the Cyclic Redundancy Check (CRC) used by the receiver to check the integrity of the received message.

### Handshake Protocol

The Handshake Protocol extends the transport protocols at the *application communication layer*, providing real-time error detection and recovery for the *application real-time control class*. It also prevents denial-of-service attacks on the master side by terminating the communication within the receiver if a message is not expected or fails the integrity check.

The vendor transport protocols must meet two essential criteria in order to be supported by the Handshake Protocol. Firstly, all communication can only be initiated from the master side. Secondly, every message received under the master-slave protocol requires an acknowledge (ACK). A unique acknowledgment signal (e.g., valid/error signal) is mandatory if a control/synchronization message lacks the CRC.

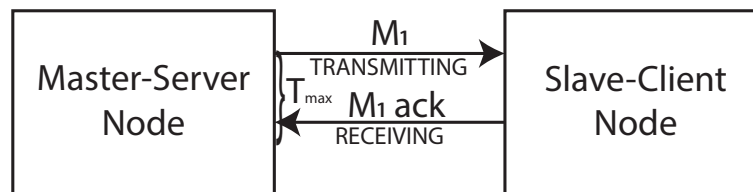


Figure 6.8: Handshake protocol in case of no errors.  $T_{max}$  is the maximum time before a timeout event

The Handshake Protocol is implemented close to the transceiver (i.e., transmitter/receiver) in the PL of the CDAS. In this way, timeout events can be managed and a retransmission mechanism can be supported using specific signals directly linked to the corresponding transceivers. Moreover, if a malicious slave attempts to initiate communication with the master, the incoming message is halted at the receiver. As these events are managed within the PL, they don't affect the software scheduler or the execution of other tasks in the CDAS. The Handshake Protocol has the following two phases, as shown in Fig. 6.8:

- **Transmitting phase:** In this phase, the master initiates the communication by sending a single or a burst of messages to a slave-client node. Once the messages

have been sent, the master waits for the ACK message and enters the receiving phase. The receiver within the master-server node can be set for checking a final single ACK or a burst of ACKs.

- **Receiving phase:** When the message arrives at the slave-client node, the Receiving phase has been initiated on that side. Here, the slave-client node receiver verifies the integrity of the message and sends back the corresponding ACK. This includes the ID of the received message and other information depending on the specific transmission protocol. Once the ACK arrives at the master side within the timeout, the receiver of the master-server node compares the ID contained in the ACK message with the ID of the transmitted message. If the IDs match, the communication is correctly terminated, as shown in Fig. 6.8.

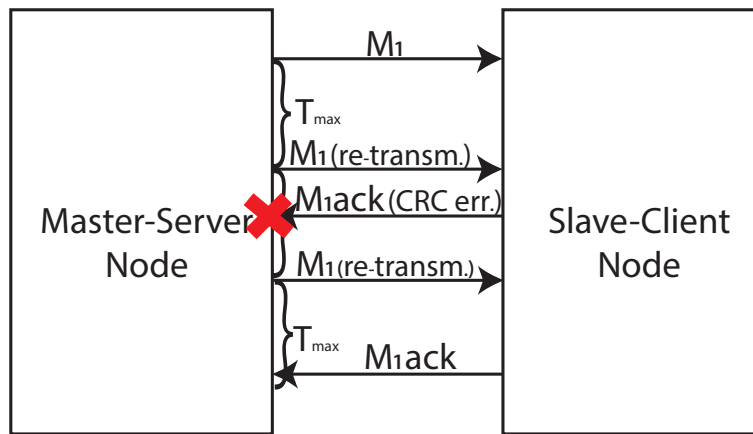


Figure 6.9: Handshake protocol. Message lost and CRC error use case

However, the Handshake Protocol generates the retransmission signal if the ACK exceeds the timeout or fails the integrity check. In the example shown in Fig. 6.9, the master does not receive the ACK within the timeout ( $T_{max}$ ), so the retransmission signal is set in the master transmitter. Then, the slave-client node sends the ACK, but a CRC error occurs, so the message is blocked and the retransmission signal is set again. Finally, the slave-client node receives the correct message and sends back the ACK, which matches with the transmitted message, and the communication is successfully completed. An error flag is set in the corresponding status register if no correct ACK is received after the maximum number of retransmissions. Depending on its severity, the error event can then be passed to the appropriate software module for handling it (e.g., the CPS runs in safe mode).

### Application Stream Protocol

The Application Stream Protocol is conceived to handle a stream of data in simplex mode. The protocol consists of messages and commands. The messages contain the data and can

be of variable size and format. The commands define the start and the end of each message and allow the receiver to synchronize with the transmitter. In this way, the Application Stream Protocol enforces the integrity and synchronization issues of the protocols in the lower communication layers.

Commands and messages consist of packets, which are the smallest amount of information that can be sent. A packet is sent each clock cycle and corresponds to the data parallelism of the sender/receiver. For instance, in a transceiver that sends 32 bits per clock cycle, the packet is 32 bits. For this reason, the packet size is only configurable at design time. To transmit such packets as a stream, the transmitter encodes and sends them as a stream of bits, and the receiver decodes and aligns them. For this reason, the synchronization is crucial for the receiver.

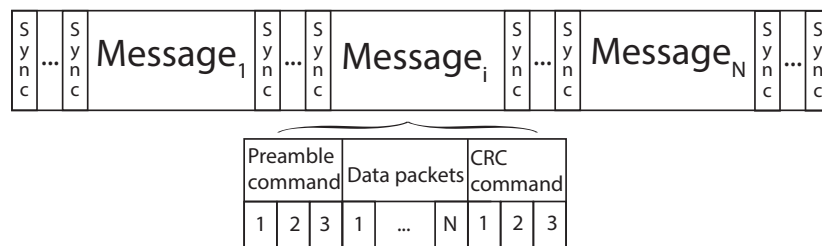


Figure 6.10: Application Stream Protocol

As shown in Fig 6.7, synchronization is performed between each message. The *synchronization command* consists of a single packet that is sent until a new message is ready to be sent. The message is announced through the *preamble command*, a fixed sequence of three packets. Finally, to end a message, the *CRC command* is defined, with a sequence of three packets. This command is also used to identify the CRC packet, which has a fixed size and is the last packet of the message. In this way, the receiver can check the integrity of the message and commit the whole message or flush the First In First Out (FIFO). Note that the *synchronization command* is the only one that consists of a packet because it is sent between messages and cannot be misread as a data packet. Furthermore, it simplifies the synchronization because the decoder aligns the data to this single command. By contrast, the other commands consist of three packets because they are part of the message, and a single packet command could be wrongly misread as data packets. Moreover, these commands enable the support of messages having variable sizes.

When the CPS is powered up, each transmitter sends *synchronization commands* (i.e., packets). In this phase, each corresponding receiver aligns the stream of bits to the *synchronization command*. As soon as a message is sent, the receiver is synchronized and ready to decode it. The receiver stores the entire message in a FIFO. When the *CRC command* is detected, the calculated CRC of the message must match with the last packet to commit the message and forward it to other tasks or internal modules of the node.

## 7 Control-Data Acquisition System

This Chapter introduces the proposed Control-Data Acquisition System (CDAS) architecture, which forms the master-server node of the System Architecture and Communication Infrastructure. It describes the hardware/software co-design methodology used for the CDAS architecture and the proposed optimizations that contribute to achieving the research objectives. The CDAS architecture includes the Data-Flow Module and the Data-Processing Module which consist of two lightweight configurable dataflow architectures that collect and process data on the fly. Furthermore, the Data-Processing Module can be used to explore the design space and optimize the targeted CPS application by tuning the processing data formats. Finally, this Chapter introduces the proposed isolation method for MPSoC-FPGAs, which is crucial for CPSs with MCSs requirements, such as medical Cyber-Physical Systems (CPSs) (e.g., the KIDS-CT scanner). All these contributions of the CDAS architecture have also been discussed by the author in articles [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7].

### 7.1 Task Partitioning

In this thesis, tasks are classified into three groups: the control-synchronization, the data-acquisition/collection, and the data-processing group. Within these groups, tasks can also be distinguished between real-time and non-real-time. For partitioning the tasks in the CDAS architecture, two steps have been proposed to provide real-time support and plug-and-play capability:

1. **Tasks associated with different task groups are partitioned into distinct modules for each slave-client node.** This modular approach not only simplifies the integration of new components into the CPS but also facilitates the isolation of tasks from different nodes and groups. As explained in Section 6.1, task groups are categorized based on their operational nature (i.e, component control/synchronization, data collection, and data processing). As a result, tasks within each group share similar characteristics and requirements in terms of memory usage, processing capacity, and I/O demands.
2. **Tasks are divided into real-time and non-real-time tasks within each task group and partitioned on the PL and PS, respectively.** This partitioning method has several advantages for both types of tasks. Tasks implemented in the PL are statically mapped and do not need to be scheduled, which is usually the

main issue for real-time tasks. In CPSs, critical tasks are usually real-time, so implementing them on PL facilitates their isolation. In fact, for tasks that are statically mapped on the PL, only spatial isolation is required. Tasks running on PS can take advantage of a CPU architecture and an OS (e.g. virtual memory, multi-threading, software stack, and software libraries).

The proposed approach can be translated into the mapping steps shown in Fig. 7.1, where the tasks of each component and use case are first assigned to the task groups and then separated between real-time (PL) and non-real-time (PS) computing. In the explained solution, each use case is mapped independently, starting from the associated slave-client node. In this way, the addition of a new component is facilitated because it doesn't affect the previous mapping decisions. Furthermore, a new component can be integrated into the CPS by iterating such steps that do not affect the hardware/software modules of the existing slave-client nodes.

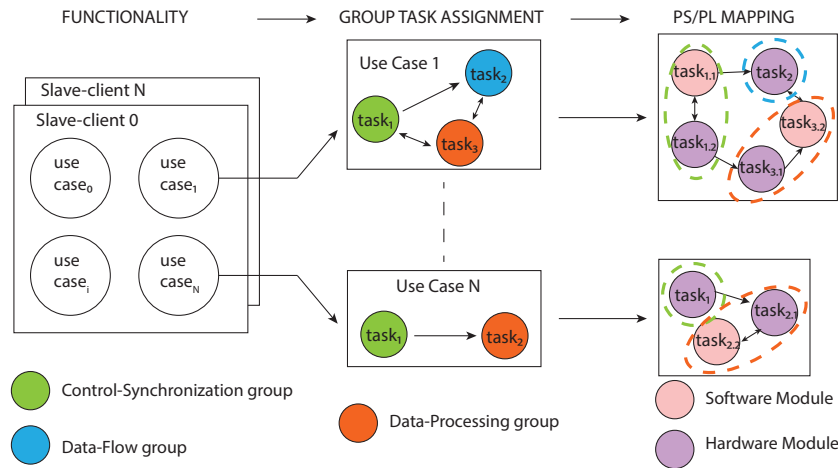


Figure 7.1: Task partitioning and mapping steps

In addition, while the first step is platform independent, the second step assumes a SoC including programmable logic, such as in the targeted MPSoC-FPGA platform.

## 7.2 Hardware/Software Architecture

The CDAS is mainly responsible for controlling and synchronizing all CPS components, collecting data from the various sensors, and processing them in real time. Furthermore, it handles all the communication tasks between components, implementing the masters for the master-slave communication and the server for the client-server communication. Based on the group classification, the CDAS architecture contains the following three modules for each external component (i.e., slave-client node):

- **Control-Synchronization Module:** This is responsible for controlling a component and then synchronizing it with other components. While other modules

are not mandatory, each component must have at least the associated Control-Synchronization Module in the CDAS. The architecture of this module will be described in Section 7.3.

- **Data-Flow Module:** This is responsible for collecting data from a component (e.g., sensors), pruning, and forwarding them to internal and external processing units in real time. It implements the various protocols for the *real-time data class* in a lightweight re-configurable dataflow architecture that processes data on the fly, without using external memory. It is implemented in the PL and is configured by a software module running on the PS. It provides the capability to configure the datapath at run time to acquire/transmit data supporting custom and standard protocols at different data rates, contributing to the plug-and-play capability. The lightweight dataflow architecture will be described in Section 7.4.
- **Data-Processing Module:** This is responsible for processing real-time data. For example, to process sensor data that must be used for updating actuator parameters. In order to fulfill real-time requirements, this work proposes a configurable and scalable processing architecture. It is directly connected to the other modules via streaming interfaces, supporting on-the-fly processing. The architecture of the Data-Processing Module will be described in Section 7.5.

As mentioned above, to optimize the implementation of these modules for MPSoC-FPGA platforms, the real-time and the non-real-time subtasks within each task group are mapped to the PL and PS, respectively. This solution optimizes the execution time and the predictability of real-time critical tasks and facilitates the plug-and-play capability.

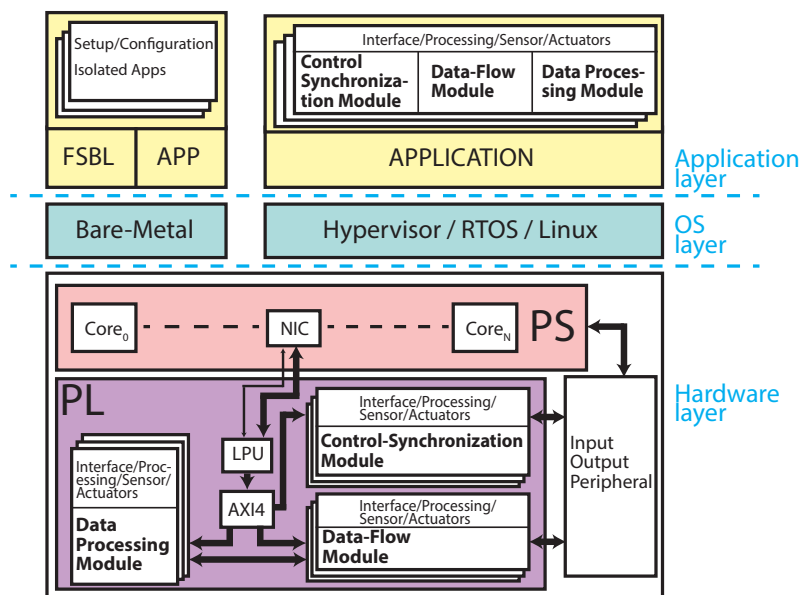


Figure 7.2: CDAS architecture for MPSoC-FPGAs

Following this task partitioning solution, the CDAS consists of the multi-layer architecture shown in Fig. 7.2.

The **Application layer** implements the software modules running on the PS. It includes the “business logic” of the application, which is a software module responsible for implementing the logic of the application, manipulating data, and coordinating the other modules [153]. In the proposed software architecture, the business logic is responsible for coordinating the commands between Tier 1 and Tier 3, and between different nodes in the same Tier.

The **Hardware layer** contains the hardware modules for the task groups running on the PL, which are the Control-Synchronization Module, the Data-Flow Module, and the Data-Processing Module. Subtasks mapped on this layer and the Application Layer, and belonging to the same slave-client node and task group communicate with each other via AXI4-Lite interfaces. Each interface links the memory space of a PL component with the global memory of the PS. This solution allows hardware modules to be executed independently of software modules that communicate asynchronously via the AXI4-Lite interface. This asynchronous communication avoids situations where real-time tasks running on the PL have to wait for non-real-time tasks running on the PS, causing them to miss their deadline.

The communication between these two layers is handled by the **OS layer** that translates the physical addresses of hardware modules into the virtual addresses of the software modules. It can also run different real-time OSs, bare-metal applications, or a hypervisor on the different CPU cores of the PS. For example, this layer includes the First Stage Bootloader (FSBL), which is responsible for configuring and booting the PL and PS parts of the MPSoC-FPGA and booting the OS. Here, the OS implements the scheduler and the isolation access policies that are essential for coordinating real-time and non-real-time tasks and for guaranteeing temporal isolation, respectively. For example, the isolation between modules associated with different slave-client nodes enables error propagation avoidance. In the rest of the Chapter, the architecture of the three modules presented above refers to the single instance associated with a client-slave node.

## 7.3 Control-Synchronization Module

As explained above, the Control-Synchronization Module controls and synchronizes all external components of the CPS. Therefore, it realizes the non-real-time class type and the real-time control-synchronization class type of the proposed Communication Infrastructure.

### 7.3.1 Hardware layer

The Control-Synchronization Module at the hardware layer contains the synchronization unit and the communication unit, as shown in Fig. 7.3. On the base of the associated



slave-client node, this module can have multiple instances for these two units.

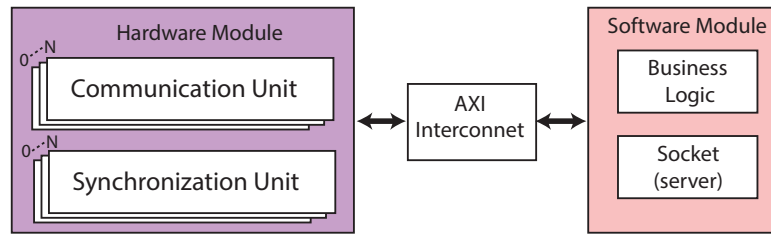


Figure 7.3: Control-Synchronization Module associated to a single slave-client node. The hardware and the software modules are implemented at the hardware and software layers, respectively

The master communication unit handles the communication tasks for the *control master-slave class*, the modules that realize the proposed Handshake Protocol for the *application real-time control class*.

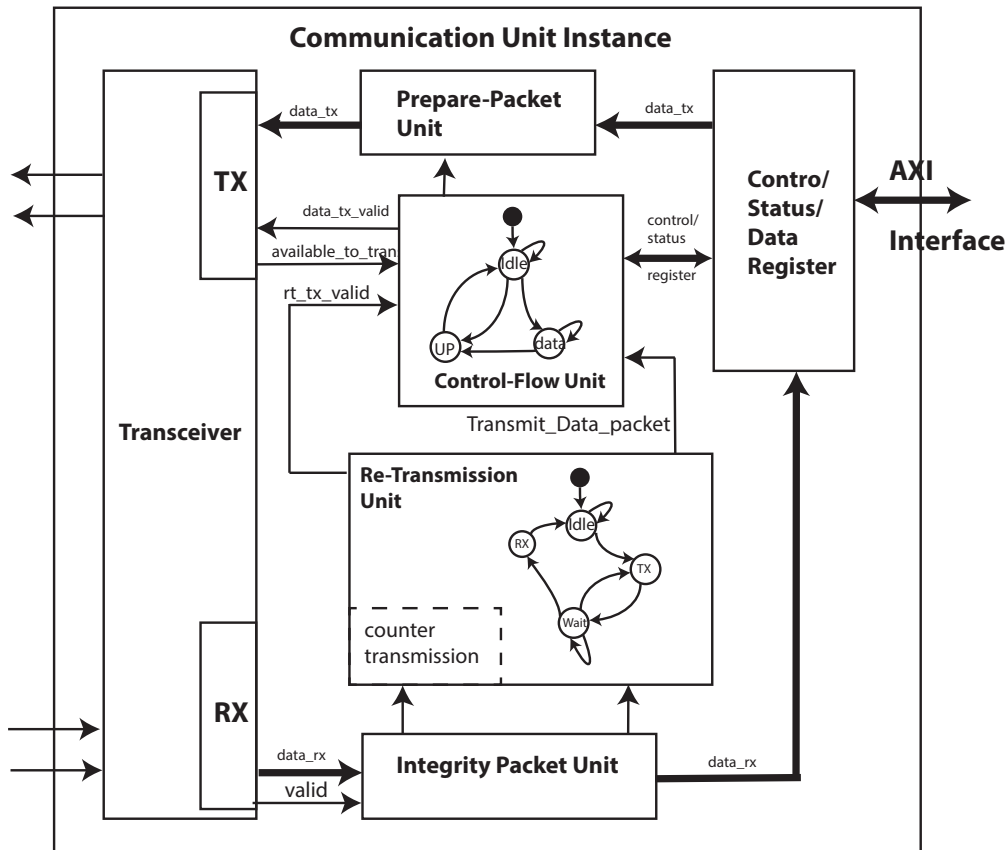


Figure 7.4: Control-Synchronization Module: Communication Unit instance

As shown in Fig. 7.4, an instance of this module associated with a slave-client node has the following units:

- **Transceiver Unit:** It implements the master transmitter/receiver for the protocol of the associated slave-client node. Each protocol has its own transceiver; therefore, if a

node uses multiple connections, multiple transceiver units must also be instantiated, each associated with it.

- **Register Unit:** It contains the control, the status, and the data registers. In order to send commands to the hardware units of the Control-Synchronization Module, the corresponding software module sets the proper bits in the control registers. Furthermore, the associated software module reads the status register to check the state of this module (e.g., check errors and transmission status). The data registers contain the received data and the data to send over the transceiver.
- **Prepare-Packet Unit:** When data to be sent are available, this unit packets them, and waits for the *send\_valid* signal for sending data to the transceiver unit.
- **Control-Flow Unit:** It implements the logic that coordinates the flow of packets. When data are ready to be transmitted or re-transmitted, it checks the availability of the transceiver and generates the *send\_valid* signal. As shown in Fig 7.4, it contains a FSM having following states:
  - **start:** The initial/reset status used for setting the transceiver. After sending the setting command, the FSM runs in **idle** state.
  - **idle:** In this state, the FSM waits for the *send\_packet* signal, which is asynchronously set in the status register by the software module. When this signal is high and the data to be sent are not yet available in the data register, the FSM runs in the **data** state. If the *send\_packet* signal is high and the data are ready or the *re-transmission\_packet* signal is high, the FSM runs directly in the **send** state.
  - **data:** In this status, the FSM waits for the *data\_ready* signal from the prepare-packet unit. When packets are ready, the FSM runs in **send** status.
  - **send:** It generates the *send\_valid* signal that triggers the send command in the transceiver. After that, the FSM runs in the **idle** state, where it waits for eventual retransmission or new data to be sent.
- **Re-transmission Unit:** This implements the logic for the re-transmission mechanism of the proposed Handshake Protocol. It uses a counter, which is set for the maximum number of re-transmissions to perform in case of errors. This offers the advantage of re-transmitting messages without additional effort for the software modules, which only have to deal with cases where errors are repeated beyond the maximum number of re-transmissions. As shown in Fig. 7.4, this unit consists of a counter and an FSM with the following states:
  - **start:** It represents the initial/reset status, in which the FSM sets up the unit and the next state to **idle**.

- **idle**: In this status, the FSM waits for the *send\_valid* signal from the Control-Flow Unit. When this signal is high, the **transmitter (tx)** state is triggered.
  - **tx**: It is a transient state in which the FSM runs only for one clock cycle to increment the counter and eventually send the re-transmission signal to the Control-Flow Unit.
  - **wait**: In this state, the FSM waits for the response or the ACK associated to the sent message. If the received message is correct, the FSM runs in the **receiver (rx)** state; otherwise, it runs in the **tx** state, where the re-transmission signal is generated.
  - **rx**: It is a transient state, which sets the unit for sending a new packet and sets the next state as **idle**.
- **Integrity Packet Unit**: This checks the integrity of the received message using the CRC. Then, it compares the ID of the received message with the aspected message. When an error is detected, this unit sends the error signal to the Re-transmission unit, which manages it, as described above.

Moreover, the Control-Synchronization Module shown in Fig. 7.3 contains one or more synchronization units at the hardware layer. These are used to synchronize internal units and client-slave nodes in real time. The logic of these units depends on the specific application. A synchronization event is the fulfillment of a synchronization condition. For example, in the KIDS-CT scanner, the synchronization condition that enables data acquisition checks that the gantry rotation, the DMS and the X-ray tube voltages are stable.

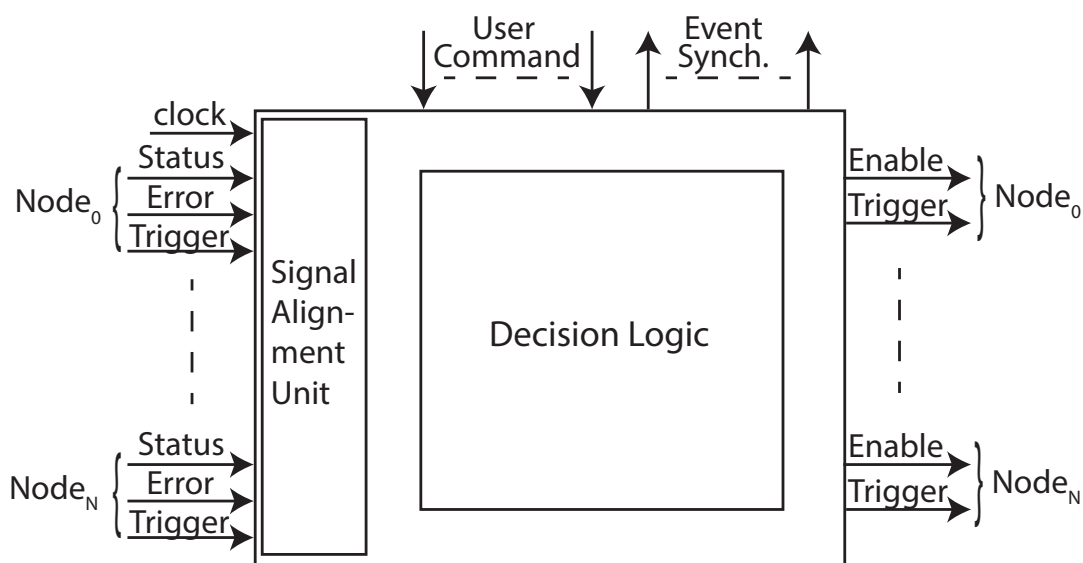


Figure 7.5: Synchronization Unit

As shown in Fig. 7.5, this unit can have input/output signals from/to multiple nodes, as the synchronization may involve more than one node. This module has an internal signal alignment unit that takes all the asynchronous inputs and aligns them to the same clock; this unit avoids metastability problems caused by the different asynchronous inputs. The aligned signals are then forwarded to the Decision Logic unit, where the synchronization is performed, using combinatorial logic. In the case of the KIDS-CT scanner, a single unit is used to synchronize all external components. By implementing the signal alignment unit with Flip-Flops (FFs) per input and the synchronization conditions with combinatorial logic, a synchronization event is updated within 3 clock cycles, which is the delay introduced by the signal alignment unit.

### 7.3.2 Application layer

The Control-Synchronization Module implements the non-real-time control and synchronization tasks at the application layer. It also implements the non-real-time communication tasks associated with the different slave-client nodes, the server for the *client-server class*, and the related datagram protocol for the *application non-real-time class*. While control and synchronization tasks depend on the selected CPS application, the server architecture can be designed as a generic architecture. In fact, this Section focuses on describing the server module that runs the communication tasks associated with each slave-client node within the Control-Synchronization Module and the *application non-real-time class*.

To limit the error propagation within each external component (i.e., slave-client node) and improve system dependability, communication tasks associated with different slave-client nodes use different server ports to establish the connection with the CDAS. In addition, to manage the connections separately at the application layer, the server software architecture has been designed with four modules: *Communication*, *Timer*, *Command*, and *Execute*. Each module handles a distinct aspect of the communication, as described below:

- **Communication module:** This module instantiates the server-socket associated with each slave-client node. When a client connection is established, this module links it to the associated socket base on the connection port. The socket then runs the command module where the message is processed.
- **Command module:** This module handles the communication with the connected client within the server-socket instance. Since a client can only execute a single command thread at the same time, this module is created as a “singleton instance” per client. Here, messages are decoded and forwarded to the corresponding Execute module based on the command operation to be executed. To do that, it utilizes the FSM shown in Fig. 7.6, which decodes the different messages.

- **Execute module:** This module runs the control and synchronization tasks associated with the received command operation. As discussed above, these tasks depend on the specific application and they represent the business logic layer [154] of the CPS architecture. Since controlling and synchronizing tasks can be implemented on the PS and PL parts, this module also interacts with the hardware modules via the AXI4-Lite interface. Finally, it generates the response message for the requested command operation, containing the operation status and the data requested by the client. All the execution steps are controlled with the FSM shown in Fig. 7.6.
- **Timer module:** When a socket is created and the connection is established, the timer is set to N seconds. If a client message (i.e., control/data/ping message) arrives within N seconds, the COUNTER timer is reset and restarted. The TIME\_OUT event is triggered if no message arrives in N seconds. When a TIME\_OUT event occurs, the client connection is closed and the corresponding Execute and Command modules are terminated.

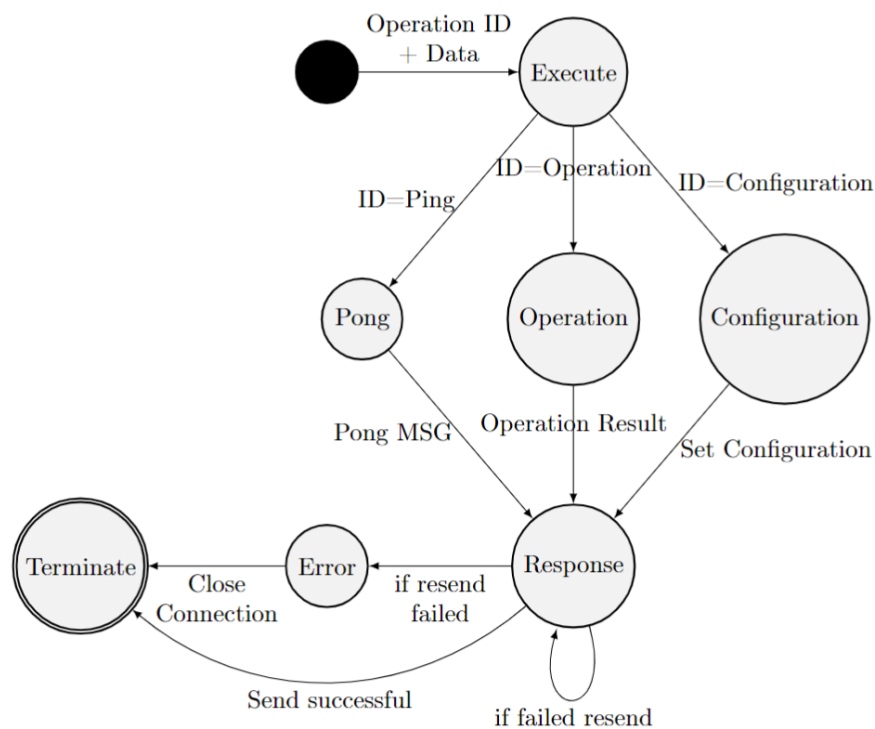


Figure 7.6: States of the *Execute* module during the execution of a received and decoded message from a slave-client node

In order to instantiate and schedule the tasks associated with the different modules of each server-socket, a Real-Time Operating System (RTOS) is required. In fact, it implements the scheduler where the priority of each module is set based on the following hierarchy:

1. *Timer module* (Real-Time priority)

2. *Command module* (High priority)
4. *Execute module* (Medium Priority)
5. *Communication module* (Low Priority)

Beyond this static hierarchy configuration, the RTOS can update at run time the priorities of the various modules and their child tasks based on the operation ID from the incoming datagram message. This priority hierarchy mitigates starvation and deadline race conditions caused by operations running in the Execute module waiting for lost messages. For instance, when a module exceeds the `TIME_OUT`, the timer associated with the server-socket generates an interrupt that terminates all tasks of the other modules and terminates the connection.

### 7.3.3 Example

The example shown in Fig. 7.7 presents a Unified Modeling Language (UML) sequence diagram of a typical scenario involving hardware and software layers of a Control-Synchronization module instance and two slave-client nodes. The first node (on the left) exchanges non-real-time messages using the Application Datagram Protocol for the client-server communication. The second node (on the right) relies on the Handshake Protocol, adopting a master-slave communication. Moreover, this use case shows how non-real-time and real-time tasks interact with each other. This example also shows the interaction between a User interface at Tier 1 of the Communication Infrastructure with a sensor/actuator at Tier 3, crossing the Tier 2 that is the CDAS. In the figure, black arrows indicate the CPS power-up process, orange arrows indicate the client-server communication, blue arrows indicate the communication between internal software and hardware tasks, and green arrows indicate the master-slave communication between the CDAS and the second slave-client node.

When the CPS is powered, the FSBL within the CDAS orchestrates the configuration of the PL and the booting of the PS, where the RTOS runs. Once all the internal modules (both software and hardware) are initialized, the CDAS sets up the other slave-client nodes. For clarity, the UML sequence diagram in Fig. 7.7 illustrates only two nodes with distinct interfaces, but in the system more slave-client nodes can be involved.

The Communication module instantiates the “*Listen task*”, which accepts incoming connections. When a connection is established, this module creates the associated socket, which instantiates the Command module. This last module inits the Timer module while waiting for messages from the client; the timer configuration is based on the specific component. It permits the CDAS to verify the availability of the external component. In fact, if no command, data, or ping messages are received within the `TIME_OUT`, the CDAS



client. A task is then initialized with the received command data, which is stored in an array for active operations via the Execute module thread.

The Execute module matches the operation ID with the related execute task, which may involve the hardware and software layers and other slave-client nodes. For instance, in the UML sequence diagram shown in Fig. 7.7, the Execute module sends a real-time command to the second slave-client node via the hardware layer. As soon as the tasks within the Execute module are completed, the success status and the data are transmitted to the client via the Application Datagram Protocol. If the transmission is successful, the Execute module deletes all retained data, and the corresponding tasks are terminated and removed from the list of active operations. In the case of an unsuccessful response message, two attempts are made, each after a 500ms delay. If no one is successful, the Execute module task activates the TIME\_OUT software interrupt and terminates all related procedures.

## 7.4 Data-Flow Module

This Section describes the architecture of the Data-Flow Module, shown in Fig. 7.2. This collects data from a client-slave node (i.e., sensor) and streams them to the Data-Processing Module and/or to other client-slave nodes. In order to give a comprehensive description of the architecture, this Section refers to a single architecture instance, as shown in Fig. 7.8.

The logic that controls this module is implemented at the software layer, while the datapath has been designed as a lightweight re-configurable dataflow architecture. Furthermore, the proposed solution has been designed not to use any external memory and thus process data on the fly. It can also be configured at design time to be integrated into other CDASs or simple DASs with different requirements (e.g., data type, data-link protocol, data rate, and multiple receivers and transmitters).

The architecture is flexible to collect/transmit data from/to single or multiple slave-client nodes. In addition, the CDAS can implement several isolated instances to guarantee data isolation between different criticality applications running on the CPS. To achieve the research objectives, the proposed dataflow architecture introduces two novel features compared to other DASs in the literature:

- ***Design-time and Run-time Configuration of the Internal Datapath:***  
At design time, the designer sets connection parameters that are based on the physical attributes of the connected slave-client nodes from/to which data will be received/transmitted (e.g., number of receivers/transmitters, interface protocol, size of the FIFOs, etc.). This design-time configuration allows the addition of transmitters/receivers without affecting the existing architecture. In fact, the user configures the communication and acquisition parameters at run time by setting the datapath registers mapped in the global memory of the PS. This run-time



adaptability enables the software module to configure the CPS for different acquisition scenarios. For instance, in the KIDS-CT scanner, the Data-flow Unit is configured at run time for acquiring images of different body parts, affecting parameters such as exposure time, number of detector slices, image dimensions, resolution, and sampling intervals.

- ***On-the-fly image acquisition and processing exploiting on-chip memory and clock domains.*** The Data-flow Unit collects data that are transmitted at different rates from the various slave-client nodes. Based on the rate, each transceiver streams data in the datapath with a specific clock frequency. Therefore, to simultaneously process data on the fly from/to different client-slave nodes, the dataflow architecture has been designed with different clock domains that are crossed by data through asynchronous FIFOs. In order to meet real-time requirements and process data on the fly, the dataflow architecture processes and/or transmits them at a higher clock frequency than the collected/acquired data.

#### 7.4.1 Architecture reconfigurability

The dataflow architecture is configured at design time and can be reconfigured at run time. The design parameters are based on the physical receivers/transmitters and processing units to be instantiated for real-time data communication within the CDAS. In order to have a correct dimension for the FIFOs and to avoid data loss, the designer first defines the depth of each FIFO and then the other parameters. The FIFO depth corresponds to the maximum size of the received messages associated with the proposed Application Stream Protocol. For example, in the KIDS-CT scanner, the FIFO depth is equal to the size of a projection row, since data are collected row by row. Then, the following group parameters must be set at design time:

- **receiver (rx):** It contains the parameters related to the physical receivers that use real-time data communication. The first parameter is **N**, which indicates the number of instances for the physical receivers; it also depends on the input data channels of the receiver since some protocols can use multiple channels/lanes for a single receiver. In addition to the system clock, the designer sets a **rx reference clock** that generates the corresponding **rx data clock** that is synchronous to the acquired data. The frequency of these two clocks can also be updated at run time. This generated clock defines the *Receiver domain* of each instance, as shown in Fig. 7.8 with the light blue color. The output data clock parameter is also set in relation to the **rx data width** parameter defining the **maximum rx data rate** of the collected data. Furthermore, each receiver has a data-link parser that is configured for different protocols. An example protocol is the proposed Application Stream Protocol.

- **transmitter (tx)**: It contains the parameters for the physical transmitters that use real-time data communication. The first parameter is  $\mathbf{M}$ , which indicates the number of instances for the physical transmitters. If the transceiver implements a PCI-Express bus, the number  $M$  depends on the lanes (x4, x8, x16). Like the receivers, each transmitter has a **tx reference clock** that generates an **tx data clock** synchronized to the input data to stream. This clock defines a *Transmission domain*, as shown in Fig 7.8, with the orange color. In relation to tx data clock, the designer set the **tx data width** to define the **maximum tx data rate**.
- **Processing**: In order to collect raw data from nodes and send processed data to nodes without losing data, the required buffers to connect the Data-Processing Module should be taken into account. For this purpose, the designer must define the parameter  $\mathbf{P}$ , which is the number of channels connected between the Data-Flow Module instance and the Data-Processing Module instance in both directions. This parameter contributes to defining the minimum amount of asynchronous FIFOs in the equation 7.1. These additional asynchronous FIFOs are clocked by the **processing clock** that defines the *Data-Processing domain*.

Finally, to define the *minimum number of FIFOs* ( $\mathbf{F}_{\min}$ ) for the CDAS, equation 7.1 has been proposed.

$$\mathbf{F}_{\min} = 2 \cdot \max(N, M) + P \quad (7.1)$$

To use this equation, the following conditions must be met: **tx data rate** > **rx data rate**, and data arrive continuously. This equation represents an upper bound on possible optimizations since it gives the minimum number of FIFO to not lose data. In fact, optimizations can also be made with respect to the *depth* of each FIFO and the ratio between the data rates. In addition to the design parameters explained above, the user sets other parameters relating to the run-time configuration, such as the data acquisition setting and the structure. For example, by setting the acquisition and message parameters, the configurable dataflow architecture manages different packet types, data sizes, data rates, and acquisition periods at run time. For example, in the KIDS-CT scanner, the acquisition scan parameters are configured through the software module based on the part of the body to be scanned, the acquisition time, and the number of projections to be collected per minute. In this application, parameters consist of the exposure time, the number of slices per projection (shot), the size of data, and the sampling period per image to acquire.

As shown in Fig. 7.8, at each stage the dataflow architecture contains registers that are used to set the run-time parameters. Each stage has its own internal datapath and a control unit, which is configured by these registers. In addition, these registers are mapped in a unique contiguous global memory region, which is accessed by the PS via

an AXI4-Lite interface. At the software layer, the related software module accesses these registers, sets them up before starting the acquisition, and can also read their status during the acquisition.

These registers are used, for example, to set architecture parameters, such as the clock frequency of the various clock domains, and to enable the integrity packet module that generates the retransmission signal used by the Control-Synchronization Module. For instance, in interventional CT procedures, the retransmission mechanism is essential because surgeons cannot afford to lose acquisition data and repeat acquisitions multiple times, so managing transmission errors in the Data-Flow Module helps avoid additional acquisitions and save X-ray doses. These reconfigurability options allow the implementation of custom and standard protocols and interfaces that facilitate the integration of new slave-client nodes into the CPS, contributing to its plug-and-play capability.

#### 7.4.2 Inter-clock domains

As introduced above, each transceiver needs a system clock and a reference clock and generates a data clock. To manage the collected data, related works have proposed solutions that use external memory to compensate for the different clock frequencies between the receiver and the transmitter. In contrast to them, this work proposes a solution where data cross the different clock domains without using external memory, taking into consideration the different data rates to avoid data loss.

The *reference clock* signal is vital in high-speed communication, which is usually used for acquiring data in real time. This clock is used to adequately sample, covert the analog signals into bits and de-serialize them according to a defined data width. By modulating and phase-shifting the *reference clock*, the *data clock* is generated. This last clock is used to stream the data in the datapath. Since a data clock is generated from a reference clock that is aligned with the specific acquiring data, each data stream has an independent *data clock* that is driven by the reference clock. Each data clock drives all the synchronous logic in the datapath traversed by the associated stream of data. Consequently, each *data clock* defines a clock domain.

The management of these clock domains is essential for on-the-fly data-processing and real-time support because such a solution allows for avoiding external off-chip memory, which has higher latency access and lower confidence of the WCETs than on-chip memory (i.e., BRAM, Static Random-Access Memory (SRAM), FFs).

As shown Fig. 7.8 with different colours, five clock domains have been used in the dataflow architecture:

- *Receiver domain*;
- *Transmitter domain*;

- *Data-processing domain*;
- *Register domain*;
- *System/Reference domain*

In addition, if two or more receivers/transmitters have different data clock frequencies, different *clock domains* are defined between each other.

### 7.4.3 Architecture description

The dataflow architecture consists of a pipelined datapath at the hardware layer (i.e., PL), which is controlled by a software module running on the PS. In the architecture, each stage is also internally pipelined and can be fully implemented using on-chip resources that have a deterministic delay. Therefore, the time taken to collect, prune and send data can be estimated with high confidence because the only stochastic execution time is given by the round-robin scheduler to write/read data in the correct asynchronous FIFO. As shown in Fig 7.8, the following five stages have been defined.

#### Receivers stage

This stage receives the data from external devices. It supports multiple receivers that depend on the physical channels of external devices. Here, the data are sampled, aligned, and put inside packets that the Data-link parser must interpret. There are three kinds of input clocks: the *register clock* that is driven by the PS and used to control/monitor the stage components; the *system clock* used to control the transceivers; the *reference clock* used for acquiring and aligning data in the transceivers. Based on the *reference clock* and the data rate, each transceiver generates a *data clock*. This new clock drives the synchronous circuit of the *Data-link protocol stage* and part of the *Scheduler stage*.

#### Data-link protocol stage

In this stage, the data-link layer takes the data from the *Receiver stage*. There are separated instances per transceiver because each of them generates an independent data stream. Each instance processes the valid data to send out to another node or to the Data-Processing Module. The *Data-link protocol stage* is internally divided into three sub-stages. The first sub-stage parses the *identifier packets* from each message in the data stream. Based on the sequence of values in the *identifier packets*, it classifies the data as header, body, and integrity packets and generates the corresponding request to the Control Unit. In the second sub-stage, the Control Unit performs the operations based on the parsed packet type. In order to have run-time configurability support the Control Unit is set up via PS registers for the different acquisition modalities and protocols. Finally,



the third sub-stage checks the data integrity and forwards them to the Data-Processing Module (out of the dataflow architecture) in the CDAS and/or to the *Scheduler stage* for transmitting data to another node. In case of errors, the third sub-stage flushes FIFOs in the *Scheduler stage*, containing the corrupted message. In addition, it sets the re-transmission bit in the controlling interface, which forwards a re-transmission request to the node that is transmitting the data.

### **Scheduler stage**

The scheduler stage is the core of the datapath. In this stage, data traverse from one into another clock domain (i.e., Clock Domain Crossing (CDC)) using asynchronous FIFOs. A dual clock reading module is used to allow the PL to access the registers mapped into the PS at different clock frequencies. Moreover, this stage has three internal sub-stages: the *Allocator sub-stage*, the *FIFO sub-stage*, and the *Select packet sub-stage*. Each data request arrives in the *Allocator sub-stage*, which assigns the data to an empty asynchronous FIFO using a round-robin scheduler. Along with the request, the related data are merged, prepared, and queued in the assigned empty asynchronous FIFO. These data come from the *Data-link protocol stage* or the external Data-Processing Module with different data rates. The data are collected in the *FIFO sub-stage* and are committed when the last packet of the message arrives, and the previous stage checks its error packet. After that, if there is an error, the entire message is deleted by flushing the FIFO in the *FIFO sub-stage*; otherwise, it is committed by sending a commit signal to the *FIFO sub-stage*. The *FIFO sub-stage* contains asynchronous FIFOs and modules that convert the signals from the receiver clock domains to the transmitter clock domains. Finally, the *Select packet sub-stage* schedules all the valid request signals with a round-robin algorithm. This last stage is driven by the *data clock* of the corresponding transmitter in the *Transmitter stage*.

### **Streaming generator stage**

This stage prepares and packages messages of the defined sender protocol for transmitting them to another slave-client node. It also generates the synchronization command when there is no data to transmit. For example, synchronization commands are continuously streamed between two messages in the proposed Application Stream Protocol described in Section 6.3.3. This functionality allows the simplex mode communication between the sender (master) and the receiver (slave) to be kept stable. When the *Scheduler stage* sends a message to transmit or store, it adds the appropriate *identifier packets* according to the protocol and the message type to handle. In addition, we can have multiple instances of this stage, one per physical transmitter. If the CPS needs PCI-E transceivers, it generates the messages according to the PCI-E protocol. This stage can be configured at design

time for packeting data with different message protocols. Inside, there is a look-up table, which can be updated with the different *identifier packets*. Based on the type of message to be sent, these packets are selected using an FSM.

### **Transmitter stage**

This stage controls the physical transceivers and forwards data to them. Each transceiver generates a *tx data clock* that drives units within the *Select packet sub-stage* and the *Streaming generator stage*. Furthermore, it contains the transceiver instances that are configured and mapped at design time. At run time, the user can update the data rate and the *tx data clock* according to the *reference clock* and the transceiver reconfigurability. For instance, the Data-Flow Module of the CDAS architecture for the KIDS-CT scanner uses only one Transmitter domain because all the transceivers have the same data rate and configuration, but different transceiver configurations are supported, where different Transmitter domains can also be defined.

## **7.5 Data-Processing Module**

This Section describes the Data Processing Module, which is designed as a configurable pipelined dataflow architecture. As shown in Fig. 7.8, this module receives data from the scheduler stage of the Data-Flow Module, then processes and forwards them back. This module is strictly dependent on the data processing algorithm to be performed by the targeted CPS application.

In order to achieve a generic architecture that contributes to the research objectives, this thesis proposes an architecture that can be configured to process custom and standard data formats and integrates processing algorithms that are also implemented within a dataflow processing core.

A processing core must use AXI4-Stream interfaces to receive data from the other stages and be integrated into the proposed Data-Processing Module. In addition, a processing core can be implemented by using a Hardware Description Language (HDL), a HLS, or an IP core design flow. In addition, the architecture can be used to explore the design space considering the data format options. In order to isolate different critical components and application domains within the CDAS, various instances of the Data-Processing Module can be implemented and properly connected to the associated Data-Flow Module. To simplify the architecture description, the following Sections refer to a single instance.

### **7.5.1 Architecture description**

The Data-Processing Module consists of the following three main stages shown in Fig. 7.9:

- *Sensor-data conversion stage*: This stage receives raw sensor data from one or more Data-Flow Modules. It converts each value to the selected processing data format. It must be configured at design time for the processing data format (e.g., integer, floating point, fixed point).
- *Processing stage*: At this stage, the values are ready to be processed in the selected data representation. It is designed to integrate the dataflow core, which implements the processing algorithm of the selected application. In order to facilitate the data communication with the existing IP core, the architecture provides the AXI4-Stream interface protocol, which is the de-facto standard for on-chip streaming communication.
- *Post-processing conversion stage*: This stage receives the processed data and converts them to the selected format, which is defined at design time. The output results are ready to be sent to another slave-client node or displayed. Consequently, they are sent back to the Data-Flow Module, which is responsible for their transmission.

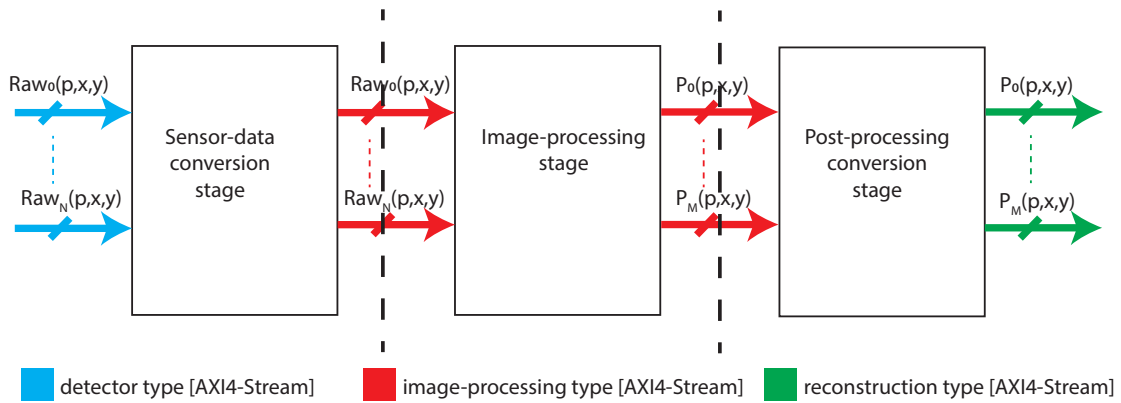


Figure 7.9: CDAS: Data-Processing Module

As explained above, the Data-Processing Module uses AXI4-Stream as input/output interface and between the internal stages. It facilitates the integration of new processing cores into the architecture because this versatile interface offers adjustable data widths, making it adaptable to various data representations and facilitating the DSE of the data formats. Thanks to the tunability of different data formats, this architecture has been used to explore the design space and find better data formats for CT pre-processing, taking into account image quality, execution time, and resource utilization. This process will be explained in Section 9.5, together with all the proposed optimizations for the CT use case.



## 7.6 Isolation Support For MPSoC-FPGAs

In order to isolate the modules presented in the previous Sections, this thesis proposes an isolation method that has led to a lightweight architecture for low-cost MPSoC-FPGA, named Lightweight Protection Unit. As shown in Fig. 7.2, the LPU is also part of the CDAS architecture for MPSoC-FPGA. It is crucial because in CPSs with MCS requirements, such as the KIDS-CT scanner, these modules contain hardware units and software modules belonging to various external components or application domains. Besides the CDAS architecture, this system can be implemented on different architectures utilizing AXI4. It also targets a wide range of devices from small embedded to high-performance FPGA and MPSoC-FPGA to guarantee isolation in a MCS application. In order to protect a desired memory space or peripheral from different masters (e.g., CPU, PE) trying to access it, the proposed solution exploits the on-chip communication and blocks transactions that are considered illegal based on defined policies.

### Proposed method

The proposed isolation method must result in a lightweight architecture that can be integrated into existing architecture for small CPSs and/or their components. In addition, it must not affect the transaction time with stochastic delay, to meet the real-time requirements. To meet such requirements, the decision path responsible for granting/denying a transaction must be combinatorial. Consequently, the isolation mechanism is based on the following three elements:

The proposed isolation method is based on the following three elements:

- **Protection Domain (PD):** This consists of a group of master components within the same application domain. Notably, a single master component may be a member of several PDs. A PD is characterized by a *domain ID* and a *domain mask*, which are used to match AXI transactions belonging to a master with PD. Also an AXI transaction contains the AXI ID, which is usually defined at design time. For masters, where the AXI ID is not defined by default, an additional component will be used between the AXI master interface and the AXI interconnect.

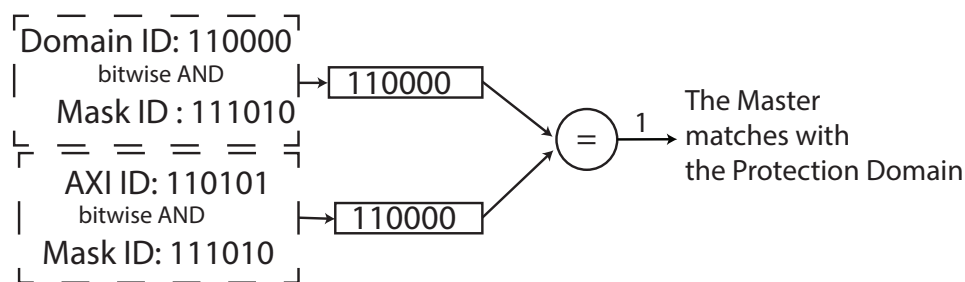


Figure 7.10: Matching steps between AXI ID and domain ID

Given that domain IDs and AXI IDs can vary in size, and an AXI ID may be associated with several PDs (and vice versa), the domain mask allows for comparison and identification of the PDs to which a master belongs at run time. Since a domain ID and an AXI ID can have different sizes and an AXI ID can be associated with different PDs and vice-versa, the domain mask permits to compare them and identify which PDs are belong to a master at run time. In detail, the domain ID bits that are not masked by their domain masks are compared to the corresponding unmasked bits of the AXI ID. If they all are equals, the transaction is part of that PD, as shown in Fig. 7.10. For example, Table 7.1 shows that the master ID *1011* belongs to domains 0 and 2.

Table 7.1: Example of AXI ID associated with different PDs

	Domain Mask	Domain ID	exemplary IDs
PD 0	1100	<u>1000</u>	<u>1011</u> , <u>1000</u>
PD 1	1110	<u>1000</u>	<u>1000</u>
PD 2	1110	<u>1010</u>	<u>1011</u>

The primary benefit of this matching solution is its efficiency in terms of time and resources because it can be implemented in the LPU with simple combinatorial logic.

- **Memory Region:** An Memory Region is defined as an aligned address space. Similar to how masters can belong to various PDs, a peripheral and a memory address may simultaneously associate with multiple Memory Regions (MRs). Each MR is delineated by a *starting address* and its *Most Significant Bit (MSB)*, which includes the addresses spanning the entire MR. In fact, an MR always comprises contiguous addresses.

To enhance the execution time of the MR matching step and the resource utilization of the LPU, the position of the Least Significant Bit (LSB) is used as a parameter. By using the LSB as a defining parameter, the LPU only needs to inspect the range of bits nestled between the MSB and the LSB for each input address. Furthermore, the LSB position is also used to define the size of the memory region that is equal to  $2^{LSB\ pos}$ . An example of this matching is shown in Fig. 7.11.

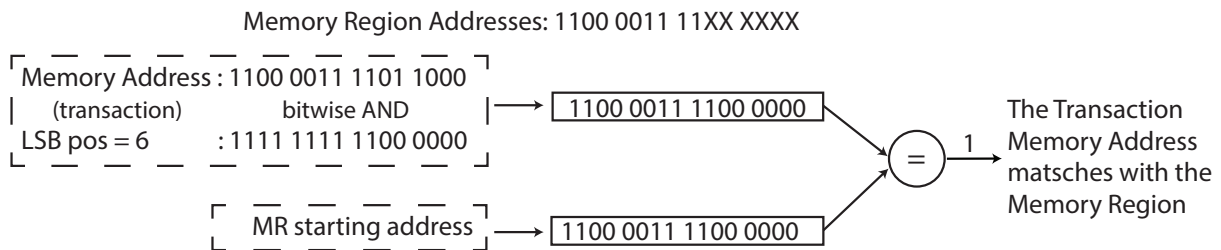


Figure 7.11: Matching steps between a Transaction Memory Address and MR

- **Access Policy (AP):** Each one defines the read/write access rights of masters in a PD to addresses in an MR. It is composed of rules in the format: “*PD\_X is allowed to read/write MR\_Y*”. A transaction is granted access if at least one of these rules permits it. If a master isn’t linked to any PD, its requests are automatically rejected. Differently from other solutions in the literature that store the APs of different LPU in a centralized database, the proposed solution will use a look-up table for write and read in each LPU to store the associated APs. In this way the grant bit in the AP can be read every clock cycle as an input to the combinatorial logic that will implement the decision path.

PDs and MRs, along with their parameters, are pre-determined during the design phase. Conversely, AP are established during run time. This methodology ensures that each LPU is optimized during the synthesis phase and that the entire decision path is implemented using the combinatorial logic. The only variable in this equation is the incoming AXI transaction and the access rules of the AP. It’s crucial that the APs are set at run-time to ensure that temporal isolation is maintained.

### 7.6.1 LPU architecture

As explained above the proposed isolation method exploits is applied on the AXI interface. Therefore, the proposed LPU must be instantiated between masters and slaves, in the PL part of the MPSoC-FPGA. In this way, each AXI transaction crosses a LPU before reaching a slave. To do it, the designer can decide to instantiate and connect it on the master side or on the slave side and to have single or multiple instances.

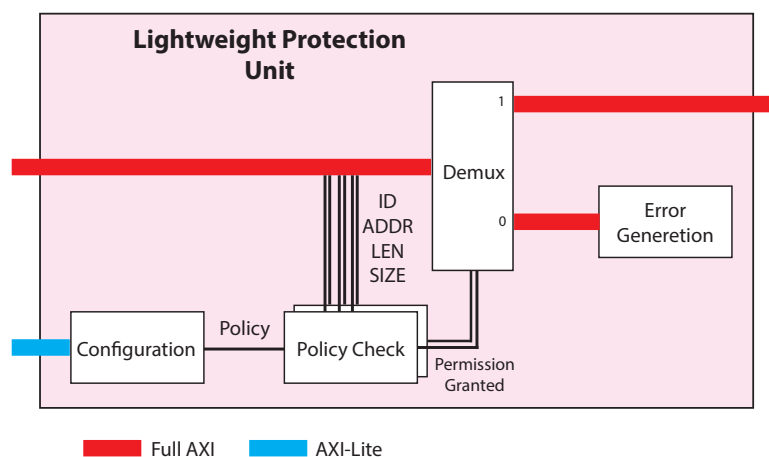


Figure 7.12: Architecture of the Lightweight Protection Unit

As shown in Fig. 7.12, a LPU has two types of AXI interfaces: the AXI-Lite and the AXI-Full which are depicted in blue and orange, respectively. The AXI-Lite interface is used to read and write controls and status registers, as well as the LPU look-up table. In contrast, the AXI-Full interfaces connect the master and slave interfaces, where isolation

must be provided. Here, the granted transactions pass through the LPU without being affected by it, while the rejected transactions are blocked and terminated.

The LPU architecture consists of the following units:

- **Configuration:** It contains the registers that are configured at setup time to enable the isolation mechanism and to set the AP. The run-time configuration permits to realize also the temporal isolation. In addition, it forwards the values written in the policy look-up table to the Policy Check unit as signals. In this way, the policies can be read every clock cycle as input signals from the Policy Check unit.
- **Policy Check:** This is the core of the LPU and it is responsible for generating the valid bit for the granted/rejected transaction. Each LPU has two instances of the policy check unit that are responsible for read and write channels. Every clock cycle this unit reads the master ID (AXI ID) and the incoming memory address of the current AXI transaction and matches the associated PDs and MRs according to the step described above. In this Policy Check unit, there is one PD matcher and a MR matcher per PD and MR, respectively, as shown in Fig. 7.13. For this reason, the number of PD and MR defined at design time will also affect resource utilization. The results of this matcher are used to read the correct value in the policy look-up table, and if there is at least one APs among the associated PDs and MRs, the permission is granted. In the example, shown in Fig. 7.13, the AXI-ID matches with  $PD_0$  and  $PD_1$ , and the memory address of the AXI transaction matches with  $MR_0$  and  $MR_1$ . Since the AP for  $PD_1$  and  $MR_1$  is equal to 1, the transaction is granted.
- **AXI-Demux:** It forwards the AXI transactions either to the Error Generator or outside the LPU. A transaction is forwarded on the base of the valid signal of the permission granted, which is connected to the selector signal of this Unit. The AXI-Demux unit differs from a simple demultiplexer, because it implements all the logic to handle all signals of an AXI transaction, and can also be set to buffer transactions, as explained in Ref. [155].
- **Error Generator:** It has received the AXI transactions that were rejected by the Policy Check. Such transactions cannot simply be blocked, because the communication would be stalled, as explained in Ref. [155]. Therefore, they are forwarded to this unit that asserts an error as defined by the AXI standard.

### 7.6.2 Example

The following Section illustrates an example, where a LPU has been instantiated and connected with two masters and two slaves, as shown in Fig. 7.14. This standalone example illustrates how a LPU can be placed in the design and aims to clarify how it derives decisions at run time.

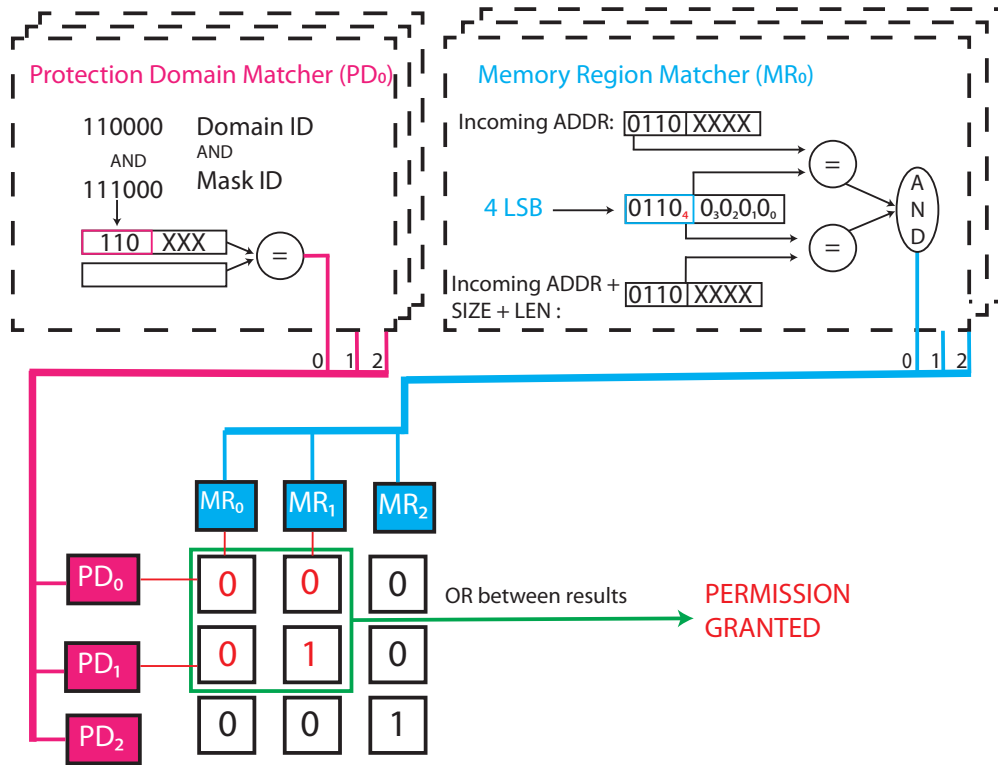


Figure 7.13: Policy check functionality

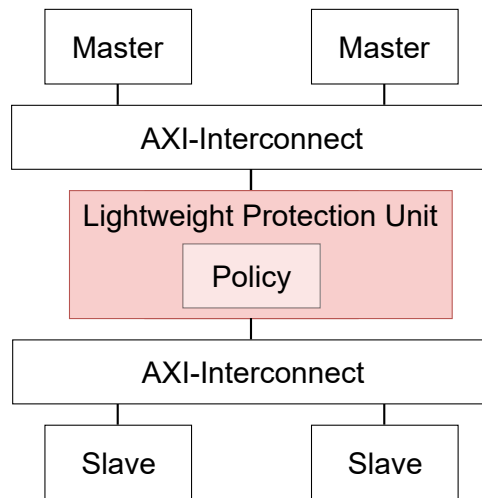


Figure 7.14: Exemplary of LPU placement

Starting from the design in Fig. 7.14, the example assumes that the first master belongs to the  $PD_1$  and  $PD_0$ . Rather, the second master belongs to the  $PD_2$  and  $PD_0$ . In the same way, the address space of the first slave is included in  $MR_1$ , and the second one in  $MR_2$ , while both are associated with the  $MR_0$ . These associations are defined at design time, while the AP is set at run time.

According to the proposed isolation methodology, a master can access a slave memory space, if there exists an AP that encompasses both a PD and an MR to which they belong. In this example, as shown in Fig. 7.15, the user has defined the following three AP: the

$PD_0$  can read the  $MR_0$ , the  $PD_1$  can write the  $MR_1$ , and the  $PD_2$  can write the  $MR_2$

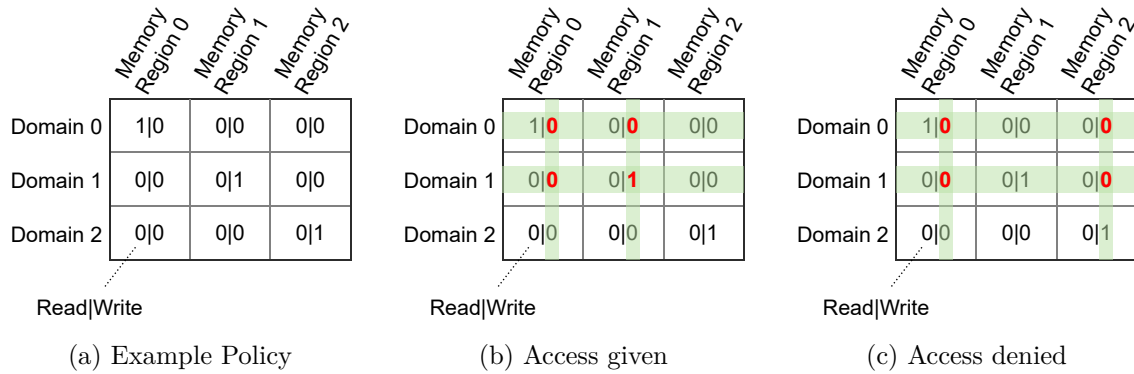


Figure 7.15: Exemplary Policy Configuration and Decision

When the first master attempts to write to the first slave, both  $PD_0$  and  $PD_1$  match the master, and  $MR_0$  and  $MR_1$  match the slave's memory address. The matching PDs and MRs activate the green lines in the look-up table shown in Fig. 7.15b. As at least one of the entries marked in red is equal to one, the permission is granted. Instead, when the same master tries to write to the other slave associated with  $MR_0$  and  $MR_2$ , all the red-marked entries are equal to zero, so access is denied.

**Part III**  
**KIDS-CT**






## 8 System Architecture For The KIDS-CT Scanner

This Chapter describes the realization of the proposed System Architecture and Communication Infrastructure applied to a CPS use case: the KIDS-CT scanner assembled in the context of this research work. Section 8.1 matches the CT task requirements with the classification proposed in Section 6.1, following the requirement definition steps presented in Section 5.1. This process allowed the author to identify the *common* and *specific* elements (i.e., requirements) between diagnostic and interventional CT procedures. Finally, Section 8.2 and Section 8.3 explain the realization of the System Architecture and the related Communication Infrastructure in the specific case of the KIDS-CT scanner. The content presented in this Chapter has been published in Ref. [DP 2, DP 3, DP 4].

### 8.1 CT Requirement Classification

In order to have a CPS that is scalable and extensible, it is essential to identify the *common* and *specific* elements, starting from the application modes/functionalities down to the associated design requirements for the hardware/software architecture of the ES within a CPS device.



STEP 1	STEP 2		STEP 3	
MEDICAL PROCEDURES	MEDICAL REQUIREMENTS		DESIGN REQUIREMENTS	
Mode: Diagnostic CT	Diagnosis; High resolution 3D image;	X-Ray dose minimization; Parameter tuning; <b>open- interface</b> for exploration of new medical procedures; (capability to add other components such as Time- of-fly cameras, Ultrasound sensors, DMSs and X-ray Tubes);	Off-line acquisition and reconstruction; High data precision formats;  <b>Data Acquisition and Collection &amp; Data Processing:</b> <b>Mixed data-precision formats; Real-time data acquisition; Real-time data processing;</b>	<b>Control and Synchronization:</b> Safety control units; <b>Real-time control-synchronization;</b> Custom control protocols; Global timestamp; <b>Plug-and-play capability</b> at communication, system and control level;  <b>Data Acquisition and Collection:</b> Custom data link protocol; Simplex data communication; Low latency inter- communication; High speed data communication; <b>Plug-and-play capability; On-the-fly acquisition; Multi-sensor acquisition;</b>  <b>Data Processing:</b> Custom architecture for pixel processing and reconstruction; Huge memory space; <b>Plug-and-play capability; On-the-fly acquisition; Support to mixed-data-precision processing</b>
Mode: Multimodality CT for Interventional  <b>Exploration of medical procedures</b>	Surgery and Diagnosis; Live reconstruction 3D image; Live inter- action between: Patient's body/Doctor /Scanner; Live update input parameters; <b>Real-Time image pro- cessing; Real-Time controlling;</b> Live X-ray current update; Mul- tiple systems data- merge; Sub-systems live interaction;			

Figure 8.1: Definition of the KIDS-CT requirements

Fig. 8.1 shows the steps to define these requirements for the KIDS-CT scanner, where the medical procedures are considered as the application/functionality mode of the CPS. By applying the explained steps, the medical and design requirements have been identified, starting from the medical procedures. Specifically, the red text in the figure highlights the requirements that are typically not met by commercial scanners but are met by the KIDS-CT scanner through the research objectives and outcomes of this thesis. Furthermore, these design requirements have been grouped and associated with the three main task groups identified in Section 6.1 for the proposed CDAS architecture: the Control and Synchronization, the Data Acquisition and Collection, and the Data Processing groups.

## 8.2 System Architecture

In the KIDS-CT scanners, the X-ray tube, the DMS, the gantry, the patient table, and the collimator system are sensor/actuator components, while the user interface system and the reconstruction system are interface/processing components. Following the model of the proposed System Architecture, all these components are independent slave-client nodes connected to the master-server node (i.e., CDAS). Using this solution, an additional component (e.g., DMS, X-ray tube) is also modeled as a new slave-client node to be connected to the master-server node for being integrated into the scanner.

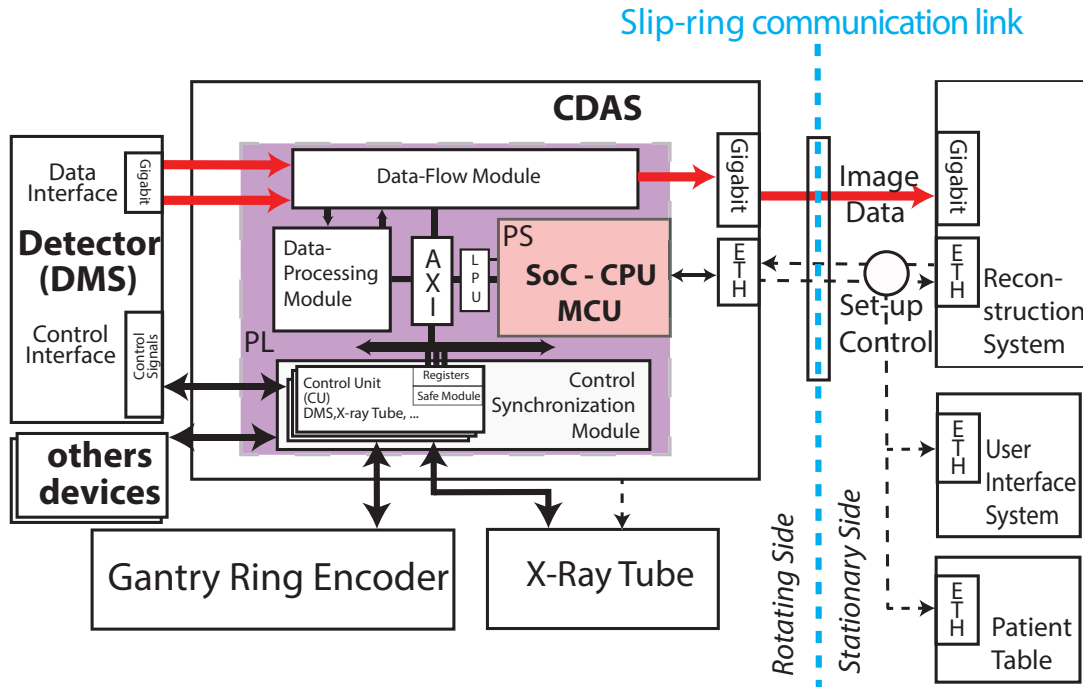


Figure 8.2: KIDS-CT System Architecture: Physical level

Fig. 8.2 shows the *physical level* model, with the mentioned nodes and connections in the scanner. Due to the physical limitations of connecting components between the rotating side (on the left side of the figure) and the stationary side (on the right side of the figure),

the connection between the CT components and the CDAS is critical in this application. Since communication between the rotating and stationary sides is limited by the number of links and the data rate that the associated transceivers can achieve, optimizing these two constraints will also reduce the cost of the device. The red arrows show the data flow from the DMS to the reconstruction system.

During the modeling of the System Architecture, it is already possible to optimize both constraints by placing the master-server node (i.e., CDAS) on the rotating side. In this way, the number of connections and the amount of data to be transmitted via the slip-ring communication link with the CDAS can be reduced. In addition, all the slave-client nodes can be easily connected to the CDAS without crossing the gantry, and the slip-ring communication links are only used to transmit the control commands with the user interface system and the processed data with the reconstruction system.

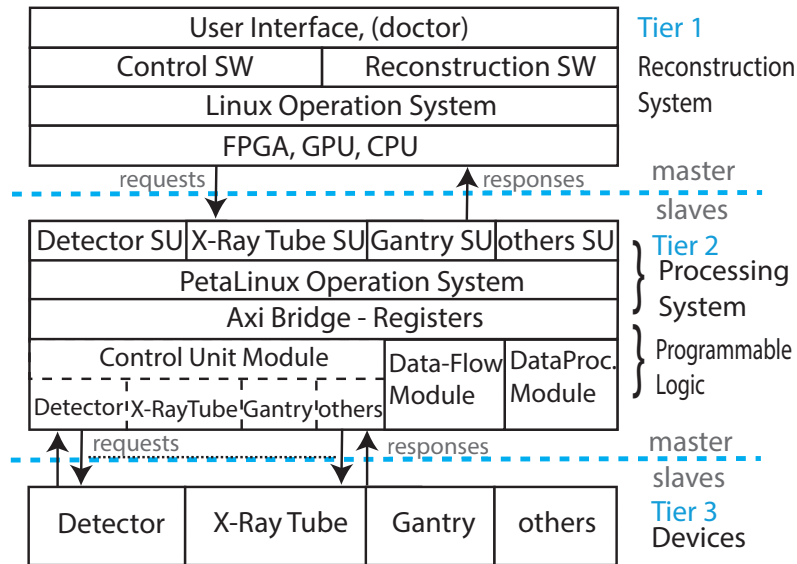


Figure 8.3: KIDS-CT System Architecture: Logical level. Copyright 2020, IEEE

At the *logical level*, the System Architecture for the KIDS-CT scanner has been modeled according to the proposed multi-tier architecture. As shown in Fig. 8.3, Tier 1 contains the user interface that allows access to the control software and the reconstruction software; the physician sets the acquisition parameters and the reconstruction parameters from this Tier, which is responsible for encoding these parameters and sending them to the CDAS. In Tier 1, there is no direct connection to sensors/actuators such as the X-ray tube and the DMS, which are only accessed by Tier 2 (i.e., the CDAS). Tier 2 receives the functional parameters coming from Tier 1 and updates the local sensor/actuator parameters to be sent to Tier 3. Finally, the sensors/actuators, such as the X-ray tube and the DMS, the gantry, are placed in Tier 3. The presence of Tier 2 enables the containment of sensor/actuator failures and blocks access by attackers originating from the user interface in Tier 1. In fact, this interface may also be accessed through the World Wide Web to reach sensors/actuators in Tier 3. Therefore, in case of failure, the CDAS is responsible

for containing the issue and setting the rest of the system into a safe mode.

### 8.3 Communication Infrastructure

Based on the System Architecture, the CT scanner components have been interconnected according to the proposed Communication Infrastructure as shown in Fig. 8.4. This depicts the interconnection of the nodes and their interfaces for the different classes. Associated with the non-real-time class type, each node uses an Ethernet port at the *communication interface layer* and IPv4 protocol at the *transport protocol layer*. For nodes in the *real-time data class*, the data between the master-server node and the other nodes has been transferred over fibre channel, using GTXs at the *transport protocol layer*, which can be set for different encodings and protocols.

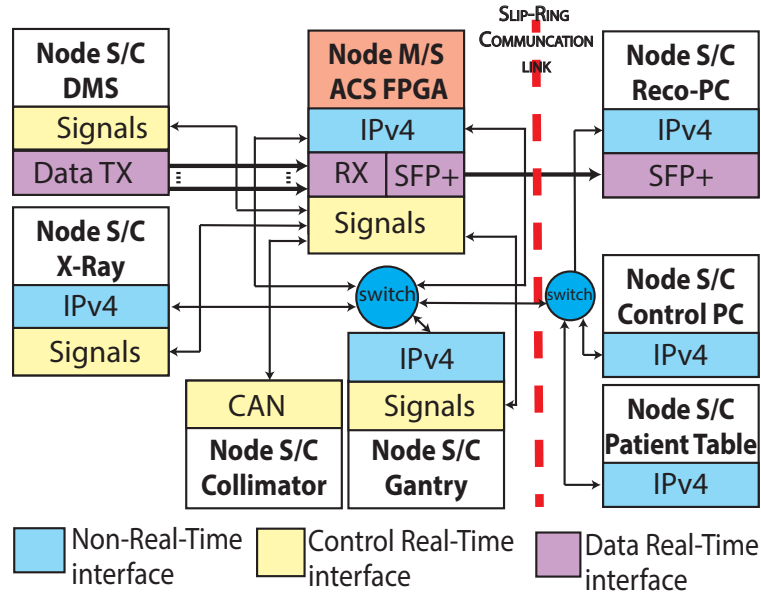


Figure 8.4: KIDS-CT: Communication Infrastructure. Signals refer also to custom vendor protocols

For the *real-time control class*, which includes communication between the master-server node and the X-ray tube, DMS, collimator, and gantry nodes, custom interfaces and custom transport protocols provided by vendors are used. To establish this communication, an extension board was designed and built. This is connected via a FPGA Mezzanine Card (FMC+) interface using the “VITA 57.4 FMC+ standard”, which allows the connection of custom vendor interfaces to the General Purpose Input Output (GPIO) pins. At implementation time, the GPIO pins are mapped to the PL of the MPSoC-FPGA as digital I/O asynchronous signals. These signals are connected to hardware units that implement transceivers of the appropriate custom transport protocol, also provided by the vendor. Fig. 8.5 shows the KIDS-CT scanner with components and their connections to the master-server node (i.e., CDAS). On top of the Transport Protocol layer, the Handshake

Protocol, the Application Datagram Protocol, and the Application Stream Protocol have been implemented consisting in the *application protocol layer*. While the lower layers have different protocols per class, this layer unifies all the different data coming from these protocols to the corresponding proposed application protocol belonging to the class. This solution facilitates inter-component communication where different protocols are used at the *transport protocol layer* and the *communication interface layer*. For example, the DMS, the X-ray tube, and the gantry use different vendor-specific protocols, which are unified at the *application protocol layer*.

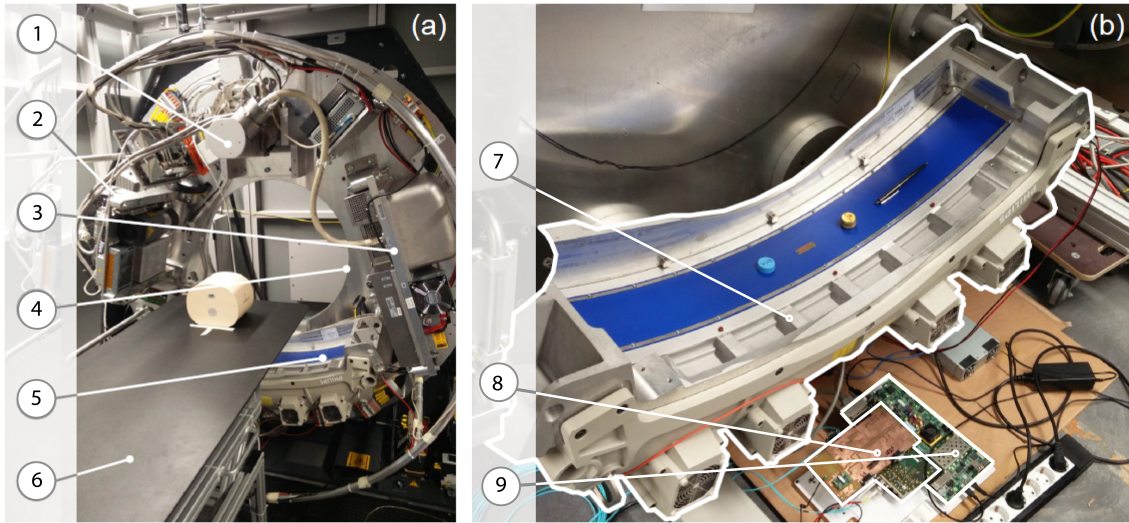


Figure 8.5: Assembled KIDS-CT scanner. (a): Experimental CT complete system with (1) X-ray tube, (2) Cooling system, (3) Generator, (4) Gantry subsystem, (5) Multiline DMS, (6) Patient table. (b): Detailed view of the DMS and CDAS implemented on the AMD-Xilinx ZC706 Evaluation Kit: (7) Multiline DMS, (8) Extension-board including the custom interfaces, (9) AMD-Xilinx ZC706 Evaluation Kit

In addition to the various interconnections, Fig. 8.4 shows the data flow from the DMS to the reconstruction system via the master-server node. In the KIDS-CT scanner, this communication is set at a rate of 6.250 Gbit/sec, and thanks to the fibre channel, the latency is in the order of nanoseconds. This flow is critical for the KIDS-CT scanner to provide real-time support for the interventional procedures. In contrast to related work that acquires and pre-processes data offline during the reconstruction steps, such a solution will allow data to be pre-processed on the fly in the master-server node during acquisition. The Application Datagram Protocol is used to set the various parameters and registers for setting up the various components and to read out the logging data. In the case of the KIDS-CT scanner, in order to identify the different commands and to check that no connection is lost, different packet types have been defined within a message, resulting in the following structure shown in Fig. 8.6.

As shown in the figure above, the Pong packet contains only the associated status and

0 Length	32 Type	64 1. Register/Data	128 1. Data	136 ....	168 N Register/Data....	} Normal
Length	Type	1. Data		....	N Data	
Length	Type	1. Operation Status	2. Operation Status	....	N Operation Status	} Ack Status
Length	Type	1. Register	1. Data	....	N Register	} Ack Register
Length	Type	1. Status Register	1. Control Register			} Pong

Figure 8.6: Structure of four typical message data sections, used in the KIDS-CT scanner

the control registers, so padding bits are required to align and encode it. The Normal packet has a register field and the data field is based on the amount of data to be transmitted, which the server calculates at run time using the Length field within the packet. Additional packet types can be defined when an additional slave-client node is plugged into the KIDS-CT scanner.

### 8.4 Optimization Of The Acquisition And Processing Datapath

In order to use the KIDS-CT scanner for interventional procedures, the data should be collected and processed in real time. For this purpose, existing state-of-the-art datapaths for diagnostic CT have been analyzed and a new solution has been proposed in this thesis to overcome the found limitations, which are explained in Chapter 4. In the related work, most of the CT scanners acquire and store data in a DAS and send them to the reconstruction system only after the acquisition, as shown in Fig. 8.7.

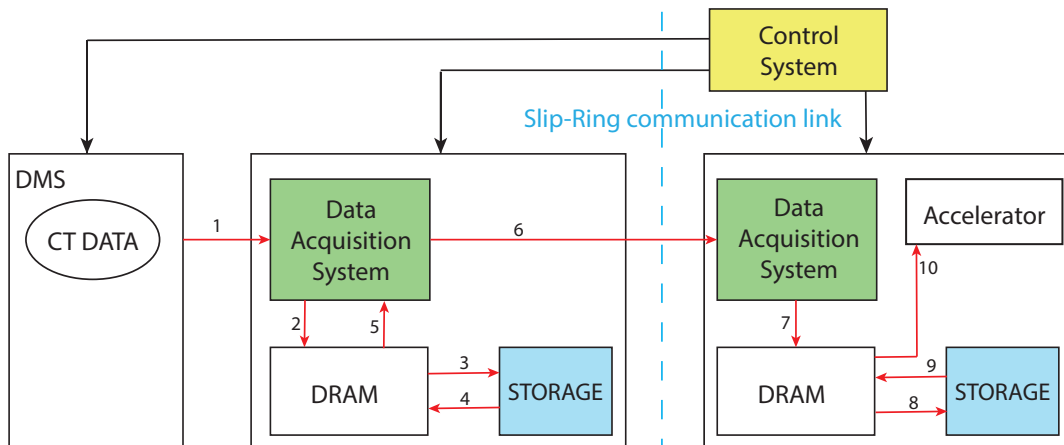


Figure 8.7: Acquisition datapath for CT scanners without real-time support

This figure shows the process where the data are stored in the external memory of the DAS or the reconstruction system. Then, these data are processed offline, on an accelerator such as GPU or FPGA, which is plugged into the reconstruction system.



Unlike related work, to provide real-time support in the KIDS-CT scanner, the data are collected from the CDAS and forwarded to the reconstruction system on the fly, during the acquisition. In this way, the reconstruction algorithm can be executed during the acquisition to display the 3D volume to the surgeon.

Due to the computationally intensive nature of the reconstruction algorithm, running the entire process on a GPU after data acquisition results in significant time loss and performance issues, preventing real-time reconstruction. In order to optimize the acquisition and perform processing steps on the fly during the acquisition, we need to consider the duration of a full rotation, and thus the duration of an *integration period*  $\Delta T$ . This represents the time spent by a pixel sensor to acquire the information and also denotes the interval between two successive projection acquisitions. It is typically about 1 ms in medical procedures, as shown in Fig. 8.8. We can see in the figure that a projection data (e.g. PRS0) is available in less time than  $\Delta T$ ; this is because the projection data transmitted refers to the previous acquisition and it must be transmitted in a time that is less than the current integration period, otherwise, the sensor buffer would be full after a few projections and would be lost during the acquisition.

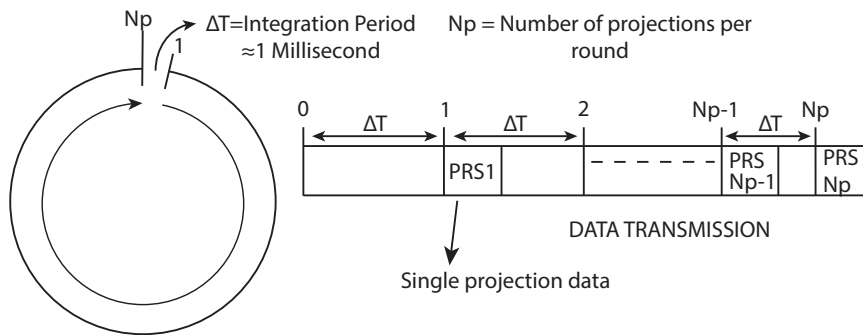


Figure 8.8: Acquisition interval between consecutive projections

Since the CDAS must wait for one *integration period* for each projection to collect, and considering that both the Data-Flow Module and the Data-Processing Module are implemented within a pipelined data-flow architecture capable of collecting and processing data on the fly, an important implication emerges: if the processing time is less than the integration period, it effectively adds zero latency to the overall data acquisition time. Thus, an optimization can already be achieved following this observation.

In order to determine which steps of the reconstruction algorithm are suitable for implementation in a dataflow architecture to integrate into the Data-Processing Module of the CDAS, the algorithm has been segmented into three distinct parts, as depicted in Fig. 8.9. These parts include: the pixel processing part, which is characterized by the absence of dependency between pixels; the slice processing part, which involves data dependency between the nearest row pixels within a projection image; and the voxel part, where there is dependency between pixels across different projections that intersect the

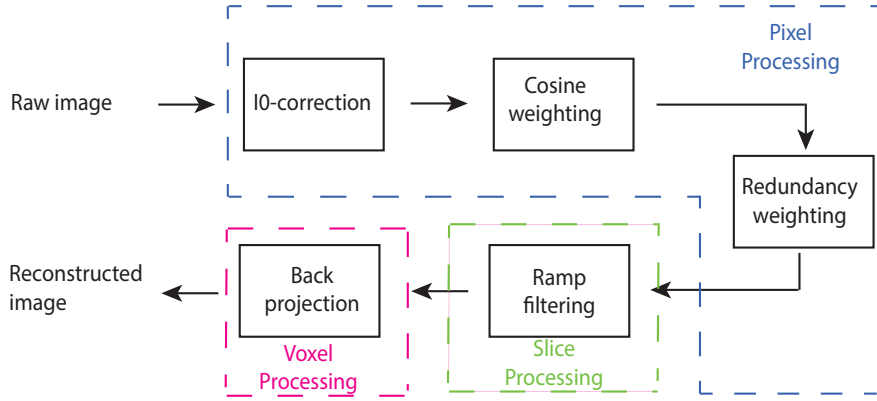


Figure 8.9: Image reconstruction: Algorithm steps

same voxel. Moreover, the lack of data dependency allows the pixel part to be performed on the fly as successive projections are acquired from the DMS. In fact, while the pixel processing part can be performed on the fly without waiting for any data, the slice processing part can be processed when an entire row is collected, and to meet the on-the-fly requirements, all rows within a single projection image should be performed in a  $\Delta T$ . To take advantage of this time when no operations are normally being performed, the CDAS passes the pixel data to the proposed Data-Flow Module, where the pixel processing part of the algorithm is performed within  $\Delta T$ , and consequently, no latency is added in the system.

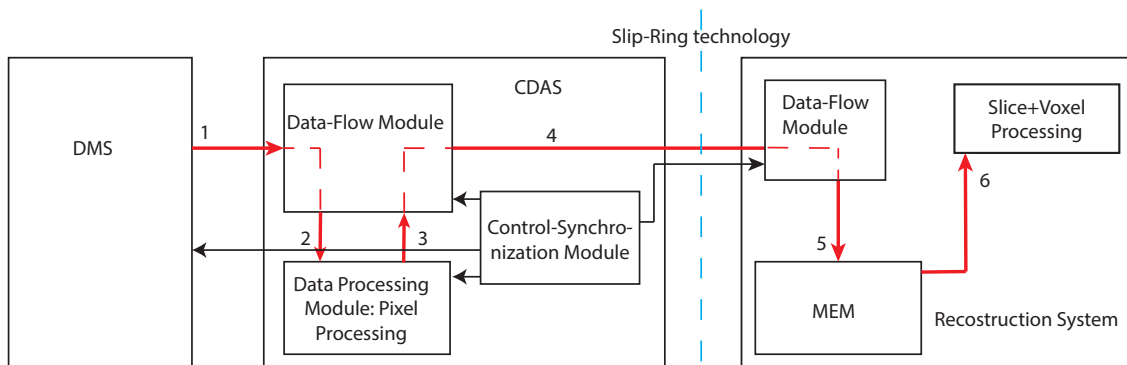


Figure 8.10: Optimized datapath for the KIDS-CT scanner

This solution results in the datapath and the algorithm mapping shown in Fig. 8.10, where data are acquired and forwarded to the reconstruction system, while the pixel processing part is performed without adding latency. The following Chapter discusses the various specific optimizations proposed to improve the performance of the CDAS for the KIDS-CT scanner and compares the final reconstruction time with a solution where the entire algorithm is implemented in a single accelerator such as a GPU.



## 9 Control-Data Acquisition System For The KIDS-CT Scanner

This Chapter provides a comprehensive overview of the CDAS architecture integrated into the KIDS-CT scanner. For this use case, the AMD-Xilinx ZC706 Evaluation Board with the XC7Z045 MPSoC-FPGA model has been selected [156]. The Section 9.1 introduces the hardware/software CDAS architecture implemented in the KIDS-CT scanner, focusing on the mapping of the tasks associated with the various client-slave nodes. Then, Sections 9.2, 9.3 and 9.4 describe the architectural configuration for implementing the Control-Synchronization Module, the Data-Flow Module, and the Data-Processing Module in the KIDS-CT scanner, respectively. Finally, Sections 9.5 and 9.6 illustrate the proposed pre-processing optimizations for the interventional CT application and the DSE applied to the various data formats for achieving the research objectives in the KIDS-CT scanner. The contents of this Chapter are partially discussed in the articles in Ref. [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7].

### 9.1 Hardware/Software Architecture

The CDAS manages all tasks and interactions within the KIDS-CT scanner. Applying the task partitioning proposed in Section 7.1 on the hardware/software architecture proposed in Section 7.2, the resulting CDAS architecture shown in Fig. 9.1 has been defined.

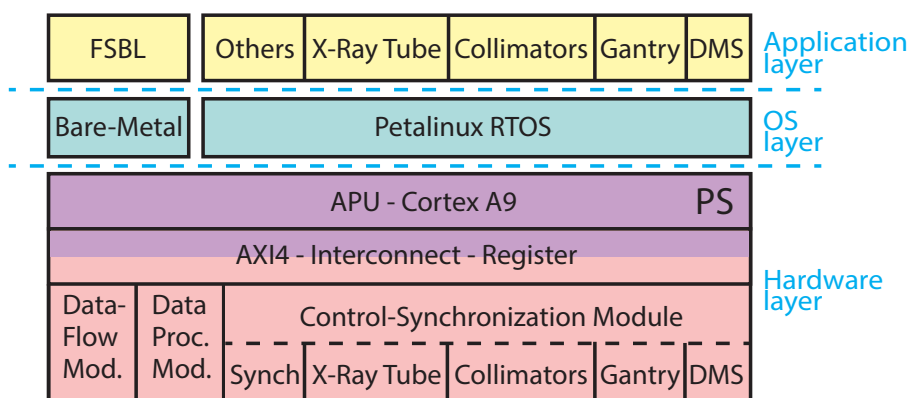


Figure 9.1: CDAS architecture for the KIDS-CT scanner

At the **Hardware layer**, the real-time tasks are mapped to the Control-Synchronization Module and the Data-Flow Module. These are modeled in Register-Transfer Level (RTL) using the SystemVerilog language, which allows the definition of parametrizable generic

modules at design time. Instead, the real-time processing tasks mapped to the Data-Processing Module are described in the C language, and the RTL model was created using Vitis™ HLS. The generated RTL was integrated into the design as an IP core for Vivado Design Suite [157] and implemented in the PL of the MPSoC-FPGA. These modules can also be parameterized at design time.

At the **Application layer**, software modules associated with the various CT components run on the APU, which consists of a dual-core Cortex-A9 [118]. It implements the server for the non-real-time communication tasks. It also implements the business logic of the KIDS-CT scanner. This includes the tasks that coordinate the setup process of the DMS, the X-ray tube, and other nodes. It is also responsible for notifying the control systems of errors that cannot be handled at run time. Due to the lack of an isolation mechanism in the selected MPSoC-FPGA, the proposed LPU has been deployed between the PS and PL parts. This also provides isolation support between PL-PL and PS-PL communication. The software modules at the Application layer configure and coordinate the Control-Synchronization Module, the Data-Flow Module, and the Data-Processing Module of each associated component, which is implemented at the Hardware layer.

At the **OS layer**, FreeRTOS and Petalinux RTOS were implemented, deployed, and evaluated. This allows the designer to choose the OS that best suits its application. FreeRTOS provides a simple OS structure, where kernels are written as bare-metal code without the need for virtual memory or additional layers usually provided by an OS. While tasks running on FreeRTOS access the physical memory directly using the physical address, Petalinux provides a virtual memory with virtual addresses; in this way, the virtual memory is linked to the corresponding physical memory only at run-time, providing portability, extensibility, and compatibility advantages of the software modules. Furthermore, Petalinux supports Unix libraries, which are essential for the plug-and-play capability because they facilitate the integration of vendor software/drivers of additional client-slave nodes. For instance, in the KIDS-CT scanner, the control module for the X-ray tube requires several Unix libraries to run. In terms of timing performance, both RTOSs achieved the same results. Therefore, Petalinux was chosen for the KIDS-CT application, where compatibility and scalability are essential features for providing the plug-and-play capability required for exploring multimodality and interventional CT procedures.

## 9.2 Control-Synchronization Module

The CDAS architecture implements a Control-Synchronization Module for each component (i.e., slave-client node), as shown in Fig. 9.1. While nodes belonging to Tier 1 of the System Architecture have only the Software Module in the implemented CDAS, nodes belonging to

Tier 3 have Software and Hardware Modules because they use custom real-time protocols and synchronization mechanisms that must be implemented in the PL. Hardware and software modules belonging to the same node, communicate with each other via an AXI4 interface.

### 9.2.1 Software architecture

This module contains the software architecture of the non-real-time tasks associated with the business logic, the server, and the Application Datagram Protocol for the *application non-real-time class*. The business logic is implemented in C and it is the main application that executes the others as subtasks. The server is also implemented in C, following the architecture presented in Section 7.3. It is configured using the configuration file, which contains the priority hierarchy, the maximum number of connections to the server (i.e., connected clients), and the parameters associated with each node. They consist of the CDAS static IP address and port of the node, the `TIME_OUT` value, and the packet structure for the Application Datagram Protocol. As shown in Fig. 8.4, in the KIDS-CT scanner, only the CDAS and five other nodes use the client-server communication, so the maximum number of client connections in the “Communication module” within the server is set to 5. Splitting the connections into different sockets provides an independent endpoint for each connection. By setting the maximum number of clients, unexpected connections and/or malicious clients attempting to connect to the server will be rejected. In addition, when an authorized client attempts to access a socket associated with another client, the KIDS-CT scanner is put into safe mode to prevent unnoticed operations. This means that the DMS is disabled, the high voltage in the X-ray tube is disarmed, the gantry rotation speed is decreased until it stops, and the software modules are set to default parameters while the error is reported. The `TIME_OUT` parameter also has a critical function for the safety requirements of the KIDS-CT scanner. In fact, to guarantee that the connection to a node such as the X-ray tube is not lost, the `TIME_OUT` of the Timer module is set to 2 seconds for all nodes. Finally, the scanner is put into safe mode if a `TIME_OUT` event occurs.

### 9.2.2 Hardware architecture

The Hardware architecture implements the real-time tasks for controlling and communicating with the nodes at Tier 3, such as the DMS and the X-ray tube. The Communication Units associated with these nodes are also implemented in SystemVerilog. Since each node has its own custom vendor protocol, a custom transceiver unit must be used for each protocol, which has been implemented in the associated component instance for that module, following the proposed solution presented in Section 7.3.

In addition to the Communication Unit, this module contains the Synchronization Unit, which is responsible for coordinating the DMS, the gantry, and the X-ray tube during

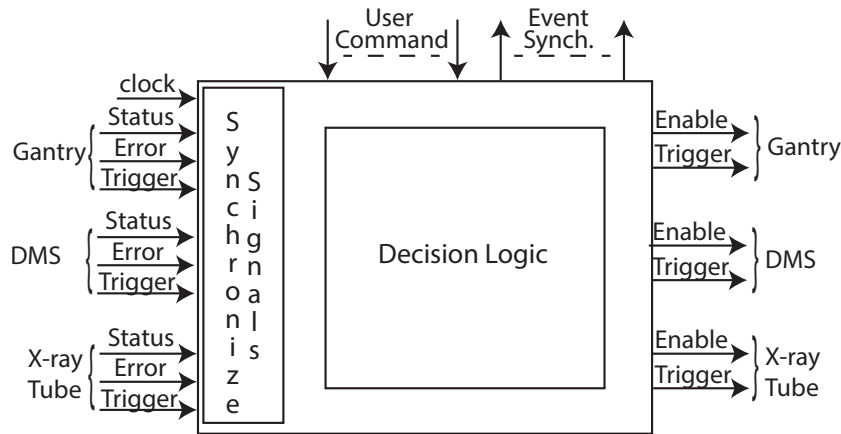


Figure 9.2: KIDS-CT Synchronization Unit

the acquisition. This unit, as shown in Fig. 9.2, gets as input asynchronous signals from the nodes and synchronous commands (e.g., user commands) from the business logic implemented in the PS, and generates the enable and the trigger signals to synchronize the operations, such as making a CT scan.

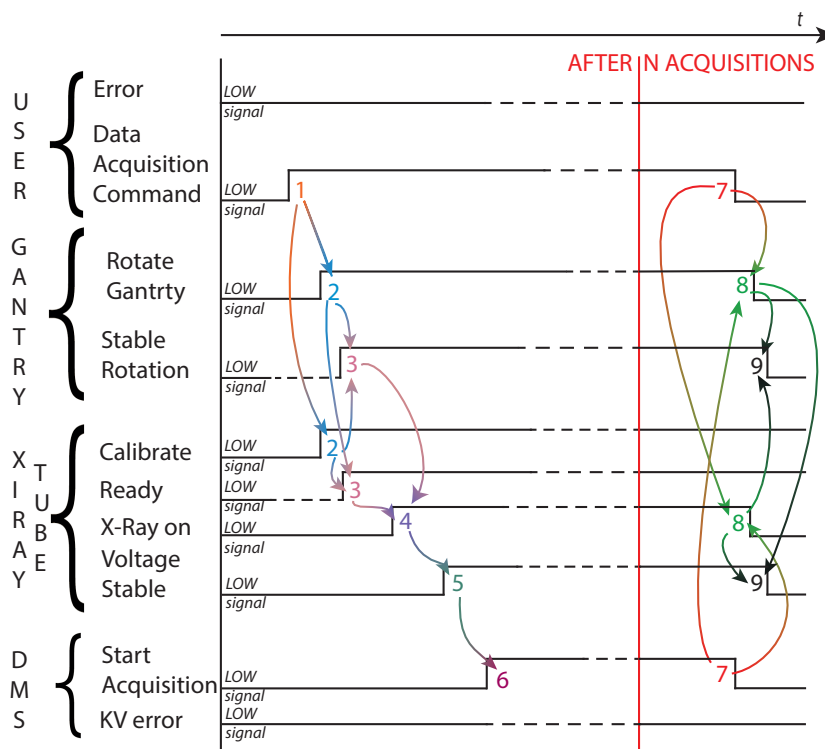


Figure 9.3: KIDS-CT: Synchronization steps for making CT scans

In order to make a CT scan, the decision logic checks the status bits associated with the voltage of the DMS, the X-ray tube, and the gantry speed, all of which must be stable when the user sends the acquisition command. Fig. 9.3 shows the steps (depicted by the numbers) to perform a scan, starting from the acquisition command sent by the user until

the last projection is collected by the CDAS, without errors.

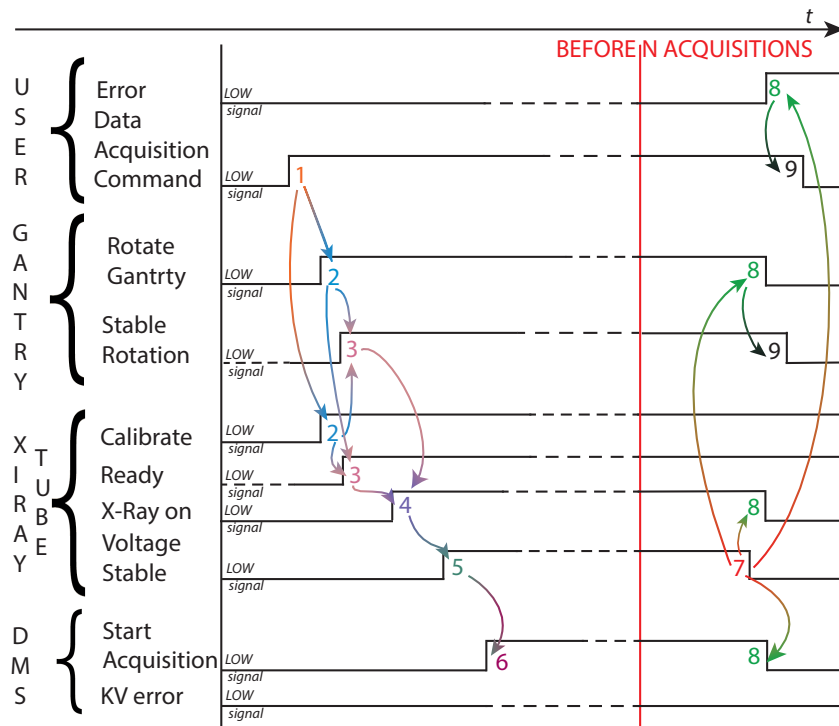


Figure 9.4: KIDS-CT: Synchronization errors during a CT scan

If the gantry rotates too fast or too slow or the voltage of the X-ray tube or DMS is not stable, the synchronization condition is not met. Consequently, the Synchronization Unit will stop the acquisition in a safe way, following the steps shown in Fig. 9.4, where the control signals that stop the system are generated within a few clock cycles after the detection of an error (number 7). In addition, the system is stop independently of the user interface where the error is only notified.

### 9.3 Data-Flow Module

This module is responsible for collecting data from sensors (i.e. DMS) and forwarding them to the reconstruction system. During acquisition, the user can also decide to perform on-the-fly data processing using the proposed Data-Processing Module within the CDAS and forward the processed data to the reconstruction system that will only perform the back-projection step.

The Data-Flow Module implements the proposed lightweight re-configurable dataflow architecture, which has been configured at design time with the parameters explained in Section 7.4 and shown in Tab. 9.1 to be integrated into the KIDS-CT scanner. Since data needs to be collected from the DMS by the CDAS and then from the reconstruction system to perform the back-projection, this module has been instantiated in the CDAS and in the reconstruction system, where another MPSoC-FPGA has been plugged-in.

For evaluation purposes, also Scale Case (SC) configuration was set up with five receivers. This collects data, processes them on the fly, and stores them at 25 Gbit/sec in global memory via PCI-E. This configuration was used to analyze the scaling of the system as additional sensors such as the DMS are added to the KIDS-CT system. This use case will be discussed in the resource evaluation of the Data-Flow Module in Section 11.2.

Table 9.1: Design-time parameters of the Data-Flow Modules within the KIDS-CT scanner: CDAS and the reconstruction system (RS)

Design-Time Parameters						
	N(rx)	P(pr)	M(tx)	FIFO depth	Data-width	GTX
CDAS	2	4	1	1 row	about 30 bits <sup>1</sup> rx / 64 tx	Optical Fibre <sup>1</sup> (rx)/ SFP+ (tx)
RS	1	0	1 (4 lanes)	1 projection	64 rx / 128 tx	SFP+ (rx) / PCI-E (tx)

Tab. 9.1 reports the parameters to see the physical interfaces and the data width for the AXI-Stream interfaces. We can notice that while the Data-Flow Module implements two receivers for acquiring data from the DMS, it uses only one transmitter to stream the processed data to the reconstruction system. We can see that the data depth of the reconstruction system has been set to 1 projection, while the CDAS has been set to 1 row. Even though one row is sufficient in the reconstruction system, a projection is required to implement the ramp filtering in this second system. However, since the system has to wait for an interval of 1 *ms* (i.e., integration period) between two projections, this means that while the system reaches a data rate of 6.250 Gbit/sec during transmission, the average data rate is equal to 1.5 Gbit/sec due to this waiting time. Therefore, with the FIFO depth equal to 1 projection, the number of FIFO in the reconstruction system can also be reduced to 1.

Table 9.2: Runtime parameters of the Data-Flow Modules within the KIDS-CT scanner

Run-time Parameters						
Stage	Ref. clock [MHz]		Data clock [MHz]		Protocol	
	CDAS	RS	CDAS	RS	CDAS	RS
Receiver	156.250		78.125	156.250	8b/10b	
Data-link Prorocol	-		78.125	156.250	custom	
Scheduler Stage	-		78.125/156.125	156.250/250	-	
Stream Generator	-		156.125	250	custom	
Transmitter	156.250	100	156.125	250	8b/10b	PCI-E

When the scanner is powered up, the run-time parameters shown in Tab. 9.2 must be set. These other parameters are required to establish the connections between the involved nodes. Both run-time and design-time parameters are essential to provide support for real-time and on-the-fly data acquisition because the number of FIFOs, the receive and transmit data clocks, and the data-with, allow the system to be dimensioned so that the

<sup>1</sup>Due to the confidential nature of this matter, the exact information cannot be disclosed.

transmit and processing data rates are higher than the acquisition data rate. In addition, this solution allows the number of on-chip FIFOs to be dimensioned in a way to avoid external memory to store data, in contrast to the current state of the art. The data module must be able to send data at a higher rate than it receives.

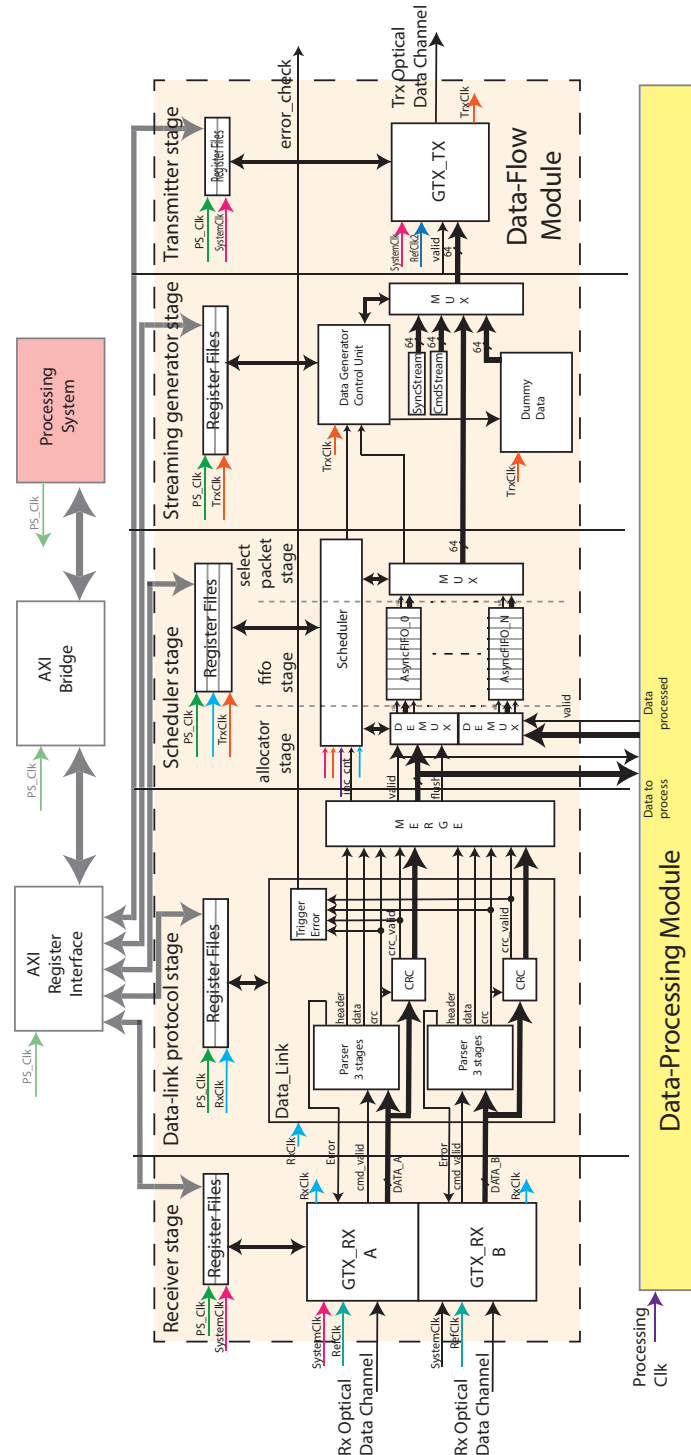


Figure 9.5: KIDS-CT: Data-Flow Module instantiated in the CDAS

Based on the parameters defined in Tab. 9.1 and in Tab 9.2, the CDAS is set to collect and forward data to the reconstruction system at a rate of 6,250 Gbit/sec. Finally,

while the internal stages of the architecture were designed in SystemVerilog as generic parametrizable modules, for the transceivers AMD-Xilinx IP blocks were instantiated and set based on the described parameters. Finally, the registers of the different stages were mapped using an AXI4-Lite interface. The Data-Flow Module for the CDAS integrated into the KIDS-CT scanner resulted in the architecture shown in Fig. 9.5.

Fig. 9.5 shows how the two transceivers in the Receive stage, where data are collected and send in the Data-Flow Module stages on two channels; the Data-link protocol stage decodes the packets of these two channels using the custom protocol provided by the detector vendor. Each projection is stored in a different FIFO selected by a scheduler implemented in the Scheduler stage. Here, the data are merged and scheduled to an available FIFO, while the previous projections are read in order from the FIFOs and passed to the Streaming generator stage or externally to the Data-Processing Module. Asynchronous FIFOs are involved in this stage to realize the CDC. Finally, the Streaming stage gets data aligned to 64 bits, packages them according to the proposed Application Stream Protocol, and passes them to the Transmitter stage, which sends the packaged data to the reconstruction system via a GTX transceiver. With the exception of the Scheduler Stage that has not a deterministic delay because different FIFOs are scheduled with a round-robin algorithm, all other stages has been designed to have a deterministic delay.

## 9.4 Data-Processing Module

In the KIDS-CT scanner, the Data-Processing Module within the CDAS implements the pixel processing part based on the mapping done for the proposed datapath optimization presented in Section 8.4. The algorithm steps are described using C language and then integrated into the Data-Processing Module as IP cores, from the RTL which is generated with the C-Synthesis in Vitis™ HLS. They communicate with each other via AXI4-Stream interfaces, allowing them to be integrated into any design using the AXI4-Stream interface protocol, independent of the targetMPSoC-FPGA or FPGA. Furthermore, the IP cores can be configured at synthesis time to handle custom and standard data formats.

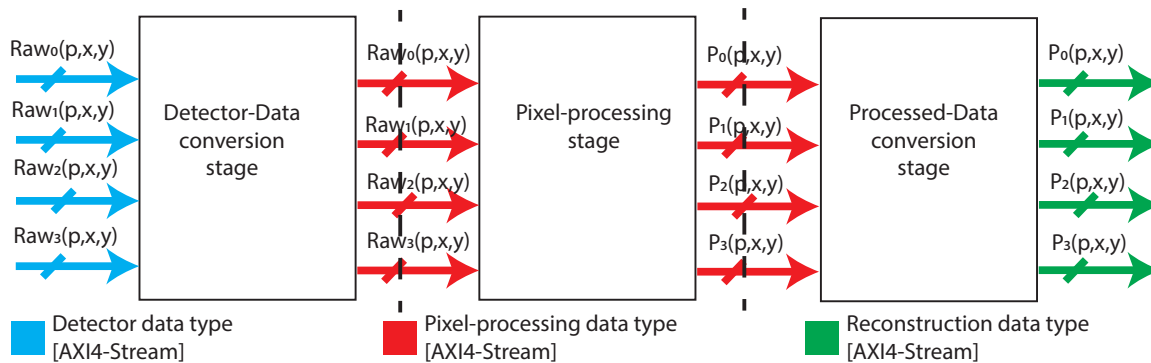


Figure 9.6: KIDS-CT: Data-Processing Module



In order to process data during the scanning, the Data-Processing Module is connected to the Data-Flow Module, in the Scheduler Stage, as shown in Fig. 9.5. Here, the collected data are forwarded to the Data-Flow Module that processes them in the architecture shown in Fig. 9.6.

The CDAS for the KIDS-CT scanner has been configured to acquire four pixels per clock cycle. Consequently, the Data-Processing Module has been configured to process four pixels per clock cycle, as shown in Fig. 9.6. In order to perform these steps in a dataflow architecture and to process data on the fly, several optimizations have also been proposed that are specific to the FDK CT reconstruction algorithm. Here the *Detector-Data conversion stage* is responsible for converting the detector data into the desired data format. The *Pixel-processing stage* performs them in a dataflow architecture, and the *Processed-Data conversion stage* converts the data to be sent to the reconstruction system. If data need to be processed with different data formats in the *Pixel-processing stage*, conversion sub-stages can be added between the steps where a different data format can also be used. By tuning different data formats in the first and third stages, the design space for the pixel processing part was explored, with the aim to find the best solution in terms of performance and image quality of the CT reconstructed images.

### 9.5 Pixel Processing Optimization

The pixel processing part has been implemented in the Data Processing stage of the Data-Processing Module, using separate IP cores per function, as shown in Fig. 9.7.

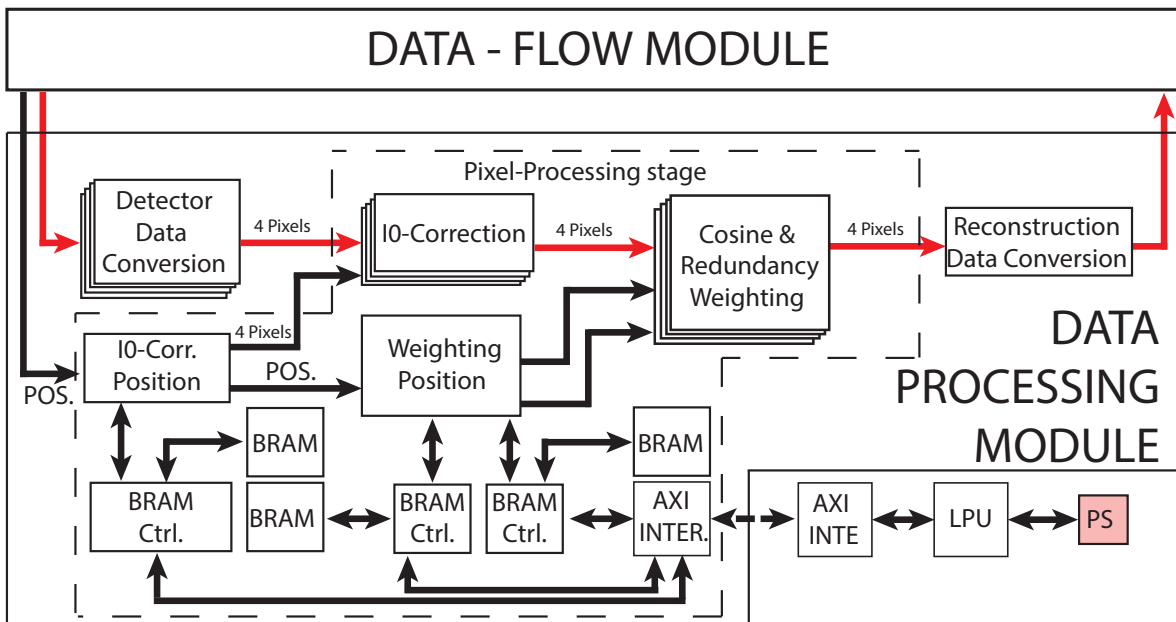


Figure 9.7: Implementation of the Pixel-processing stage for the KIDS-CT scanner

As shown in the figure above, this Module is externally connected to the Data-Flow Module and internally includes the following steps: I0-correction, and Cosine & Redundancy Weighting. Since these steps require additional offline data that depend on the pixel position to be processed, BRAMs and FFs are used to store this information, which can also be accessed by the PS via an AXI4 interface. While the data parameters are set at run time, the type of processing to be applied and the number of pixels to be processed per clock cycle are set at design time.

### 9.5.1 I0-correction step

In Section 2.4.3 the theory behind the CT reconstruction algorithm was introduced. As explained, the raw data are acquired in the attenuation domain, but the I0-correction is usually applied to the intensity domain data because commercial CT scanners provide them already converted. According to the steps that usually are performed in the literature, the reconstruction algorithm should first convert the data from the attenuation to the intensity domain and then apply the I0-correction to each pixel, performing non-linear arithmetic operations, as also shown in the equation 9.1. In order to perform these complex operations on the PL parts, DSP slices are required, resulting in a low-performance data processing [158]. In order to compare the proposed optimization, this standard solution has been also implemented and their results are discussed in Section 11.3.

$$P(u, v, \alpha) = \log_e \left( \frac{c \cdot \exp(-(f \cdot Raw_0(u, v, \alpha)))}{c \cdot \exp(-(f \cdot Raw(u, v, \alpha)))} \right) \quad (9.1)$$

In the equation 9.1, the  $Raw_0$  input represents the pixel of an object-free projection that is collected during the scanner calibration; the  $Raw$  input represents the current collected pixel;  $c$  and  $f$  are constants that depend on the sensor parameters. Since the KIDS-CT scanner provides raw pixel data in the form shown in the equation 9.1, this can be used instead of the equation 2.3 usually used in commercial CT scanners. In addition, this equation contains logarithmic and exponential functions, which can be simplified because they are also the corresponding inverse functions. Therefore the equation 9.1 can be simplified into the equation 9.2.

$$P(u, v, \alpha) = \log_{10}(2) \cdot f \cdot (Raw_0(u, v, \alpha) - Raw(u, v, \alpha)) \quad (9.2)$$

The equation 9.2 contains only one multiplication and one subtraction arithmetic operation, which can be implemented in PL within a few clock cycles, using Look-Up Tables (LUTs).

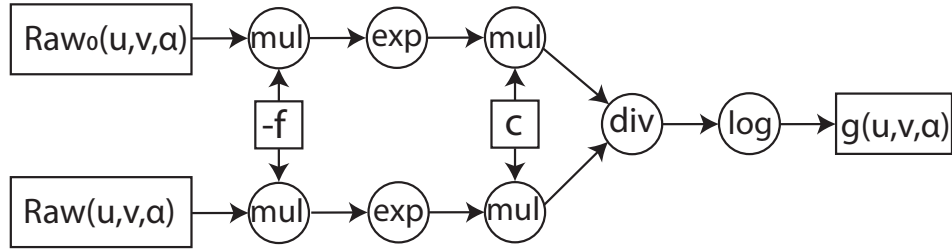


Figure 9.8: Data flow graph of the standard I0-correction

In order to process a pixel on the fly and implement these equations in a dataflow architecture, they were modeled with the data flow graph shown in Fig. 9.8 and in Fig. 9.9.

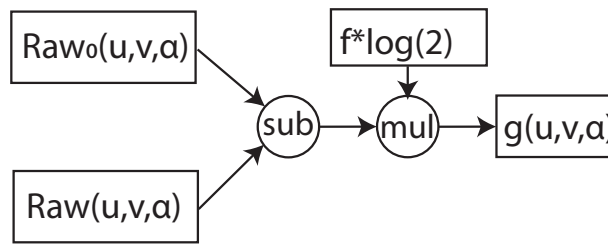


Figure 9.9: Data flow graph of the optimized I0-correction step

In the data flow graphs, the input/output data and constants are modeled with the square boxes, the operations are modeled with the circular boxes, and the data flow is modeled with arrows (i.e., vertex) [159]. These show the flow of data and the dependencies between the operations. The different operations in the boxes have different latencies and resource utilization in the PL part. The values of these metrics depend on the operation implemented and the chosen data format. Therefore, by performing the I0-correction directly on the attenuation domain data, it is possible to have an optimization already in this step. Furthermore, by tuning the data formats, it will be possible to find the optimum in the design space, considering the latency, the resource utilization, and the image quality in the evaluation of the different solutions in the design space.

### 9.5.2 Cosine weighting and redundancy weighting steps

After the I0-correction step, each pixel must be weighted with the cosine of the angle between the iso-ray (the ray passing through the iso-center) and the ray intersecting the detector pixel. As explained in Section 2.4.3 in the case of the cylindrical detector (i.e., detector elements are arranged along an arc), the cosine weighting reported in equation 2.4 is calculated by multiplying the I0-corrected pixel with the multiplication factors “ $\cos(u \cdot \Delta\beta) \cdot \frac{D}{\sqrt{D^2 + (v \cdot \Delta v)^2}}$ ”

In order to optimize this operation, various approaches could be followed:

1. **Perform all operators at run time without pre-computation.** This approach is computationally intensive because it requires performing two multiplications and two powers, one addition, one division, and one cosine operation, but minimizes memory usage. It should be implemented according to the data flow graph shown in Fig. 9.10.

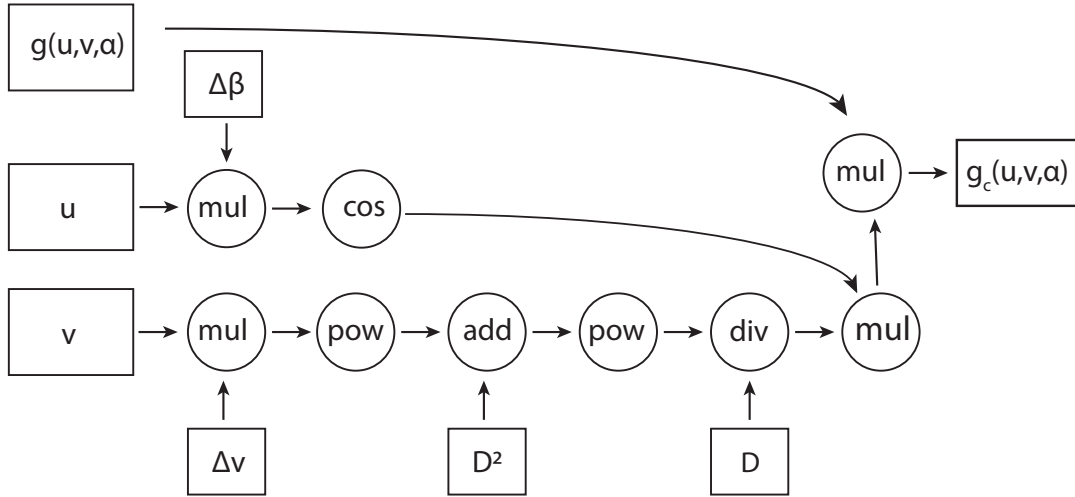


Figure 9.10: Data flow graph of the first approach for the cosine & redundancy weighting

2. **Perform only the multiplication between the cosine weight and the input pixel at run time, using a memory space with the size of  $N_u$  by  $N_v$  to store the weights.** In the KIDS-CT scanner, this requires a memory space of  $672 \cdot 64 = 43008$  pixels with the size of the selected data format. This approach optimizes latency because only a single multiplication needs to be performed, but requires the highest on-chip memory utilization. It should be implemented according to the data flow graph shown in Fig. 9.11.

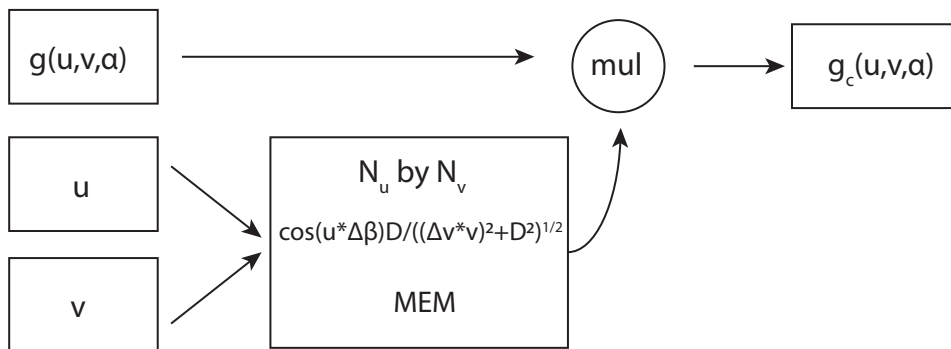


Figure 9.11: Data flow graph of the second approach for the cosine & redundancy weighting

3. **Perform of the multiplication with the input pixel and between the row factor  $\cos(u \cdot \Delta\beta)$  and the column factor  $\frac{D}{\sqrt{D^2 + (v \cdot \Delta v)^2}}$  at run time, resulting**

in a memory space having the size of  $N_u + N_v$  pixels. This solution requires  $672 + 64 = 736$  values with the size of the chosen data format. It optimizes the latency compared to the first approach and the memory usage compared to the second approach. Its data flow graph is shown in Fig. 9.12.

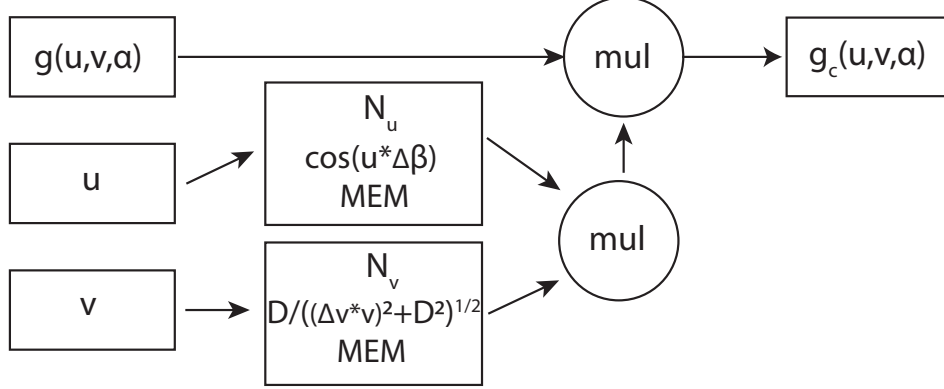


Figure 9.12: Data flow graph of the third approach for the cosine & redundancy weighting

The factors used for the third approach represent the distance from a pixel in the PCS and the origin in the DCS. As shown in Fig. 9.13, for each pixel  $(u_i, v_i)$ , there are three other pixels  $(u_j, v_j)$ ,  $(u_l, v_l)$ , and  $(u_m, v_m)$  with the same distance (i.e., cosine weight).

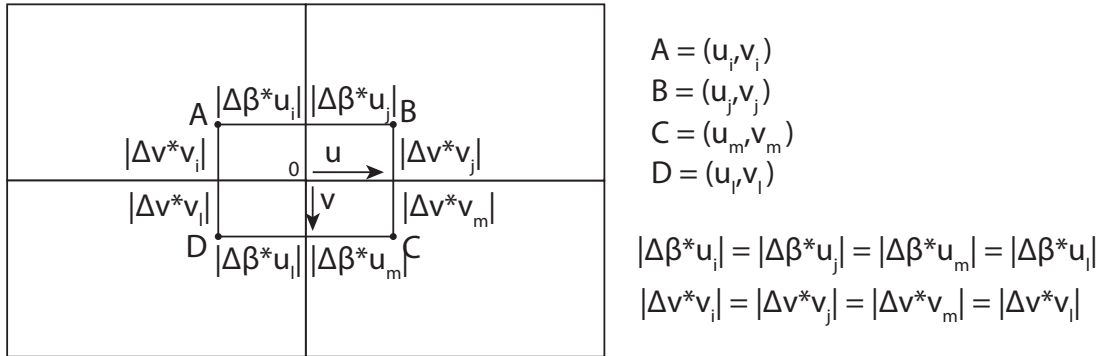


Figure 9.13: Example of pixels with same weights within a projection

Therefore, it is possible to determine all the weights of a projection, knowing only the values associated with one of the quadrants. This means that the second proposed approach can be optimized with a memory space of  $\frac{N_u}{2}$  by  $\frac{N_v}{2}$  values and the third approach with a memory space of  $\frac{N_u}{2} + \frac{N_v}{2}$  (i.e.  $336 + 32 = 368$  values of the chosen data format). The proposed optimization for the third approach allows the cosine weighting to be implemented using only two multiplier hardware units. It requires the storage of only 368 values, which are much less values than the storage of the entire cosine weighting matrix required for the second approach.

After performing the cosine weighting, the redundancy weighting should also be applied to the input data by multiplying each pixel by  $w_r(u, v, \alpha)$ . In the KIDS-CT scanner,

$w_r(u, v, \alpha) = \frac{1}{2}$ , which is a constant value; therefore, it can also be integrated as part of either the row or column factors of the redundancy weighting, which are stored off-line.

Although it is possible to implement both the second and the third proposed approaches and apply the redundancy weighting to them, the third solution has been chosen for the implementation of the KIDS-CT scanner in order to balance the use of resources between the different steps implemented in the PL. In addition, the proposed optimization to reduce the amount of data to be stored offline was applied, and the redundancy weighting was integrated into the row factor, as shown in Fig. 9.14.

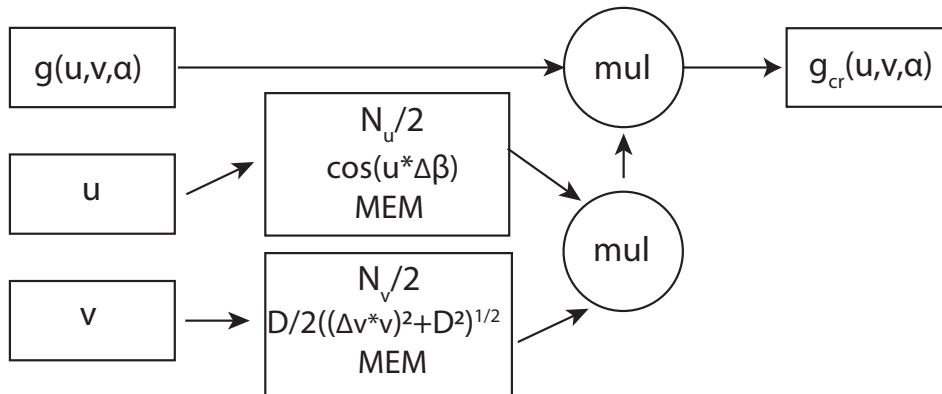


Figure 9.14: Data flow graph of the third approach with the memory-optimized for the cosine & redundancy weighting

The pre-computed weighting values have been mapped to a dual port BRAM. Therefore, the pixel position is used to read the corresponding factors, which are streamed to the weighting IP core generated using Vitis™ HLS. The pixel position is calculated while the I0 correction has been performed to be integrated into the dataflow pipeline architecture and not add any delay caused by this memory reading.

## 9.6 Design Space Exploration

In order to optimize performance and provide real-time support, this thesis also proposes an exploration of the design space for the pixel processing part. By tuning the data formats of each step, different arithmetic hardware units are used, resulting in different latency and resource utilization of the Data-Processing Module and image quality of the reconstructed 3D image (i.e., slice). While resource utilization is not critical in this application, the latency and the image quality are essential variables for interventional procedures.

Due to the size of the design space, given by the input options for the data format and clock frequency parameters, and due to the time required to implement, generate the bitstream and deploy on the KIDS-CT scanner and then to analyze the image quality of each resulting solution, it is not possible to consider all of them. Therefore, in order to

make the DSE affordable while limiting the exploration of the design space, two steps have been defined: the *Selection of input parameters* and the *Selection of metrics*. These two steps are guided by the specific data of the application, from the format provided by the sensors to the format of the final data to be displayed.

### 9.6.1 Selection of input parameters

The selection of input parameters aims to simplify the problem by reducing the design space to be explored. Since the DSE focuses primarily on finding the optimal data format for CT pixel preprocessing, the clock frequency parameter is set to a fixed value. This is possible because it does not affect the image quality. Instead, it only affects the latency of each data format by a linear factor which is proportional to the frequency. Thus, for the DSE it has been fixed at 100MHz, which is the frequency set for the *system clock* in the target MPSoC-FPGA for the CDAS architecture.

Although custom representations can be used, only standard data formats have been chosen because they can be implemented on other accelerators, such as GPUs and Tensor Processing Units (TPUs). Therefore, the design space is limited to floating-point and fixed-point formats, which are introduced in Section 2.5.1. The exploration of the design space starts from the 32-bit floating-point solution, which is the de facto standard solution for diagnostic CT reconstruction algorithms. Since the DSE aims to find a better solution than single-precision floating-point and to improve computing performance and the hardware cost while maintaining the image quality, only suitable data formats with a data width between 16 and 32 bits were selected in this step, since the raw sensor data collected by the CDAS can be represented by the short format, which has a data width of 16 bits. Therefore, the 16-bit floating-point format was selected as the second solution. In addition, the double format (64-bit data width) was also considered as the reference format in the image quality analysis. Therefore, the research was limited to three different standard floating-point formats: *half*, *single*, and *double* precision.

Then, moving to fixed-point representation the selection of the data format becomes more complex than floating point. While for the floating-point standard, there is only one encoding option per data width, for the fixed-point standard, a different encoding can be set for each selected data width. In fact, there are 408 configuration options in the range of 16 to 32 bits for the fixed-point representation, but the goal of the selection is to find only two configurations to analyze for this representation. For this reason, only the upper bound (i.e. 32 bits) and the lower bound (i.e. 16 bits) configurations were considered for DSE, which reduces to 16 plus 32 possible configurations as a fixed-point representation. Since 48 configurations cannot even be performed in terms of time cost.

In order to achieve the desired two configurations for the fixed-point representation, where the former refers to the upper bound (32 bits) and the latter to the lower bound (16 bits), the selection was based on the input data format (i.e., raw sensor data); these data are represented in the Data-Flow Module as 16-bit unsigned values. Therefore, to represent the entire value in a 32-bit fixed-point format, both the I part and the F part were set to 16 bits. With this selection, the raw sensor data can be accurately represented in the 32-bit fixed-point format without the need for any approximation during the conversion.

Since a 16-bit fixed-point representation cannot contain a 16-bit unsigned value, the raw sensor data must be approximated when this format is selected. For this reason, both the raw sensor data and the processed data in 64-bit floating point were analyzed to find the best combination of I part and F. Notably, most values after the pixel processing part could be represented using only 4 bits in the I part and the rest of the mantissa was for the F part. Consequently, the optimal configuration for a 16-bit fixed-point was determined to be 4 bits for I and 12 bits for F. In addition, to verify that this configuration is the best one, for this application data the MSE of the performed data has been also estimated with the neighbor configurations has been also performed. The MSE is the mean squared difference between a reference value and an approximated value [160]. This is often used to measure the image quality between two images [161].

Following the explained methodology, the number of data formats aimed for the selection was achieved. Thus, the selected formats are 16-bit and 32-bit floating points and 16-bit and 32-bit fixed points where I and F are both equal to 16 bits for the 32-bit fixed point and equal to 4 and 12 bits for the 16-bit fixed point. In addition, the 64-bit floating point was selected as the reference data format for the evaluation. To prove that all the selected configurations are suitable to be analyzed in the DSE, their MSE estimations will be discussed in Chapter 12.

### 9.6.2 Selection of metrics

In order to evaluate the different solutions given by the selected parameters, appropriate metrics must also be selected, taking into account computing performance and hardware cost, as well as the image quality of the reconstructed images. Due to the time required to perform a quantitative analysis of the image quality, which is essential for finding the best data format in interventional CT procedures, a selection of metrics was made.

For evaluating hardware cost and computing performance, the “resource utilization” of the PL and the “latency” of the different solutions were respectively selected as metrics. The resource utilization in the PL is quantified in terms of DSP, FF, BRAM, and LUT resources [162]. These metrics were computed at synthesis time by the Vitis™ HLS tool for



each processing step and selected parameters, as pre-analysis; then the metrics were also computed at implementation time using Vivado Design Suite 2021.2. This approach was preferred for the DSE as it allows for the estimation of resource utilization of the processing steps before their integration into the CDAS. Then, the implementation results, obtained after integration into the CDAS using Vivado Design Suite 2021.2, will be reported in Chapter 12. These results are more accurate as they reflect the implementation considering the mapping and floorplan of the entire design.

To analyze the image quality of the different solutions, the following metrics were selected: first the MSE of the processed projections and then the MSE, the “noise”, the “low-contrast” and the “uniformity” of the reconstructed volume. The MSE was applied to the 2D projections to validate the resulting solution before performing the reconstruction and the other metrics but this is not enough because it cannot express a qualitative analysis of the images; it is only useful to understand if the chosen design solution is acceptable for the reconstruction. However, the uniformity and noise metrics are important in CT imaging to identify any image degradation caused by the arithmetic approximation of the different data formats. The low-contrast metric is important for tumor detection [163], useful in tumor ablation.

The selected parameters and metrics make the DSE affordable. In the next Part of this thesis, the calculation of the selected metrics will be explained in detail and the results of the different design options will be presented. These results are useful for the designer to select the best data format for CT imaging either for diagnostic or interventional procedures. Furthermore, this selection approach can also be applied to other CPS devices where data processing is required.

## 9.7 Component Isolation

The proposed isolation mechanism has been realized by the LPU presented in Section 7.3. It was implemented in SystemVerilog and packaged as a soft IP core, using the Vivado Design Suite. This tool permits an RTL module to be packaged into an IP core that is compatible with any AMD-Xilinx platform from the 7-series family [164]. In fact, the LPU was then integrated into the CDAS architecture, following the IP design flow proposed by AMD-Xilinx [165].

In the KIDS-CT scanner, as shown in Fig. 8.2, a single instance of the LPU is connected between the AXI master interface of the PS and the AXI interconnect. This is the hardware module that connects all the hardware units associated with the various external components that are instantiated in the PL. In the actual implementation, the LPU was

set at design time to support 2 PDs (i.e., the APUs on the PS) and 5 different MRs. These MRs are associated with the components within the Control-Synchronization Module, the Data-Flow Module, and the Data-Processing Module. Although the PD and the MR are set at design time, the APs that define which PD can access the MRs are defined in the FSBL and set at boot time before the OS is started. For this, the PS is connected to the LPU configuration registers via an additional dedicated AXI4-Lite interface, which can only be accessed by the FSBL as it is not mapped to the OS space memory. For the KIDS-CT scanner, it was decided to use this conservative solution for setting policies because they do not need to be updated at run time.

## **Part IV**

# **Validation & Evaluation**



## 10 Validation

This Chapter discusses the validation of the work proposed in Chapters 5, 6, and 7, and of their realization presented in Chapters 8 and 9 for the target KIDS-CT scanner. First, it presents the validation methodology, including software tools and hardware units used for the target MPSoC-FPGA. Then, it describes the validation of the various modules within the CDAS architecture, which is applied during the design phase, the post-implementation phase, and the post-integration phase of the CDAS architecture into the KIDS-CT scanner. The contents of this Chapter have also been used to validate the research work presented by the articles in Ref. [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7].

### 10.1 Validation methodology

The implementation of the CDAS architecture has been deployed and validated on the AMD-Xilinx ZC706 Evaluation Board mounting the XC7Z045 MPSoC-FPGA. As described in Chapter 9, the modules mapped to the PL part were described using HDL and IP cores, and synthesized and implemented using Vivado Design Suite 2021.2 following the Vivado System-Level Design Flow [166]. The software stack, including the FSBL and Petalinux at the OS layer, and the software modules at the application layer, were developed in C language, implemented using the Vitis Unified Software Platform [167], and deployed on the PS portion of the target MPSoC-FPGA.

All of these tools used for implementing hardware and software modules are also essential for validating and evaluating the implemented design, allowing designers to analyze, verify, and modify the design at any step of the flow [168]. Although they are specific to the AMD-Xilinx MPSoC-FPGA platforms, tools with similar validation functionality are typically provided by other vendors.

Independent of the tools and the target MPSoC-FPGA, a bottom-up validation approach has been adopted for the CDAS architecture, including also the proposed System Architecture and the Communication Infrastructure. It consists of the following three steps: the design phase, the post-implementation phase, and the post-integration phase. In the first phase, each hardware unit within the proposed modules was simulated first as a standalone and then after the integration into the corresponding module; this process was iterated in a bottom-up approach to cover the validation of the whole module. Then, in the second phase, the CDAS has also been validated by running the CDAS standalone in debug mode. Finally, in the last phase, after the integration of the CDAS into the KIDS-CT, Integrated Logic Analyzers (ILAs) were used to monitor the behavior of the

architecture while it was running the different application use cases, such as the scanning of a phantom. These phases are described in detail in the following Sections.

## 10.2 CDAS Design Phase

During the design phase, a behavioral clock-cycle simulation was performed for each hardware unit. For this purpose, a test bench was written for each unit. In addition, in order to validate the whole module after the integration into the CDAS architecture, different approaches were defined for the Control-Synchronization Module, the Data-Flow Module, the Data-Processing Module, and the Lightweight Protection Unit.

- **Control-Synchronization Module:** To validate this module, sequences of control, status, and synchronization stimuli (i.e., signals) must be generated and checked. Therefore, SystemVerilog testbenches have been implemented that consider the different conditions and compare the actual and the expected output.
- **Data-Flow Module:** It processes a large amount of data in a streaming fashion, pruning and merging them in a precise order. Therefore, this module must be validated by verifying that data are processed in the expected order without loss. For this reason, the SystemVerilog testbench generates and checks several fixed data patterns that are sent to and read from this module. A data pattern consists of commands (e.g., synch, preamble, and CRC commands) and messages that follow the proposed Application Stream Protocol described in Section 6.3.3. In addition, each message contains a sequence of ordered values.
- **Data-Processing Module:** The units within this module have been described in C language and implemented using HLS. Since they implement pixel processing steps with different data precisions, each algorithm step has been validated within the Vitis HLS tool, where testbenches have been developed in C. A testbench reads a raw projection image and sends pixels as AXI stream packets to the hardware unit under test. Then the resulting image data that has been generated with the simulation are compared with the same image data implemented on the CPU. For comparing the two processed image data the MSE is used as metric.
- **Lightweight Protection Unit:** Since it involves the handling of AXI4 transactions, the testbench should generate/check transactions according to the protocols and verify that granted transactions are kept unmodified and denied transactions are handled without causing communication deadlocks.

To generate/check stimuli according to the AXI4 interface protocol, the AXI Verification IP (VIP) core provided by AMD-Xilinx was instantiated in the design. As a result, a test design with the three possible configurations was implemented, as

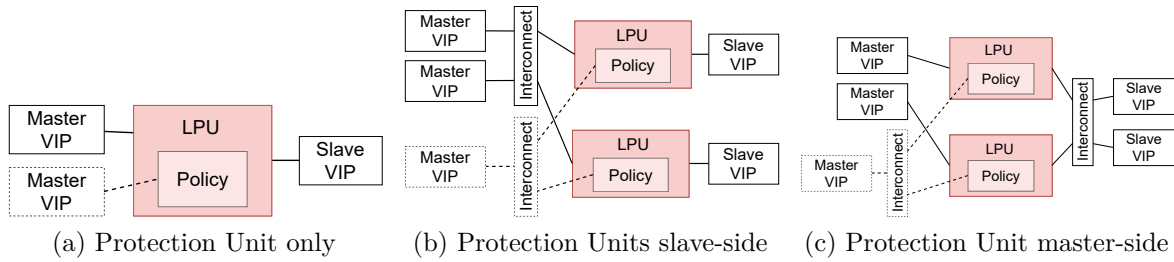


Figure 10.1: Design configurations for the LPU validation. The dashed boxes are related to the LPU configuration interface, while the other boxes are related to the transactions to grant/deny

shown in Fig. 10.1. All depicted implementation options consist of a control master (dashed master VIP box) that sets the policies at runtime, two PDs (master VIP boxes), and two MRs (slave VIP boxes) set at design time, as shown in Fig. 10.2.

(a) Protection Domain configuration

(b) Memory Region configuration

Figure 10.2: Validation settings of the LPUs

### 10.3 CDAS Post-Implementation Phase

In order to validate the CDAS architecture in the post-implementation phase, different techniques have been proposed for the different modules. For the Control-Synchronization Module, a debug software application has been developed that allows the writing/reading of control/status registers, thus enabling the debug mode. When the system is in debug mode, the Data-Flow Module can also be tested because the user can select different multiplexers that forward either the input data or a fixed pattern of data generated by a traffic generator in the dataflow architecture. To validate and debug the different stages of the Data-Flow Module, traffic generators have been instantiated in two stages of the architecture, as shown in Fig. 10.3.

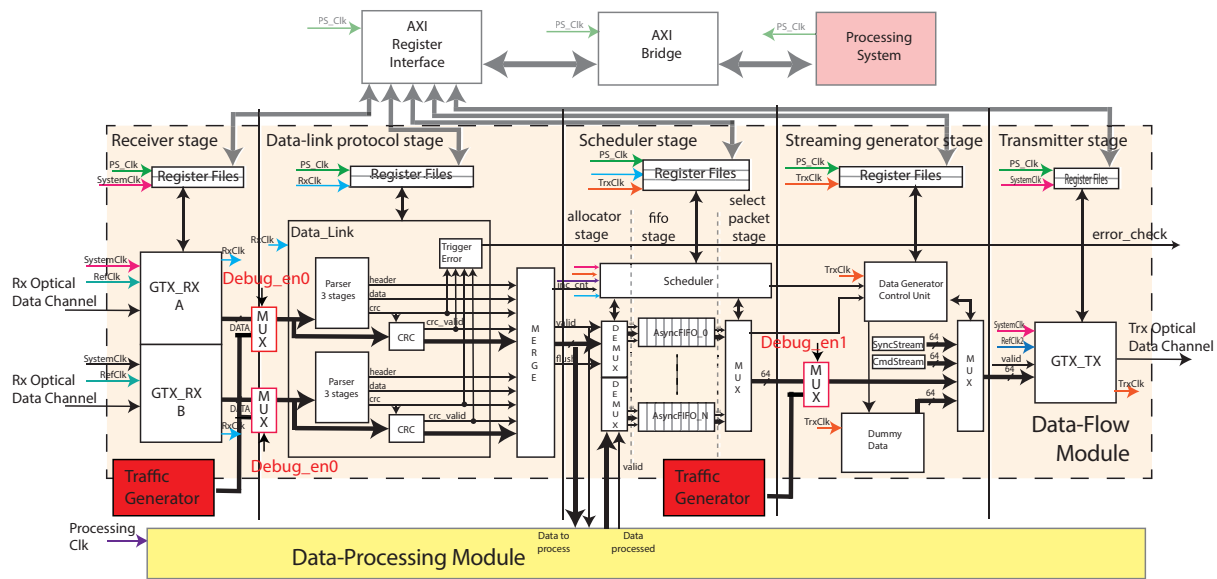


Figure 10.3: Validation of the Data-Flow Module. Traffic generators in red boxes

The traffic generator in the receiver stage of the pipeline sends commands and messages containing an incremental sequence of values according to the Application Stream Protocol. Another generator in the scheduler stage sends messages that are then packaged and dispatched by the Streaming generator stage. These two traffic generators, which are controlled by the PS at run time, allow the designer to debug and validate the entire Data-flow Module before integrating it into the target application. Moreover, to validate the hardware units within the various proposed modules, ILAs were instantiated and connected to their interfaces. These are logic analyzer cores provided by AMD-Xilinx that can be used to monitor the internal signals of a design while it is running [169]. Because ILAs increase the resource utilization in the MPSoC-FPGA, a “`#ifdef`” for each ILA is used in the HDL modules to give the designer the option to remove them from the release design. This allows the designer to disable some or all of them before compiling the SystemVerilog code, and they will not be included in the design during synthesis and implementation. With the support of the described techniques, the CDAS architecture



was validated at run-time already before integrating it within the KIDS-CT scanner.

## 10.4 KIDS-CT Post-Integration Phase

In order to validate the CDAS architecture at the post-integration phase within the KIDS-CT scanner, additional ILAs were instantiated in the design. This approach mirrors the post-implementation phase, where ILAs were selected and included in the design for validation and potential debugging in the final product. Since ILAs are typically accessed through a Joint Test Action Group (JTAG) connection implemented in the target board via a Universal Serial Bus (USB) interface, a USB device server [170] was placed on the rotating side of the KIDS-CT scanner. This allows the user interface system node to access the ILA in debugging mode using the same connection that is part of the communication associated with the *non-real-time interface class*. This setup was crucial to validate all the connections to the client-slave nodes and the behavior of the CDAS architecture in the post-integration phase. For this validation phase, an ILA was instantiated for each transceiver associated with the *real-time control interface class* and the *real-time data interface class*. In this way, it was possible to monitor and verify the correct communication between each node and the CDAS. In this way, the CDAS architecture and the KIDS-CT scanner functionalities were tested and validated during the acquisition of the phantoms, which were also used for the image quality analysis.

# 11 Performance Evaluation

The following Chapter presents the performance evaluation of the CDAS architecture within the KIDS-CT scanner. In order to analyze the scalability of the CDAS architecture and the impact of adding a new component into the CPS, different configurations of the modules of the CDAS architecture are considered. The evaluation focuses on the hardware cost and computing performance of the implemented CDAS architecture for the KIDS-CT scanner. For this evaluation, the resource utilization and latency have been chosen, which are the same metrics selected for the DSE. The former is crucial to provide plug-and-play capability, while the latter to support real-time applications. For example, in the mapping and in the placement & routing of a design, a low percentage of resource utilization allows for easier extension compared to a design where most resources are utilized, which may be constrained by resource limitations. The content of this Chapter is further presented in the articles in Ref. [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7].

## 11.1 CDAS Architecture For The KIDS-CT Scanner

The implemented CDAS architecture has been evaluated on the XC7Z045 MPSoC-FPGA, with the support of Vivado Design Suite 2021.2. This tool reports the amount of resource utilization and permits to estimate the delay for the timing analysis of an implemented design.

### Resource utilization

This Section reports the resource utilization of the CDAS architecture considering the proposed optimizations and the Debug Units for the validation. Before delving into the resource utilization results, it is important to clarify that in each table the resources of the “TOP” (i.e. the main RTL module in a hardware design) are not a simple sum of the sub-module resources, since additional logic may be used in the “TOP” to connect or control internal sub-modules. In addition, during the synthesis optimization, Vivado Design Suite allows the synthesis tool to “flatten the hierarchy, perform the synthesis and rebuilt the hierarchy based on the original RTL” [171]. Therefore, resources of a certain RTL block can be moved to another RTL block within the hierarchy such as between TOP and submodules. Then, in 7-Series FPGA, 1 BRAM is a 36 Kbit block of memory; but this block can be divided into two independent 18 Kbit blocks. Therefore, if 1 block 18 Kbit is used in the logic, Design Suite reports it as 0.5 BRAMs. For instance, the

Data-Flow Module and the debug units in Tab. 11.1 consist of 18.5 and 5.5 BRAMs.

The implemented CDAS architecture for the KIDS-CT scanner uses less than 8% of the available resources in the target MPSoC-FPGA, as reported in Tab. 11.1, thanks to all the proposed optimizations. This implementation also includes the debug units used for post-integration validation. Although debugging units can usually cause a degradation in computing performance (e.g., timing issues) due to the additional resources, the CDAS architecture uses so few resources that the performance is not been affected by debug units. However, the proposed optimizations for the pixel processing part, implemented into the Data-Processing Module, are crucial for the achieved results.

Table 11.1: Resource utilization of the CDAS architecture for the KIDS-CT scanner; “TOP” is refers to the main block that contains all the modules;  $n$  refers to the amount of resources as number and  $pct$  to the amount of resources in percentage

	LUT		FF		DSP		BRAM	
<i>Total available resource</i>	<i>218600</i>		<i>437200</i>		<i>900</i>		<i>545</i>	
<i>Unit of measure</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>
Control-Synch. M.	1403	0.64	2063	0.47	0	0	8	1.47
Data-Flow M.	4948	2.26	8084	1.85	0	0	18.5	3.39
LPU	363	0.17	198	0.05	0	0	0	0
Glue logic	1347	0.62	1870	0.43	0	0	0	0
Debug Units	3272	1.50	5173	1.18	0	0	5.5	1.01
Data-Processing M.(std.)	73681	33.71	70722	16.18	1377	153	46	8.44
<b>TOP(std.)</b>	<b>85538</b>	<b>39.13</b>	<b>88376</b>	<b>20.21</b>	<b>1377</b>	<b>153</b>	<b>78</b>	<b>14.31</b>
Data-Processing M.(opt.)	5732	2.62	8039	1.84	45	5	11	2.02
<b>TOP(opt.)</b>	<b>17062</b>	<b>7.81</b>	<b>25430</b>	<b>5.82</b>	<b>45</b>	<b>5</b>	<b>43</b>	<b>7.89</b>

In order to compare the standard (std.) and the proposed optimized (opt.) pixel processing steps, Tab. 11.1 also reports the resource utilization of the CDAS architecture, including the Data-Processing Module (std.) that implements the pixel processing part according to the standard algorithm explained in Sec. 9.5. This comparison shows that while the optimized architecture uses only 45 DSPs, corresponding to 5% of the available DSPs, the architecture implementing the standard solution cannot even be implemented on the target MPSoC-FPGA, since it requires 1377 DSPs, corresponding to 153% of the available DSPs. Since the target MPSoC-FPGA can not contain the CDAS architecture with the Data-Processing Module (std.), the reported results for this configuration refer to the Vivado Design Suite 2021.2 post-synthesis report.

Moreover, the Control-Synchronization Module uses only 0.6%, 0.39%, and 1.46% of the LUTs, the FFs, and the BRAMs, respectively; this low resource utilization is also the result of the partitioning between real-time and non-real-time tasks implemented on PL and PS, respectively. Although such optimization may seem superfluous, it is essential to facilitate the addition of components to the CPS for plug-and-play capability and to extend the Data-Processing Module for further processing steps related to real-time support.

Finally, to demonstrate that the result for the LPU is independent of the specific settings applied in the KIDS-CT scanner use case, Section 11.4 presents its resource utilization analysis. This analysis considers various configurations where different numbers of PD and Memory Resources MR have been set at the design time.

### Timing Analysis

This Section discusses the timing analysis of the CDAS architecture. In order to meet the different timing requirements, the architecture has been implemented on the target MPSoC-FPGA with the clock domains reported in Tab. 11.2, as also explained in Section 7.4.2 (Inter-clock domains).

Table 11.2: Clock domains of the CDAS architecture for the KIDS-CT scanner

Clock domains	PS	DRAM	System	Reference (GTX)	Receiver	Transmitter
Frequency [MHz]	667	533.333	100	156.250	78.125	156.250
Clock domains	Data-Proc. Module		Register	Control-Synch. Module		
Frequency [MHz]	100 or 200		100	100 and 10		

While the clock domains associated with the Data-Flow Module are set to achieve a data rate at which the collected data can be forwarded without requiring external memory, as explained in Section 7.4, the other clock domains are set to meet the component communication and the data processing requirements, providing the real-time support. The Data-Processing Module can be set to 100 MHz and 200 MHz for this reason two values have been reported in tab. 11.2.

In order to use the KIDS-CT scanner for interventional procedures, the image should be reconstructed in real time, while the projections are being acquired. For this purpose, an optimization of the acquisition and processing datapath has been presented in Section 8.4 and shown in Fig. 8.10. Here, the CDAS is the key component where data are collected, processed, and forwarded on the fly. In order to perform these operations without additional latency to the acquisition, the sum of the latency of the Data-Flow Module and the Data-Processing System must be less than  $\Delta T$ , which represents the *integration period*. To use the KIDS-CT scanner for interventional procedures,  $\Delta T$  is usually set in the neighborhood of 1 *ms* and a minimum of 200  $\mu s$ . Consequently, the total latency of the Data-Flow Module and the Data-Processing System must be at least less than 1 *ms* to be integrated with no additional latency.

Table 11.3: Latency of the CDAS architecture for the KIDS-CT scanner

CDAS	Standard solution	Optimized solution
Data-Flow Module	205.8 <i>ns</i>	205.8 <i>ns</i>
Data-Processing Module	2780 <i>ns</i>	260 <i>ns</i>
Sum	2985.8 <i>ns</i>	465.8 <i>ns</i>

Thanks to the proposed optimizations of the Data-Flow Module and the Data-Processing Module, the total latency of the acquisition and processing datapath within the CDAS architecture is equal to 465.8 *ns*. Unlike, the total latency, where data are processed with the standard solution, is equal to 2985.8 *ns*, as reported in Tab. 11.3. This means that in the worst case, when the minimum *integration period* (i.e., 200 $\mu$ s) is used, the standard solution also fails to meet the timing requirements.

## 11.2 Data-Flow Module

The Dataflow-Flow Module consists of the proposed lightweight dataflow architecture described in Section 9.3. In addition to the configuration for the CDAS architecture, the same module has been reused in into the Reconstruction System (RS) architecture to collect data from the CDAS architecture in the stationary side of the KIDS-CT scanner. The Data-Flow Module for these systems has been configured with the design-time and run-time parameters reported in Tab. 9.1 and Tab. 9.2. In order to evaluate the impact on resource utilization, when an additional component such as a sensor/actuator is plugged into the CPS, the SC system was also implemented and evaluated. This has been implemented starting from the RS base architecture, but the Data-Flow Module has been configured as follows: 4 GTXs mapped to Small form-factor pluggable (SFP+) connectors in the Receiver Stage, 1 FIFOs in the Scheduler Stage, and 1 PCI-E transceiver with 4 lanes to stream data to the main memory of the workstation, where it is plugged in. This setup allows the SC system to reach a data rate of 25 Gbit/sec, while the CDAS and the RS are set at 6.250 Gbit/sec, according to the requirements of the KIDS-CT scanner.

### Resource utilization

The resource utilization for the different configurations (i.e., CDAS, RS, SC), including the Debug Units for post-implementation validation, has been reported in Tab. 11.4.

Table 11.4: Resource utilization of the Data-Flow Module. Post-implementation resource occupancy (Vivado report)

<i>Configurations</i>	<b>LUTs</b>			<b>FFs</b>			<b>BRAMs</b>		
	<i>CDAS</i>	<i>RS</i>	<i>SC</i>	<i>CDAS</i>	<i>RS</i>	<i>SC</i>	<i>CDAS</i>	<i>RS</i>	<i>SC</i>
Register file	139	163	163	581	596	596	0	0	0
Receiver S.	353	515	1192	549	704	1555	0	0	0
Data-link Prt. S.	173	91	246	158	101	190	0	0	0
Scheduler S.	1349	224	1189	1897	411	1621	8.5	28,5	114
Stream Gen. S.	31	0	0	24	0	0	0	0	0
Transmitter S.	531	11654	16915	626	13021	17271	0	20.5	29.5
Debug Units	438	201	213	161	124	123	0	0	0
Debug ILAs	2493	3442	3208	3722	6022	5385	10	9	9
<b>TOP</b>	<b>4949</b>	<b>16637</b>	<b>23572</b>	<b>8083</b>	<b>21028</b>	<b>27359</b>	<b>18.50</b>	<b>58</b>	<b>152.5</b>

Although the CDAS and the RS receive the same amount of data, the former uses two channels, while the latter uses only a single channel in both the Receiver Stage and the Data-link Protocol Stage. In the Receiver Stage, LUTs and FFs implement the logic for packet alignment and control of the GTXs; receivers use specific GTX resources that are hardwired into the MPSoC-FPGAs. Since the GTXs are configured for different data widths and protocols, a direct relationship between the number of channels and the resource utilization in this stage can not be defined. However, these stages have been compared for the SC and the RS configurations as they are integrated into the same base architecture. This comparison indicates that the SC configuration requires only 1.41 times more LUTs, 1.30 times more FFs, and 2.52 times more BRAMs compared to the RS configuration while achieving a data rate that is 4 times higher. The comparison of the different utilization for the various configurations is also shown in Fig. 11.1.

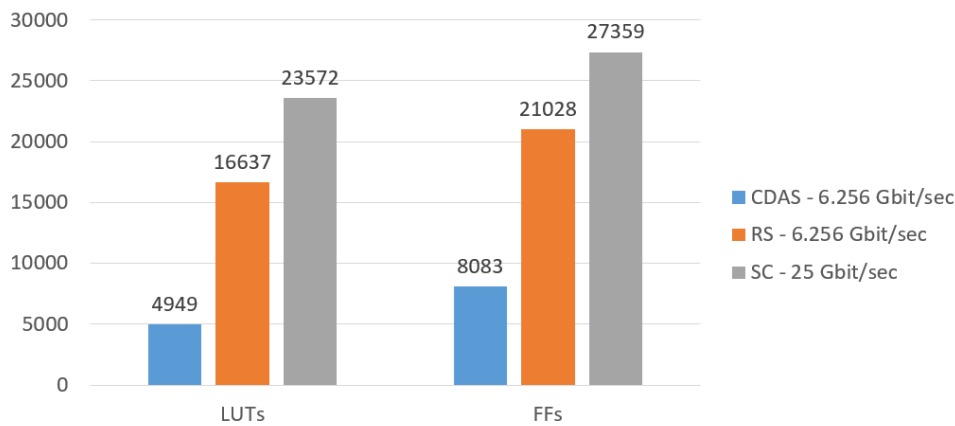


Figure 11.1: Resource utilization of the Data-Flow Module

Delving into the stages of the different configurations, we can notice the different amount of BRAMs in the Scheduler stage. The number of BRAMs depends on the "*number of FIFOs*" and their "*depth*". In the CDAS configuration, 8 independent FIFOs with the depth of 1 projection row have been configured. Although the size of a row is smaller than 1 BRAM of 36 Kbit, in order to implement asynchronous FIFOs, the entire BRAM is needed. Different BRAMs per FIFO are required in the design because each FIFO can be accessed simultaneously with different clock frequencies. The additional 0.5 BRAM is used by another asynchronous FIFO implemented in the Scheduler sub-stage to store the index of the selected FIFO for the enqueueing of projection rows. This way, even if projection rows are written out of order in FIFOs, their index is kept and then used to dequeue projections in order. In this case, 1 BRAM of 18 Kbits is sufficient, since only a small amount of data must be stored. In contrast to the CDAS, the RS and the SC configurations are set with 1 FIFO because they are set to the depth of 1 projection, according to the design parameters reported in Tab. 9.1. In this case, the number of BRAMs is determined by the depth of the FIFO configured to collect a single projection.

While the CDAS architecture needs to packet and send data with the Application Stream Protocol, the RS and the SC must collect these data and write them into the main memory of the connected workstation via PCI-E. Consequently, in these configurations the Stream Generator Stage has been disabled, resulting in its resource usage being effectively zero. Unlike, the Transmitter Stage of these two configurations, different transceivers are used for each configuration, which mainly affects resource utilization. For the RS and the SC configurations, a significant amount of resources are allocated to the PCI-E transceiver.

### 11.2.1 Timing Analysis

To evaluate the timing performance, an analytical model of latency was defined in addition to the simulation and test results given by the running KIDS-CT scanner. This model, based on the queueing theory [172], shows that under the condition in which  $F_{min}$  (Equation 7.1) is valid, the queues satisfy the “stability condition”, so FIFOs can be used because they are not permanently overloaded. Furthermore, the model provides an estimate of the WCET, which cannot be derived directly from the running system, where only the average packet latency is measurable.

To accurately estimate the WCET, which in this context is determined by the architectural latency, each stage must be taken into account. As explained in Section 9.3, the Data-link Protocol Stage and the Stream Generator Stage have a deterministic latency, which consists of 3 clock cycles in the implemented architecture. The latency of the Receiver Stage and the Transmitter Stage is given by two elements: the latency of the external communication, which can be omitted because it is on the order of nanoseconds, and the latency of the internal receiver and transmitter, which is of 6 clock cycles, as also shown in Tab. 11.5.

Table 11.5: Lightweight re-configurable dataflow architecture latency

	number of clock cycles	Latency [ns]		
		CDAS	RS	SC
		$\rho = 0.0064$	$\rho = 0.004$	$\rho = 0.016$
Receiver S.	6	76.8	38.4	38.4
Data-link Prt. S.	3	38.4	19.2	19.2
Allocator sub-stage (Sched. S.)	1	12.8	6.4	6.4
FIFO sub-stage (Sched. S.)	Eq. (11.1)	12.8	4	24
Select packet sub-stage (Sched. S.)	1	6.4	4	4
Stream Gen. S.	3	19.2	0	0
Transmitter S.	6	39.4	24	24
Sum of Latencies		205.8	96	116

The *Scheduler Stage* is the most complex stage to estimate the latency because a packet is enqueued in a FIFO, and the time at which it is dequeued is not deterministic. In

addition, due to the different FIFOs to schedule, the *Scheduler stage* has the largest packet latency, as shown in Tab. 11.5. In order to analytically calculate the average latency of a packet in a FIFO, the Equation (11.1) has been used.

$$Latency_{FIFO\text{Sub-stage}} = \frac{\rho}{1 - \rho} \frac{E(X^2)}{2E(X)} \quad (11.1)$$

In equation (11.1),  $E(X^2)$  and  $2E(X)$  represent the mean square and the mean of the message length. Equation (11.1) takes the message length because the packets are enqueued in FIFOs and are scheduled when the entire message is received and committed. In Equation (11.1),  $\rho = (\mu/\lambda)$  represents the average fraction of time in which the *Scheduler stage* is occupied and can not accept new messages;  $\rho < 1$  defines the “stability condition” of the queues;  $1/\mu$  is the *mean arrival time*, and  $1/\lambda$  is the *mean transmission time* of the packets. To avoid losing packets,  $\rho$  must be less than 1.

In the KIDS-CT scanner, the DMS [22] has an *integration period* of  $\sim 200 \mu\text{sec}$ , and it generates a maximum of  $\sim 200$  packets per *integration period*. This means that  $1/\mu$  (packet mean arrival time) is  $1000 \text{ ns}$  in the worst case. Since the SC configuration can collect data from 4 DMSs,  $1/\mu$  is equal to  $250 \text{ ns}$ . Then, the dataflow architecture transmits one packet per clock cycle, so  $1/\lambda$  (mean transmission time) depends on the *data clock* of the transmitters. This means that  $1/\lambda$  is equal to the associated transmitter *data clock* period, which is  $6.4 \text{ ns}$  for the CDAS transceiver and  $4 \text{ ns}$  for both the RS and the SC transceivers. Therefore, the  $\rho$  of the CDAS, the RS, and the SC are equal to 0.0064, 0.004, and 0.016, respectively, which are less than 1. Thus, we can conclude that the architecture is capable of consuming data faster than producing them for all configurations. Therefore, the buffers meet the “stability condition”, and the number of FIFOs can be set according to  $F_{min}$ , avoiding external memory for the buffering.

Finally, to calculate the packet latency for the whole dataflow architecture, the contribution of each sub-stage has been considered using the following equation (11.2):

$$Latency = T_{Rx} + (6 + 3 + 1)P_{Rx} + \frac{\rho}{1 - \rho} \frac{E(X^2)}{2E(X)} P_{FIFO} + (1 + 3 + 6)P_{Tx} + T_{Tx} \quad (11.2)$$

In equation 11.2,  $P_{Tx}$  and  $P_{Rx}$  represent the clock period of the transmitter and receiver *data clocks*.  $P_{FIFO}$  is the clock period of the FIFOs, which in this case corresponds to  $P_{Tx}$ . These variables are multiplied by the contributions of the various stages shown in Tab. 11.5. This Table also shows the latency of each stage, which is calculated by multiplying the period of the transmitter/receiver data clock by their clock cycles. The value given in this Table does not consider the communication latency, which is expressed in Equation (11.2)



by the variables  $T_{Rx}$  and  $T_{Tx}$ . This latency can be neglected for Optical Fibre<sup>1</sup> links because it is typically a few nanoseconds. For calculating the amount of clock cycles spent by each pixel in the FIFO sub-stage, the Equation 11.1 has been used, resulting in the following values: 1.6 for the CDAS, 0.4 for the RS and 5.5 for the SC. Since these values represent the number of clock cycles, they have been rounded to 2, 1, and 6, respectively.

The total latency of the different configurations is given in Tab. 11.5, where the RS has the lowest latency. Although the RS and SC have the same clock periods, the latency of the FIFO sub-stage depends on the value of  $\rho$ , which is the maximum in the SC configuration. However, by comparing the latency and the data rate of the RS and the SC, it can be seen that although the SC has a 4 times higher data-rate, the latency is only a factor of 1.2. This result is important because it shows that by scaling the architecture and increasing the data-rate, the latency is slightly affected.

### 11.3 Data-Processing Module

As explained in Section 11.1, the performance of the Data-Processing Module affects the acquisition datapath latency of the CDAS architecture. Due to the lack of DSP resources, the standard solution can not be implemented on the target MPSoC-FPGA because it requires 153% of the available resources, as shown in Tab. 11.6. Instead, the proposed optimization solution using only simple Multiplications (MULs), Additions (ADDs) and Subtractions (SUBs) that fulfill the available resources of the selected MPSoC-FPGA, using only 5% of the DSPs, as shown in Tab. 11.7. In both tables, each row refers to the 4-pixel parallel processing. For instance, the Cosine & Redundancy Weighting requires 812 DSPs for four pixels, so each pixel requires 203 DSPs.

Table 11.6: Resource utilisation of the Data-Processing Module with the standard solution

Standard solution	LUTs		FFs		DSPs		BRAMs	
	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>
Unit of measure								
Stage 1: Detector-Data Conv.	154	0.07	145	0.03	0	0.00	0	0.00
Stage 2: I0-Correction	29746	13.61	29513	6.75	565	62.78	4	0.73
Stage 2: Weighting	43716	20.00	40967	9.37	812	90.22	42	7.71
Stage 3: Processed-Data Conv.	32	0.01	49	0.01	0	0.00	0	0.00
Glue logic	14	0.01	8	0.00	0	0.00	0	0.00
<b>TOP</b>	<b>73681</b>	<b>33.71</b>	<b>70722</b>	<b>16.18</b>	<b>1377</b>	<b>153</b>	<b>46</b>	<b>8.44</b>

When we analyze the optimized solution reported in Tab. 11.7 and compare it with the standard solution in Tab. 11.6, we can see the enormous difference in terms of resource utilization. In fact, in the optimized solution of the I0-correction step, only 1 MUL and 1 SUB have been implemented, while the standard solution requires 4 MULs, 2 Exponentials (EXPs), 1 Division (DIV) and 1 Logarithm (LOG), consuming an enormous

<sup>1</sup>Due to the confidential nature of this matter, the exact information cannot be disclosed.

amount of LUTs, FFs and DSPs. The same observation can be made for the Cosine & Redundancy Weighting step, where the optimized solution requires only 2 MULs, while the standard solution requires 4 MULs, 2 Powers (POWs), 1 Cosine (COS), 1 ADD, and 1 DIV. Between these different operations, as shown in Tab. 11.8 also Truncate (TRUNC) operations are executed to cut off the resulting numbers, which exceed the representable mantissa. Instead in the conversion stage, the Casting (CAST) operation converts data formats.

Table 11.7: Resource utilisation of the Data-Processing Module with the optimized solution

<b>Optimized solution</b>	<b>LUTs</b>		<b>FFs</b>		<b>DSPs</b>		<b>BRAMs</b>	
	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>
Stage 1: Detector-Data Conv.	135	0.06	127	0.03	0	0.00	0	0.00
Stage 2: I0-Correction	2673	1.22	3706	0.85	21	2.33	3	0.55
Stage 2: Weighting	2635	1.21	4084	0.93	24	2.67	8	1.47
Stage 3: Processed-Data Conv.	32	0.01	48	0.01	0	0.00	0	0.00
Glue logic	254	0.12	341	0.08	0	0.00	0	0.00
<b>TOP</b>	<b>5732</b>	<b>2.62</b>	<b>8039</b>	<b>1.84</b>	<b>45</b>	<b>5.00</b>	<b>11</b>	<b>2.02</b>

Although the standard solution does not use BRAM resources to store the pre-calculated weights, it uses more than 4 times the number of BRAM to buffer the data processing datapath, which is 10 times deeper than the optimized solution. For this reason, it is also not efficient in terms of memory usage, making it not a suitable solution for MPSoC-FPGA. Since the optimized solution uses only 2.62 % of LUTs, 1.85 % of FFs, 5 % of DSPs, and 2.02 % of BRAMs, it is really efficient in terms of resource utilization and offers the possibility to extend the CDAS architecture for additional processing steps and components to plug into CPS. In fact, optimizing resource utilization has been a key constraint in the design flow of the whole CDAS architecture.

The optimized solution improves not only resource utilization but also latency, which is a key parameter for real-time support. As shown in Tab 11.8, the standard version has a high latency not only due to the number of operations but especially for their complexity. In fact, LOG and POW have latencies of 31 and 76 clock cycles, respectively. Moreover, the table only shows the latency of the critical path in the standard solution, because in Stage 1 there is another parallel path that performs a MUL and an EXP, and in Stage 2 before the last two MULs, there is a parallel path that performs one MUL, one WRITE and one COS. For instance, the COS operation has a latency of 23 clock cycles. For this evaluation, the synthesis has been set to 100 MHz, so the latencies are equal to 260 *ns* and 2780 *ns*.

It is also important to observe that the same operation can have two different latencies, such as the MUL operation in Stage 2 which has 6 clock cycles, while other MUL operations have only 4 clock cycles. These different latencies result from the implementation binding

Table 11.8: Latency of the Data-Processing Module

	<b>Optimized Sol.</b>		<b>Standard Sol.</b>	
	<i>Operation</i>	<i>c.c.</i>	<i>Operation</i>	<i>c.c.</i>
<b>Stage 1</b>	READ	1	READ	1
	CAST	5	CAST	5
<b>Stage 2: I0-Correction</b>	SUB	5	MUL	4
			BUF	2
			EXP	18
			MUL	6
			TRUNC	2
	MUL	4	DIV	12
			TRUNC	2
			LOG	31
			TRUNC	2
			BUF	2
<b>Stage 2: Cosine &amp; Redundancy Weighting</b>	MUL	4	MUL	4
			WRITE	1
			POW	76
			ADD	7
			TRUNC	2
	MUL	4	POW	76
			TRUNC	2
			DIV	12
			MUL	4
			MUL	4
<b>Stage 3</b>	CAST	2	CAST	2
	WRITE	1	WRITE	1
<b>Sum</b>		<b>26</b>		<b>278</b>

used in Vitis™ HLS, which offers the option to implement the same operation with fabric (i.e., LUT and FF), or DSP, or a hybrid solution, as explained in Ref. [77].

## 11.4 Lightweight Protection Unit

In order to evaluate the LPU, this Section considers the resource utilization of its internal hardware units after it has been integrated into the CDAS architecture for the KIDS-CT scanner. As reported in Tab. 11.9, most of the FFs are utilized by the configuration block, which consists of the lookup table into which APs are written at run time. Since the Policy Check unit is implemented using combinatorial logic, decisions are made within the clock cycle, using only LUTs resources. From this first result, it follows that with the proposed method and architecture, it is possible to implement a solution where transactions are granted/denied without adding latency since each transaction is checked and granted/denied during the transmission, within a clock cycle.

Besides implementing the LPU for the KIDS-CT scanner, three additional primary

Table 11.9: Resource utilization for the test design

	LUTs	FFs	DSPs	BRAMs	LUTRAMs
AXI demux	66	28	0	0	0
Configuration block	80	141	0	0	0
Policy check (write)	87	0	0	0	0
Policy check (read)	75	0	0	0	0
AXI error	29	29	0	0	0
Glue logic	27	0	0	0	0
TOP	363	198	0	0	0

configurations were examined. This was done to demonstrate that the proposed LPU architecture is well-suited for any generic low-cost MPSoC-FPGA. The first with LPUs placed on the master side, the second with LPUs placed on the slave side, and another with a single LPU placed between two AXI-Interconnect blocks, as shown in Fig. 11.2. In the first configuration, each master component is connected to an LPU that is responsible for managing transactions from that specific master. This setup, as shown in Fig. 11.2a, ensures that only the addresses within the associated MRs are checked and accessed, potentially avoiding unwanted transactions and mitigating congestion in the downstream interconnect by blocking transactions before they pass through the interconnect unit.

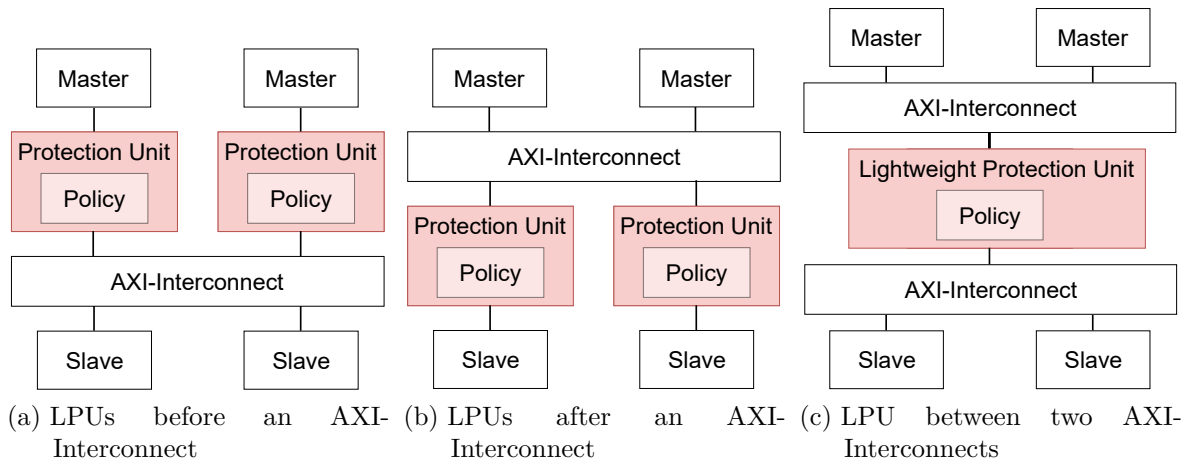


Figure 11.2: Exemplary deployment of LPU

Conversely, placing the LPU on the slave side, as shown in Fig. 11.2b, assigns a distinct LPU to each slave, which is only responsible for the MRs of its respective slave. However, in the third arrangement, as shown in Fig. 11.2c, since only a single LPU is used alongside two AXI-Interconnect buses, this solution results to be resource-efficient, especially when the number of masters and slaves is balanced and limited.

In order to show that the proposed LPUs is suitable for low-cost MPSoC-FPGA, the presented placements were also evaluated on the XC7Z020 MPSoC-FPGA, which has almost four times less resources than the XC7Z045 MPSoC-FPGA targeted for the

KIDS-CT scanner. In the second design, the clock frequency was set to 100 MHz, and various configurations with different numbers of PDs and MRs were evaluated. Additionally, as detailed in Tab. 11.10, this Section includes a comparative analysis of resource utilization. This analysis includes the MPPU proposed by Kornaros et al. [113] and the HIMM proposed by Kumar Saha and Bobda [114].

Table 11.10: Resources' utilization and latency (c.c. means clock's cycle)

	<b>LUTs</b>	<b>FFs</b>	<b>LUTRAMs</b>	<b>Latency</b>
LPU, 1PD/16MRs	339	198	0	<10 ns (<1 c.c.)
LPU, 16PDs/1MR	191	198	0	<10 ns (<1 c.c.)
LPU, 16PDs/16MRs	950	678	0	<10 ns (<1 c.c.)
LPU, 1PD/1MR	164	168	0	<10 ns (< 1 c.c.)
MPPU [113]	655	1082	12	(4 c.c.) only the decision
HIMM [114]	86	75	6	220-35000 ns

In the comparison, the proposed solution performs much better than the MPPU solution in terms of resource utilization, while it performs better than the HIMM solution only in terms of time (i.e., latency). Because the HIMM solution stores the access policy in the main memory, it must access this memory whenever access is requested for a new peripheral. This large range in access time given due to the use of external memory makes this solution the worst for real-time use cases, such as CPS with MCS requirements (e.g., the KIDS-CT scanner). In fact, for real-time applications, the WCET should be considered as latency, which is equal to 35000 *ns*, while the proposed solution can be considered as a zero-latency solution. This great timing result has been achieved because the decision path of the Policy check units is shorter than the critical path. Furthermore, the proposed LPU allows an area optimization because PDs and MRs are set at design time. This allows the Policy check unit to be designed using only combinatorial logic and to be implemented using only with LUTs.

## 12 Design Space Exploration

This Chapter presents the image quality analysis performed on the reconstructed CT images, taking into account the selected parameters and metrics presented in Section 9.6 for the DSE. Section 12.1 gives an overview of the prerequisites to be met and addresses the problem of selecting significant data that can be used to characterize the image quality independently of the human body being scanned. In addition, it explains how quantitative image quality parameters, such as low-contrast, uniformity, and noise, have been expressed and estimated with a quantitative approach. Then, Section 12.2 and Section 12.3 discuss the evaluation results regarding the selected metrics and the different selected data formats, respectively. These results have also been essential to validate the proposed optimization of the pixel processing part and the different implementations where data formats are tuned. In fact, the results of the image quality values have been compared with a GPU implementation of the same algorithm, from which this work started for all the proposed data processing optimizations. Finally, Section 12.4 discusses the results of the DSE pointing out which data formats are best suited for iCT procedures. The content of this Chapter has been partially discussed by the author in Ref. [DP 1].

### 12.1 Image Quality Prerequisites

Most work in the literature considers different CT scanners and/or acquisition configurations to investigate how those influence image quality [173–175]. However, this work aims to find out the influence of data formats on image quality, independent of the CT scanning configuration. Therefore, to perform the image quality analysis required for the DSE and to achieve significant results that can be used with the proposed metrics and are representative of interventional CT procedures, the following conditions must be met:

1. Using the same CT scanning configuration.
2. Using a phantom, which is a representative object for medical procedures. In this case, this must be representative of the image characterization independently of the specific human body part to be acquired during a medical procedure.
3. Using a quantitative method for the selected metrics to fairly estimate the image quality.

### 12.1.1 CT scanning configuration

In order to acquire and reconstruct the images with the different data formats, the experiment was conducted without turning off the scanner between the different acquisitions. In this way, the X-ray tube and the DMS are calibrated with the same values between the different acquisitions. In addition, by performing all the acquisitions in the same window, environmental interferences should affect the different acquisitions in the same way.

Furthermore, to perform the I0-correction step, an axial full-projection acquisition, without objects is required, as explained in Sec. 2.4.3. Since this image is affected by the calibration parameters and the environmental interferences, conducting the experiment as explained above allows to use the same I0-projections for the whole experiment. Fig. 12.1 shows an I0-projection stored in *short* data format and collected during during the experiment.



Figure 12.1: I0-image: Image without object

Finally, the KIDS-CT scanner was set up with the standard acquisition parameters typically used with the selected DMS and shown in Tab. 12.1.

Table 12.1: KIDS-CT scanner: scanning parameters

	<b>CT scanning parameters</b>	<b>Settings</b>
<b>DMS</b>	number of row slices	64
	x and y slice width	0.7 mm <sup>1</sup>
	number projections per round	1000 <sup>1</sup>
<b>Gantry</b>	rotation speed	1 round per second
<b>X-ray tube system</b>	voltage	120 KV
	intensity	250 mA
<b>RS</b>	reconstruction algorithm	FDK algorithm [63]
	z slice width	1 mm
	size of the reconstructed image	512 x 512 pixels
<b>*note</b>	<i>Additional setting parameters can not be disclosed due to the confidential nature of the information</i>	

Although parameters could be selected or tuned to achieve better image quality, this research only investigates the influence of data formats on image quality and performance. Therefore, only these parameters were tuned in the experiment to explore the design space.

<sup>1</sup>This parameter has been generalized due to the confidentiality nature of the information

### 12.1.2 Phantom selection

The CATPHAN® 500 [163] was chosen for the experiments. This has 4 modules housed in a 20 cm case, as shown in Fig. 12.2. Each module is used to perform a different image quality metric such as the *Geometry Alignment*, the uniformity, the noise, and the low-contrast of the reconstructed image. Before describing the modules, it is important to introduce the Hounsfield Unit (HU), also known as “*CT number*” It is the relative quantitative measure of radio-density [176]. Radiologists use the HU to interpret the CT images. Since different body tissues have different densities, this value can be used to identify the different soft tissues and bones.

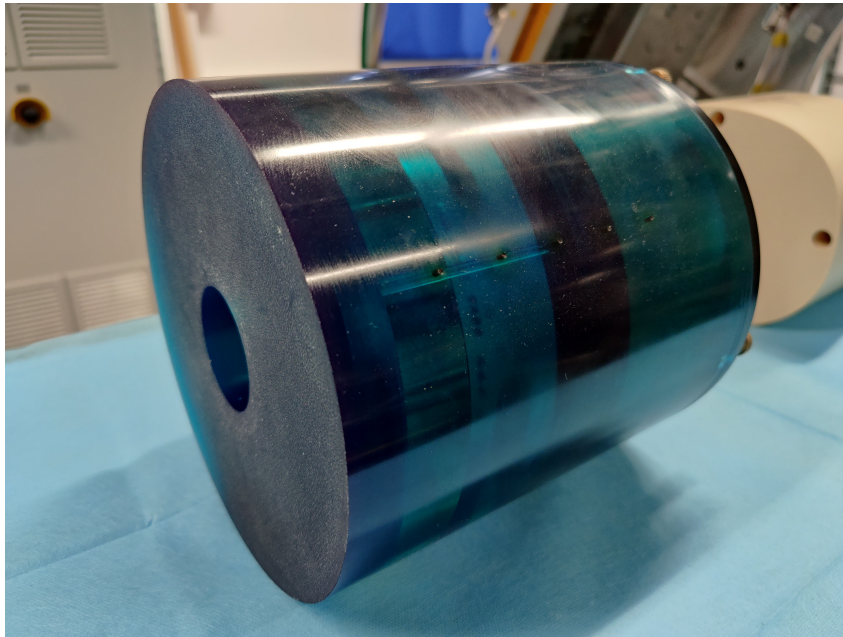


Figure 12.2: CATPHAN® 500 [163] phantom

Among the four modules, only the following were used to perform the selected image quality metrics:

- **CTP515 Low-Contrast Module:** This module is composed of cylindrical rods varying in diameter and featuring three levels of contrast, intended for assessing low-contrast performance as detailed in [163]. As shown in Fig. [cat500Low], these rods are placed along the *z*-axis to avoid volume averaging errors [177]. The different low-contrast points are useful for identifying small low-contrast parts in the image, such as tumors in interventional procedures. The module includes sub-slice targets with a standard contrast of 1.0% and lengths on the *z*-axis of 3, 5, and 7 mm. For each of these lengths, there are targets with diameters of 3, 5, 7, and 9 mm [177]. This phantom module is required to perform the low-contrast image quality metric to find out if different data formats affect the different levels of contrast in the reconstructed image.



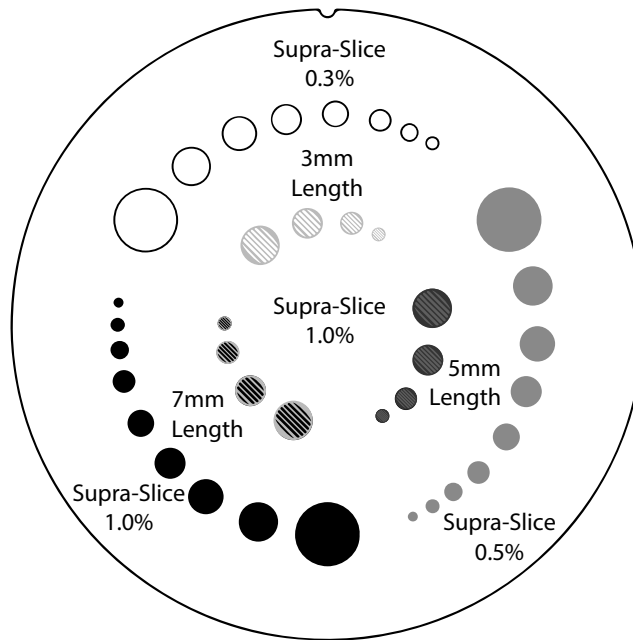


Figure 12.3: Section of the CTP515 Low-Contrast Module [177]

- CTP486 Uniformity Module:** This module is made of a uniform material that has a “*CT number*” within 2% of the water density, according to standard scanning protocols [163]. It is used to measure and then estimate spatial uniformity and noise. As shown in Fig. 12.4, this module contains various ROI for assessing the uniformity across different areas of a phantom section. In order to evaluate the accuracy of the *CT number* in comparison to the expected value, the average and standard deviation of a large number of points within a specific ROI of the scanned image are calculated.

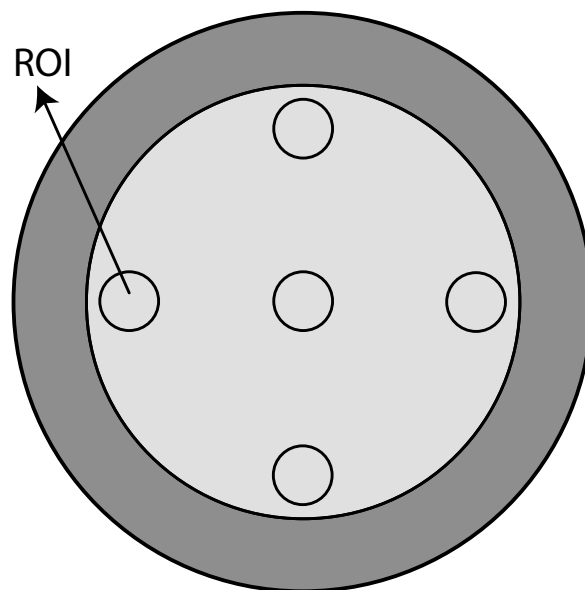


Figure 12.4: CTP486 Uniformity Module [177]

- **CTP401 Slice Geometry and Sensitometry Module:** This module is used to verify the phantom position. As shown in Fig. 12.5, the module includes five acrylic spheres designed to assess the scanner's ability to image sub-slice spherical volumes. These spheres have diameters of 2 mm, 4 mm, 6 mm, 8 mm, and 10 mm. This particular phantom was chosen for human visual analysis of the CT images, considering the various materials and sizes of the spheres.

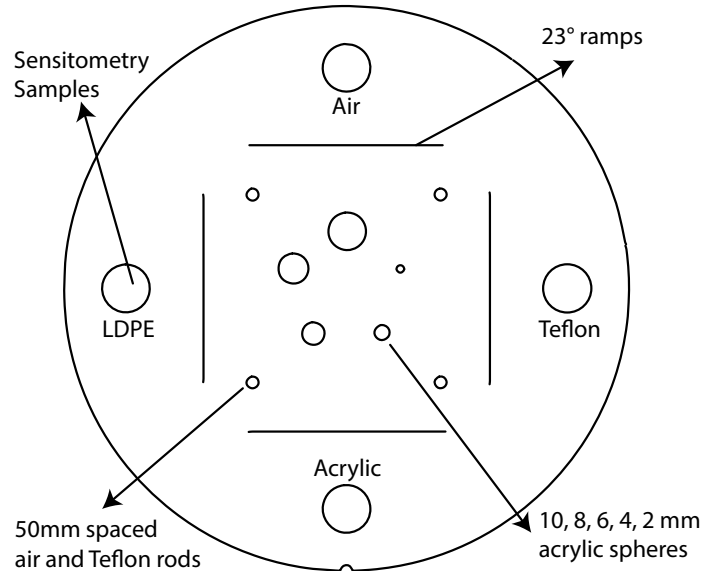


Figure 12.5: CTP401 Slice Geometry and Sensitometry Module [177]

### 12.1.3 Calculation of the image quality metrics

For each image quality parameter, a mathematical value of it was calculated. For estimating the pixel error of the 2D projections, the **MSE** was calculated, because it gives the error interpretation of the approximated image [161]. The equation of MSE is the following:

$$MSE = \frac{1}{V} \sum_{j=1}^V (A_j - S_j)^2 \quad (12.1)$$

Here,  $A_j$  is the pixel value of a reference image, and  $S_j$  is the pixel value of the image to be analyzed [160]. As reference image, the pre-processed image with *double* format was selected, as discussed in Section 9.6.

For the calculation of the noise, uniformity, and low-contrast values related to the reconstructed volumes, different ROIs per module were considered, as suggested in the CATPHAN® 500 manual [163]. The identification of the ROIs allows to extract from the images only the values related to the material of interest. In the specific case, the ROIs shown in Fig. 12.6 with red and blue circles were selected from the reconstructed images. Furthermore, to compare the values related to the different data formats, the material

values contained in the phantom and a reference volume per module were reconstructed. The reference volume was reconstructed using the Generic Computed Tomography (GCT) toolkit, which is a GPU-accelerated image reconstruction and processing toolkit for CT that includes different detectors and projection geometries [178, 179].

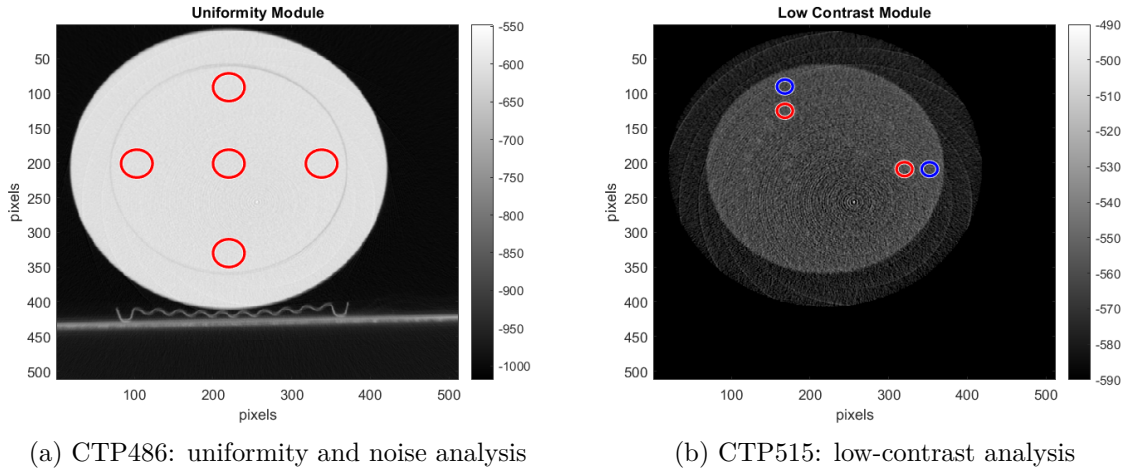


Figure 12.6: Placements of the ROIs

For the **low-contrast** analysis, the CTP515 module was used, considering the ROIs marked in Fig. 12.6. A 20-pixel diameter ROI was placed inside a larger ROI to calculate the ratio between the two contrast areas. For this analysis, both the 1.0% supra-slice and the 0.5% supra-slice were considered. The Contrast Noise Ratio (CNR) was then estimated using equation 12.2, where 32 reconstructed slices were averaged. In this way, the 32 slices contain different contrast values to be analyzed.

$$CNR = \frac{|S_A - S_B|}{\sigma_B} \quad (12.2)$$

In equation 12.2,  $S_A$  and  $S_B$  are the signal intensities of the target supra-slice and the background regions, respectively, and  $\sigma_B$  is the standard deviation of the background.

The **noise** was calculated using the standard deviation of the *CT numbers* of the ROIs of the CPT846 shown in Fig. 12.6. This module has five ROIs with a diameter of 40 pixels. While the *Noise* was estimated using the standard deviation, the **uniformity** was estimated by calculating the maximum difference between the mean value of the central ROI and the mean value of one of the peripheral ROIs. In this context, the mean value corresponds to the average of the *CT numbers* in HU of the pixels within a ROI.

## 12.2 Image Quality Analysis

Before analyzing the results of the different data format configurations with the selected metrics, a human visual analysis of the CTP401 module was performed. With the human

visual analysis, we can observe that the grid between the sensors has been removed by the pixel processing steps, and the resulting images look good with the proposed optimizations. Fig. 12.7 shows a projection of the CTP401 module before and after the pixel processing steps using the different data format configurations.

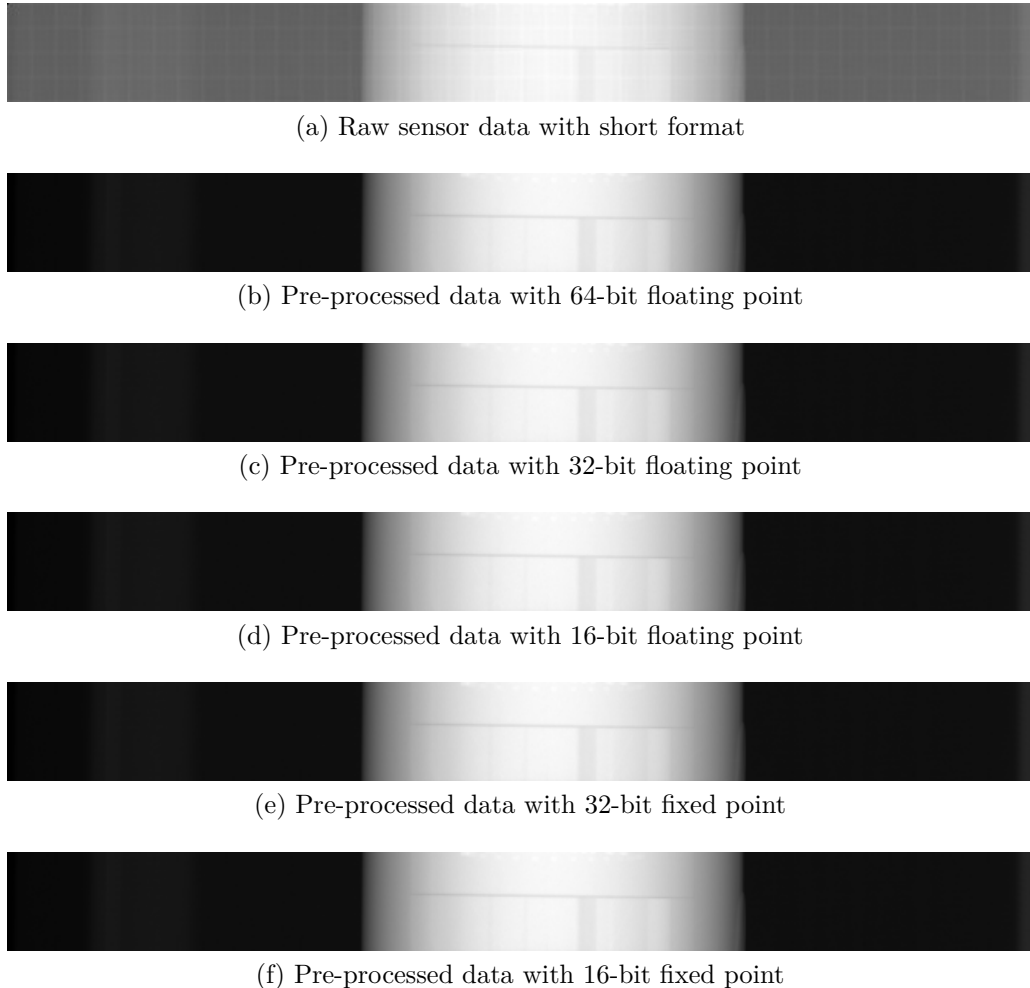


Figure 12.7: Projection of the CTP401 module, before and after the pixel processing part

Since no differences were observed by the human eye between the different configurations, the MSE between the 64-bit floating point and the other formats was calculated for the projections after the pixel processing part. As explained in Section 9.6, the MSE of the projections is the key metric used to reduce the selected data formats for the DSE, since it allows a rough estimation of the image quality without performing the reconstruction and other image quality analysis that require a reconstructed image to be estimated and take more time than calculating the MSE. Specifically, the MSE was used to select the two 16-bit and 32-bit fixed-point format configurations, reducing the number of possible data format configurations from 48 to 2.

As reported in Tab. 12.2, the MSE of the different data formats applied to the processed

Table 12.2: MSE of the 2D projections

MSE	FP64	FP32	FP16	FXP32	FXP16
CTP486	0.0	8.35e-16	7.84e-08	9.29e-04	5.38e-02
CTP401	0.0	8.72e-16	9.92e-08	1.01e-04	5.88e-02
CTP515	0.0	8.53e-16	7.89e-08	9.73e-04	5.64e-02

projections ranges from  $10^{-16}$  for 32-bit floating point to  $10^{-02}$  for 16-bit fixed point<sup>2</sup>. These values are very good in terms of image quality since the MSE is usually between 2.36 and 2.37 also in medical image compression [180]. Moreover, we can observe that the MSE is of the same order of magnitude independently of the selected module, but it changes with respect to the format used for the data processing. This means that it varies not in relation to the specific scanned phantom but with the selected data representation.

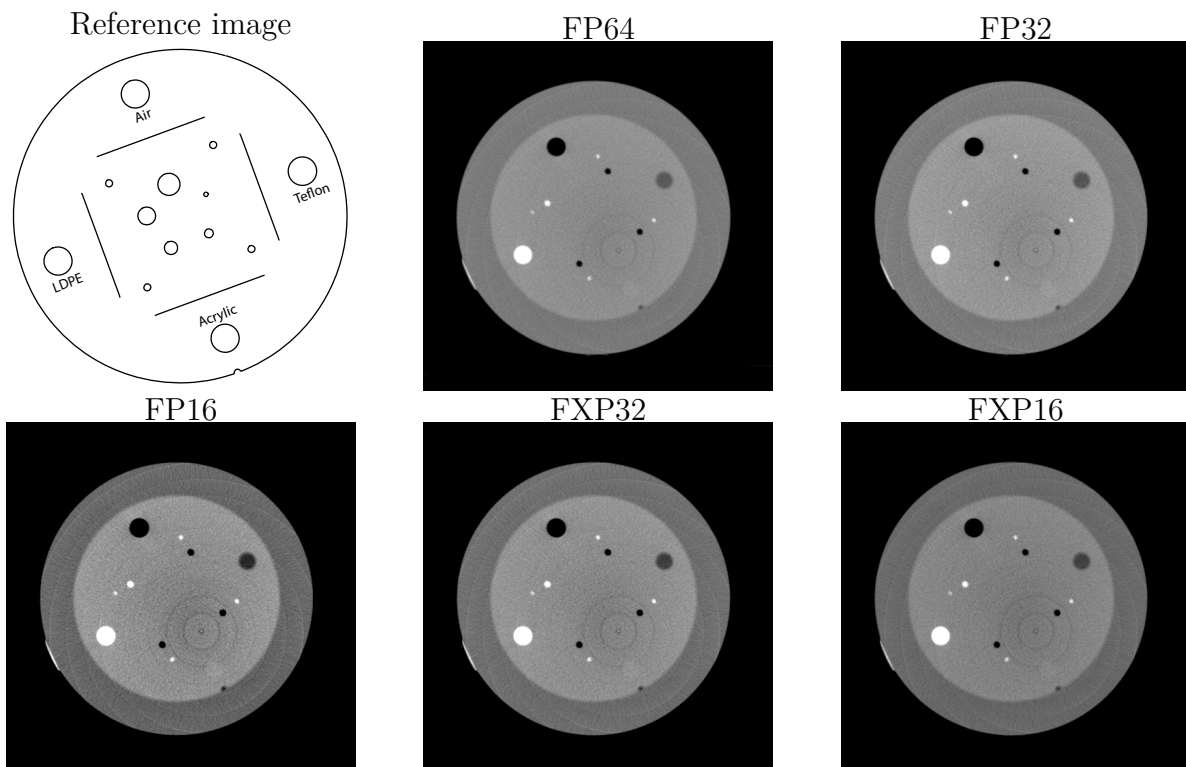


Figure 12.8: Reconstructed image of the CTP401 module for human visual analysis

After this initial analysis, which showed that the selected format can be used for the DSE, the reconstruction was performed for all modules and all data formats to analyze the reconstructed 3D images, which are typically used by radiologists for diagnosis and by surgeons during the interventional procedures. Human visual analysis was also performed on the reconstructed images, as shown in Fig. 12.8. In the figure, we can see that all target materials (i.e., Air, Low-Density PolyEthylene (LDPE), Teflon, and Acrylic) can be

<sup>2</sup>All tables in the thesis report data formats as follow: 64-bit floating point (FP64), 32-bit floating point (FP32), 16-bit floating point (FP16), 32-bit fixed point (FXP32), 16-bit fixed point (FXP16)

distinguished independently of the data format configuration used in the pixel processing steps. This means that the image quality does not seem to change in terms of human visual analysis. Therefore, the MSE was also calculated on the reconstructed images to understand how the pixel processing part influences them when this is performed with different data formats. In this case, as shown in Tab. 12.3, the MSE related to all configurations is less than  $10^{-5}$ , which can be considered close to 0 in this application.

Table 12.3: MSE of the 3D volumes

MSE	GCT	FP64	FP32	FP16	FXP32	FXP16
CTP486	0.0	9.79e-08	9.79e-08	9.83e-08	2.42e-07	8.72e-06
CTP515	0.0	1.01e-07	1.01e-07	1.02e-07	2.53e-07	9.14e-06
CTP401	0.0	1.12e-07	1.12e-07	1.12e-07	2.75e-07	9.58e-06

However, the MSE does not consider all image quality metrics that are relevant to medical procedures. Therefore, in order to understand how data formats affect the quality of reconstructed images, measurements of low-contrast, noise, and uniformity were performed. The reconstructed images needed to calculate these metrics are shown in figures 12.9 and 12.10.

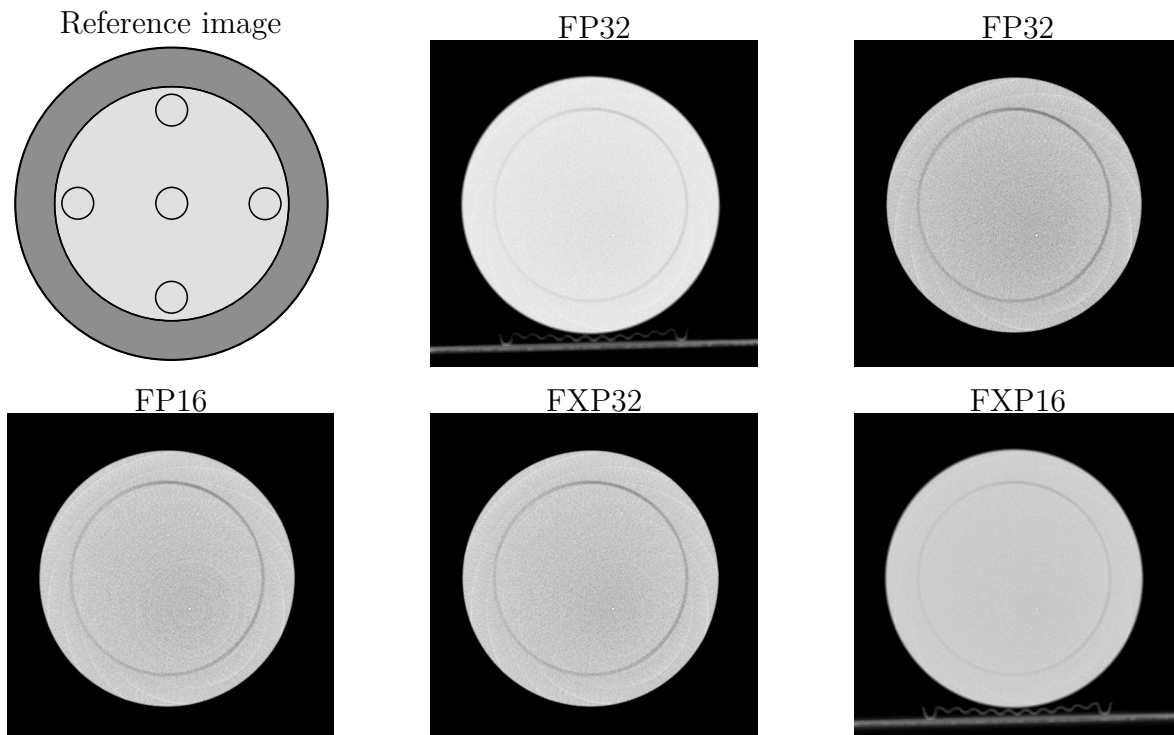


Figure 12.9: Reconstructed image of the CTP486 module, for noise and uniformity analysis

In this case, the same images were also reconstructed pre-processed and reconstructed from scratch with GCT in order to compare the resulting values of the materials in HU for the different processing solutions and data formats. Although the reference values provided with the CATPHAN® 500 are typically used to understand the quality of a

detector sensor (i.e., DMS sensors) from the reconstructed images, this research uses these reference values to understand the impact of the data format on the reconstructed images. In addition, the FDK itself can affect the image quality during reconstruction, so the best approach is to compare the images reconstructed with GCT, without and with the proposed optimizations, with the different data formats. The GCT algorithm was performed with float format, using the “NVIDIA A100 40GB PCIe” GPU, which is designed with the NVIDIA ampere architecture [181]. This information is important to know because even if the same algorithm uses the same data format, the resulting images can have different values because the implemented floating-point arithmetic units can use different rounding and approximations of the real number.

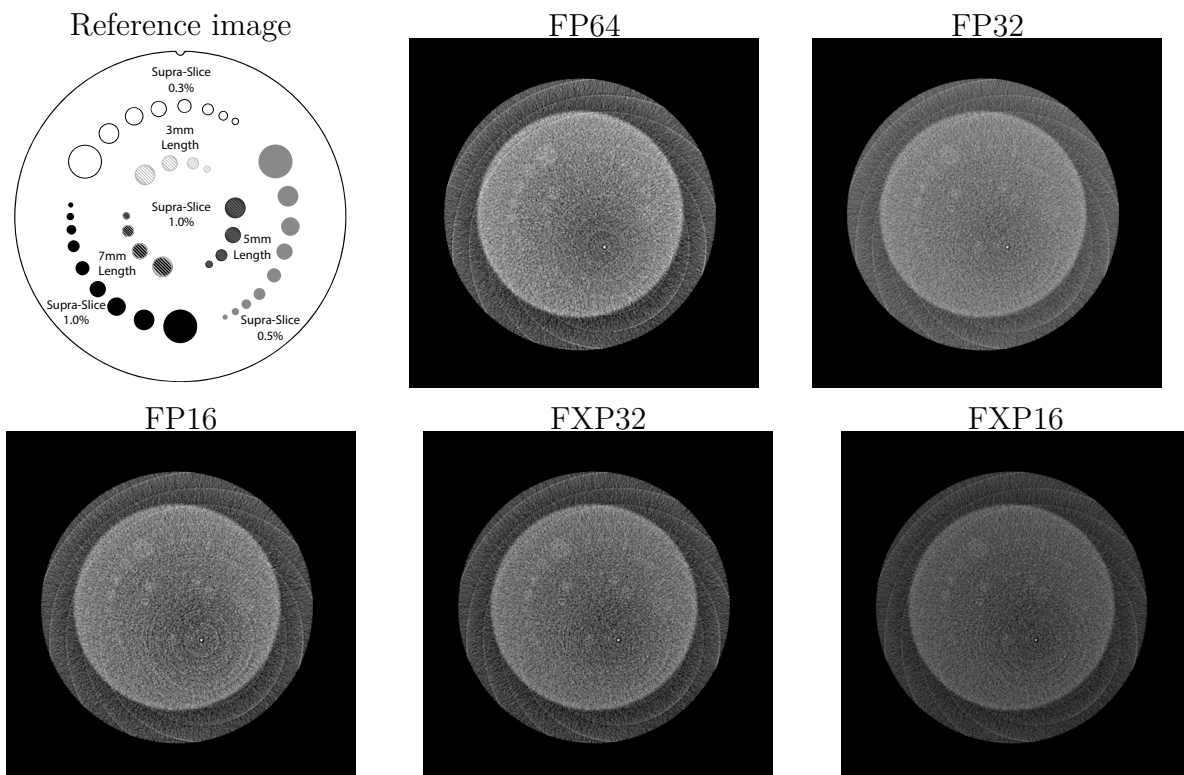


Figure 12.10: Reconstructed image of the CTP515 module, for low-contrast analysis

The results of the image quality analysis are reported in Tab. 12.4. Here we can see that for all the metrics and the materials, the values of the selected data formats are close to the GCT results, except for the 16-bit fixed point. Specifically, for all formats other than 16-bit fixed-point, the values for uniformity and noise deviate from the GCT by a maximum of 8.28% and 21.15%, respectively. On the other hand, the values for the 16-bit fixed-point format show a deviation of 50.9% and 57.19% from the GCT for uniformity and noise, respectively. Furthermore, LDPE and Teflon materials expressed in HU can not be identified in the volume reconstructed from the 16-bit fixed-point format, whereas they are close to the GCT results in the volumes reconstructed from other formats.

From this image quality analysis, it can be concluded that all data formats result in

Table 12.4: Image quality analysis

	<b>GCT</b>	<b>FP64</b>	<b>FP32</b>	<b>FP16</b>	<b>FXP32</b>	<b>FXP16</b>
Uniformity	13.95	13.65	13.65	14.20	12.75	6.85
Noise	12.96	10.94	10.94	11.42	10.22	5.55
CNR	1.08	1.28	1.28	1.23	1.28	1.35
Air in [HU]	-901.44	-898.44	-898.44	-898.56	-905.12	-949.41
Teflon in [HU]	863.69	867.25	867.25	867.13	744.54	-66.59
LDPE in [HU]	-76.08	-77.37	-77.37	-77.45	-138.01	-538.81

acceptable image quality after pixel processing and reconstruction steps, with the exception of the 16-bit fixed point. As this format results in an image where materials cannot be identified and noise and uniformity are low only due to the blur effect of the image, it cannot be used for interventional applications where tumors need to be identified.

### 12.3 Hardware Cost & Computing Performance

As discussed in Section 9.6, to determine the optimal data format for the pixel processing part of the reconstruction algorithm for interventional CT procedures, it is essential to consider not only the image quality but also the computing performance (i.e., latency) and the hardware cost (i.e., resource utilization) as metrics of the selected formats.

Before analyzing the values in detail, it is important to inform the reader that no `pragma` has been used in Vitis™ HLS to specify the mapping of arithmetic operations either to DSPs or to fabrics (e.g., LUT and FF). For example, the pragma “`BIND_OP`” may be used to optimize latency and/or resource utilizations for a deputed arithmetic operation by binding it to DSP or fabric. Consequently, the use of this pragma can lead to different results for each configuration in terms of distribution of the resource utilization between fabric and DSP within the same configuration. Consequently, the usage of DSP and fabric affect the latency in the order of a few clock cycles per operation.

In this case, where pragmas were not been used, Vitis™ HLS balanced between resource utilization and latency of the Data-Processing Module. For example, to balance this usage in the fixed-point representations, the two MUL operations in the Cosine & Redundancy Weighting were implemented with DSPs, while the SUB and the other MUL in the I0-correction step were implemented with fabric. This is important to note because the two different implementations of the same MUL result in completely different amounts of resource utilization and latency.

The resulting resource utilization and latency are shown in Tab. 12.5. Here, we can see that the resource utilization of the implementation performed with floating-point representations uses more LUT, FF, DSP resources than the selected fixed-point representations, independently of the two selected precisions.

Comparing the resource utilization among the different formats, we observe that although



Table 12.5: Resource utilization and latency used for the DSE. In the table the resource utilization and the latency of the stage 1 and 3 have been included in the I0-correction and Weighting steps, respectively

<i>Unit of measure</i>	<b>LUTs</b>		<b>FFs</b>		<b>DSPs</b>		<b>BRAMs</b>		<b>Latency</b>
	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>n</i>	<i>pct</i>	<i>c.c.</i>
<b>FXP16: TOP</b>	<b>3212</b>	<b>1.47</b>	<b>2812</b>	<b>0.64</b>	<b>8</b>	<b>0.89</b>	<b>8</b>	<b>1.47</b>	<b>14</b>
Interconnection	254	0.12	341	0.08	0	0.00	0	0.00	-
I0-Correction	376	0.17	174	0.04	0	0.00	1	0.18	5
Weighting	2586	1.18	2084	0.48	8	0.89	7	1.28	9
<b>FXP32: TOP</b>	<b>4054</b>	<b>1.85</b>	<b>4111</b>	<b>0.94</b>	<b>33</b>	<b>3.67</b>	<b>11</b>	<b>2.02</b>	<b>13</b>
Interconnection	254	0.12	341	0.08	0	0.00	0	0.00	-
I0-Correction	804	0.37	1113	0.25	1	0.11	3	0.55	3
Weighting	2995	1.37	2624	0.60	33	3.67	8	1.47	10
<b>FP16: TOP</b>	<b>4773</b>	<b>2.18</b>	<b>6644</b>	<b>1.52</b>	<b>33</b>	<b>3.67</b>	<b>10</b>	<b>1.83</b>	<b>24</b>
Interconnection	254	0.12	341	0.08	0	0.00	0	0.00	-
I0-Correction	2520	1.15	3630	0.83	16	1.78	3	0.55	15
Weighting	1996	0.91	2640	0.60	17	1.89	7	1.28	9
<b>FP32: TOP</b>	<b>5732</b>	<b>2.62</b>	<b>8039</b>	<b>1.84</b>	<b>45</b>	<b>5.00</b>	<b>11</b>	<b>2.02</b>	<b>26</b>
Interconnection	254	0.12	341	0.08	0	0.00	0	0.00	-
I0-Correction	2808	1.28	3833	0.88	21	2.33	3	0.55	15
Weighting	2667	1.22	4132	0.95	24	2.67	8	1.47	11
<b>FP64: TOP</b>	<b>7753</b>	<b>3.55</b>	<b>12070</b>	<b>2.76</b>	<b>145</b>	<b>16.11</b>	<b>11</b>	<b>2.02</b>	<b>34</b>
Interconnection	252	0.12	341	0.08	0	0.00	0	0.00	-
I0-Correction	4839	2.21	6517	1.49	57	6.33	3	0.55	19
Weighting	2667	1.22	5185	1.19	88	9.78	8	1.47	15

these formats are designed to double the width for both representations, the resource utilization for FFs and LUTs only increases by a factor of 1.5 between them. In contrast, the DSP usage increases by a factor of 3.2 when comparing 32-bit to 64-bit floating-point representations, and by a factor of 4.1 when comparing 16-bit to 32-bit fixed-point representations.

Although different strategies may yield varying results, the resource utilization and latency reported in Tab. 12.5 are significant in the context of this DSE. These findings allow designers to comprehend the relation between data formats in terms of computational performance and their impact on the overall design. In particular, optimizing DSP usage and latency becomes crucial when the CDAS architecture needs to be extended to be integrated into an autonomous CT scanner, as presented in the patent in Ref. [DP 8]. It also becomes necessary to extend the Data-Flow Module to perform the slice processing part (i.e., ramp filtering) on the fly, as this demands a large amount of resources and requires performing operations row by row before calculating an output result.

## 12.4 Design Space Exploration Considerations

Even though the best data format in terms of resource utilization is the 16-bit fixed point, due to the low image quality found with the selected metric, this format can not be used for interventional procedures. On one hand, among the remaining formats, the best one is the 32-bit fixed-point, which optimizes computing performance and resource utilization while keeping the image quality acceptable for interventional procedures; this can be implemented with the lowest number of DSP, LUT and FF, and it is 2 times faster than the other suitable formats. On the other hand, the best data format that optimizes the image quality while keeping the hardware costs is the 16-bit floating point, which also has the lowest DSP resource utilization, but the image quality is higher, penalizing the latency compared to the 32-bit fixed-point representation. Therefore the designer can choose between these two data formats, in relation to the specific requirements of the CT scanner and internal components.

## 13 Evaluation Of Functionalities

This Chapter evaluates the provided functionalities, particularly the plug-and-play capability and the real-time support, which represent the main objective of this research work. Section 13.1 explains how the proposed work facilitates the plugging of a new DMS into the KIDS-CT scanner, providing the plug-and-play capability. Section 13.2 analyzes how the proposed work enables real-time support for the KIDS-CT scanner. Finally, Section 13.3 compares the proposed CDAS with related work solutions in terms of functionalities. The content of this Chapter is further discussed in the articles in Ref. [DP 1, DP 2, DP 3, DP 4, DP 5, DP 6, DP 7].

### 13.1 Plug-and-Play Capability

This Section validates and evaluates how the proposed work can provide plug-and-play capability in a CPS. Since this capability cannot be measured or evaluated with quantitative metrics such as latency and resource utilization, this thesis analyzes the effort to add a new component in a plug-and-play fashion within a CPS. Specifically, the author focuses on the target KIDS-CT scanner, which has been modeled with the proposed System Architecture, and the Communication Infrastructure, which includes the CDAS architecture. The effort required to add a new component such as a DMS is considered. To highlight the differences and advantages of the proposed solution over others, this Section also considers the effort to add the same DMS to a generic CT scanner.

For this purpose, the AMD-Xilinx CT architecture presented in Chapter 3 was chosen as a meaningful comparison. In contrast to the centralized approach of the proposed work, the AMD-Xilinx solution uses a distributed approach. In fact, in the AMD-Xilinx solution, the CS is distributed on the *System Sequencer*, on the *HV Supply Control*, and on the *Data Acquisition & Gantry Control*, and the DAS is also distributed on the *Data Acquisition & Gantry Control* [128].

The XC-Thor photon counting detector [182] was selected as an additional DMS for the KIDS-CT scanner to investigate the plug-and-play capability. It is a flat panel detector with two interfaces: a Gigabit Ethernet interface and a custom interface. The former supports two communication modes for real-time data and non-real-time tasks, while the latter consists of dedicated custom signals associated with real-time synchronization/control tasks.

The use of the Gigabit Ethernet interface for both real-time data and non-real-time tasks makes the integration into the AMD-Xilinx CT architecture infeasible without incorporating an additional component that routes signals/data to the various CSs and DASs. This is because the tasks associated with this interface are implemented in separate physical architectures in the AMD-Xilinx solution. It also means that the task requirements must already be considered by the designer at the system design stage. In contrast, with the proposed solution, all these interfaces can be directly connected to the CDAS without considering the associated tasks during this step. The former can be connected to the available Gigabit Ethernet interface, and the latter to a connector wired to the GPIO pins of the MPSoC-FPGA. These pins support different input/output standards and are usually described as signals in the HDL design.

In the specific, the process of integrating this DMS into the KIDS-CT scanner begins by adding a new slave-client node to the System Architecture and matching its interfaces to the proposed classes of the Communication Infrastructure. This mapping step treats standard and custom interfaces in the same way, as the mapping is independent of the type of protocol and task. Once the interfaces have been mapped to the classes, they must be identified at the *transport protocol layer* and *application protocol layer*. This pre-matching step is essential to understand in which module of the CDAS architecture the transceivers and software drivers of the different interfaces should be integrated.

The transceiver for the custom interface associated with the *real-time control interface class* is instantiated in the Control-Synchronization Module of the CDAS. It can be encapsulated in the proposed Communication Unit, which also implements the Handshake Protocol. In this way, at the *application protocol layer*, the Handshake Protocol can be used to handle errors and lost messages. Both the *application non-real-time class* and the *application real-time data class* can concurrently access the Gigabit Ethernet interface and use the proposed Application Datagram Protocol and Application Stream Protocol, respectively.

In addition to the effort required to add instances for various interfaces and protocols in the *communication interface layer* and the *transport protocol layer* of the CDAS, these must also be mapped to the PS part as I/O memory-mapped devices. Furthermore, an additional module for the new DMS must be instantiated on the software layer. This module contains the communication component for the server and the control logic for the new detector, which is not discussed here because it is component-specific and is usually provided by the vendor.

When the new module is initialized, a configuration file is defined for the DMS. This file contains connection settings, thread scheduling, and necessary data packets tailored to the vendor and system requirements. As a result, the server module is configured to operate independently of the first DMS on a single thread of the CDAS. To facilitate communication with the PL and PS components of the extension in the existing device

tree, new entries containing the IDs and the physical addresses of the new hardware units must be integrated. Within the software layer, the four server modules are adapted to the new DMS according to the manufacturer's specifications.

Communication with the control side is established via the Gigabit Ethernet interface, and the configuration file in the communication module defines the maximum number of clients.

The commands for the new DMS are integrated into the structure of the Execute module and linked to the CDAS PL based on the existing device-tree setup. Alongside the Execute module, an operation ID for communication is included in the command datagram. New or custom datagram structures and the decode/encode lookup tables must also be defined from the configuration file. The Operation IDs are mapped to the Execute module commands and scheduling options are configured.

The final step is to set the Timer module into the Control-Synchronization Module at the Software layer of the CDAS associated to the timeout of the component (specified by the vendor). It's important to note that all hardware/software extensions only affect the CDAS and not the other components. However, in the AMD-Xilinx architectural model, all components are involved due to the implementation of HV power control, sequencer, data acquisition, and gantry control tasks on different physical components, as in most related work solutions. These solutions result in higher integration effort and cost of solutions in related work and the inability of them to provide plug-and-play functionality.

## 13.2 Real-Time Support

As discussed in the different Parts of this thesis, the real-time support in a CPS is the result of different contributions, from the centralized System Architecture and the related Communication Infrastructure to the micro-architecture of the modules within the CDAS, where also tasks have been mapped taking into account this requirement.

It is not possible to provide an absolute quantitative metric for a real-time CPS, independent of the specific use case application. However, common features that a real-time CPS should include can be outlined. Some of these features provided by the proposed work for real-time support can be partially found in various related works that also provide real-time support. For example, the centralized solution is also common to the ADAS solution proposed by Tesla[28].

Specifically in the proposed **System Architecture**, the centralized control architecture has allowed to realize the proposed Synchronization Unit, where errors can be caught and handled in a few clock cycles, whereas a distributed architecture would take much more time because of the additional external communications are involved. Moreover, with this solution, all the different tasks can communicate within the CDAS. Therefore, the WCET

is affected by external communication, which usually introduces an additional stochastic element in the timing estimation.

In the **Communication Infrastructure**, the division of the nodes into classes has also contributed to the real-time support, because it allows the separation of the real-time and non-real-time communication tasks, avoiding missed deadlines due to the interference of non-real-time tasks. Furthermore, the additional separation between real-time control-synchronization and data tasks also facilitates the estimation of the WCET and avoids the convoy effect that can be caused by the huge difference in size of control and data tasks. In fact, data tasks, such as acquisition tasks in the KIDS-CT scanner, are considered long-running tasks that can cause control-synchronization tasks to stall.

In the **CDAS** architecture, the proposed mapping method also contributes to the real-time support. In this method, task groups are partitioned into different modules for each node, and then real-time and non-real-time tasks within each module are separated and mapped to the PL and PS, respectively. In this way, tasks with different requirements are separated, and those that must meet real-time requirements are implemented directly on the PL, which usually facilitates the WCET estimation. In the PL, the execution path and timing of a task can be precisely controlled and predicted because the hardware logic is explicitly defined for the task. Within the CDAS architecture, the lightweight architecture of the **Data-Flow Module** and the **Data-Processing Module** also contribute to the real-time support of on-the-fly data processing. This feature not only enables real-time data acquisition and processing but also optimizes the acquisition and processing datapath in the case of the KIDS-CT scanner, providing a solution where no latency is added. This is achieved by taking advantage of the *integration period*. In fact, this time is used by the Data-Flow Module to reduce the data and by the Data-Processing Module to perform the pixel processing steps and then send the data to the Reconstruction System. Moreover, the LPU provides a solution that analyzes the AXI transaction during communication without adding latency to the critical path. In this way, the isolation is provided with no timing penalty, unlike most of the related work that can be implemented on low-cost MPSoC-FPGA.

All these elements together contribute to providing the plug-and-play real-time CDAS architecture for MPSoC-FPGAs targeting CPS with MCS requirements such as the KIDS-CT scanner.

### 13.3 Comparison With Related Work

In order to compare the proposed and the related work, the different functionalities and performance have been analyzed. While the LPU as a standalone architecture has already been compared to related work in Section 11.4, the CDAS architecture has not been

compared with related work. Although resource utilization and latency are not provided for all the related work, the comparison has been made in terms of functionality and supported data rate for collected data, as reported in Tab. 13.1.

Table 13.1: Related work comparison. The acronyms in the table are the following: System Architecture (SA), Centralized Architecture (C), Distrusted Architecture (D), not specified (n.s.), at synthesis time(s.t.), at run time (r.t.), not applicable (n.a)

	SA	Independent acquisition streams	Maximum tested bandwidth [Gbps]	Re-configurability	Buffer memory type	Data Proc.	CS & DAS	real-time	iso-lation	exten-sibi-ly	Application
Tesla [28]	C	n.s.	n.s.	✗	n.s	✓	CDAS	✓	n.s.	✗	automotive
Intel-Altera [122]	D	n.s.	n.s.	✗	off-chip	✓	CS, DAS	✗	n.s	✗	diagnostic CT
AMD-Xilinx [128]	D	n.s.	n.s.	✗	off-chip	✓	CS, DAS	✗	n.s	✗	diagnostic CT
Eltec [102]	n.a.	✗	n.s.	✗	off-chip	✗	DAS	✗	n.s.	✗	diagnostic CT, image camera
Marjanovic [96]		✗	1.024	s.t.	off-chip	✓	CDAS	✓	n.s.	✗	MRI
Yang [97]		✗	n.s.	✗	off-chip	✓	CS, DAS	✓	n.s.	✗	image acquisition
Shi [98]		✗	2.25	✗	off-chip	✗	CS, DAS	✓	✗	✗	image acquisition
Xie [99]		✗	7.2	✗	off-chip	✗	DAS	n.s.	✗	✗	radar
Flouzat [100]		✗	3.2	✗	off-chip	DAS	✓	✓	✓	n.s.	automotive
Salgoro [101]		✗	8	✗	off-chip	✗	DAS	✗	n.s.	✗	PET
(proposed work)	C	✓	37.5	s.t. & r.t.	on-chip	✓	CDAS	✓	✓	✓	multimodality CT, iCT, generic CPS

In terms of features, as shown in Tab. 13.1, the proposed architecture is the only one able to acquire simultaneously different streams with different protocols, to offer extensibility of the system. In addition, it achieves the highest data rate between the related work for which this information has been shared. While the other DASs collect data and store them in off-chip memory, the proposed CDAS uses only in-chip memory, acquiring and processing data on the fly. Isolation is also a problem that most of the works do not consider or do not provide information about. Instead, the proposed CDAS has been designed considering the isolation problem which is essential for targeting MCS. Finally, all the CS, DAS and CDAS found are closed systems that can not be configured for other applications. In fact, among the proposed works that target medical applications, it is not possible to extend them to multimodality techniques or other medical devices. Although there are related works where CDAS architectures have been proposed, these are closed systems that can not be configurable with the exception of the CDAS for MRI proposed by Marjanovic et al. in Ref. [96]. The proposed system is the only one that can be configured at design time, but it only reaches 1.024 Gbps and cannot be extended to add other components.



## **Part V**

### **Finale**



## 14 Conclusion

This Chapter provides a brief summary of the research contributions. It then discusses the results in relation to the objectives formulated in Section 4.2. Finally, opportunities for future improvement and research are highlighted.

### 14.1 Summary

After having analyzed the weaknesses of the current architectures, targeting CPSs and CT applications, this research work has proposed a generic CDAS architecture for MPSoC-FPGAs, providing real-time support and plug-and-play capability. In addition, it has also proposed a System Architecture, which is based on a centralized solution and the related Communication Infrastructure, which is organized in “classes” and “layers”. These form the basis for the realization of the proposed CDAS architecture and its integration in a CPS device, such as the KIDS-CT scanner, thereby achieving the research objectives.

In order to define a System Architecture, a Communication Infrastructure, and a CDAS suitable for complex CPSs such as the KIDS-CT scanner, this research work has developed a methodology for defining and classifying application and design requirements, starting from the application modes/functionalities. At each step, *common* and *specific* requirements are considered separately. The former defines the requirements that are shared across different CPS functionalities, while the latter considers constraints for specific functionalities and potential extensions (i.e., plug-and-play capability). From the common requirements identified, tasks have been divided into the control-synchronization group, the data-flow group, and the data-processing group. However, this methodology has also been applied to the KIDS-CT scanner to identify specific requirements for the different modes, such as diagnostic CT procedures, interventional CT procedures, including multimodality techniques for exploring these procedures, which can not be done with commercial CT scanners.

Before considering the CDAS architecture, the System Architecture has been modeled by separating the *physical level* and the *logical level*. The former consists of two types of nodes: the master-server node consists of the CDAS, and the client-slave node represents sensors/actuators, user interfaces, and DPSs. The latter has been modeled with a multi-tier architecture that permits the logical separation of sensors/actuators from user interfaces and processing units, facilitating the containment of failures. Furthermore, the separation

into levels facilitates extensibility and interoperability, which are key features for providing the desired plug-and-play capability.

Associated to the System Architecture to establish the communication between nodes a Communication Infrastructure has been proposed. This is composed of layers, each with three classes. The classes facilitate real-time support; these permit the separation of real-time control-synchronization, real-time data and non-real-time communication tasks within each layer, satisfying the different requirements and avoiding interferences between them. Furthermore, the communication stack is defined by the following layers: communication interface, transport protocol, and application protocol. For this last layer, a *application protocol layer* per class has also been proposed to unify the different vendor transport protocols. This layered organization permits the CDAS to communicate with components from different vendors, contributing to the plug-and-play capability.

The defined three task groups take into account the functionality of the tasks but do not consider their timing requirements. Therefore, a task partitioning methodology has been also proposed to design the CDAS architecture for real-time support. First, tasks per node belonging to different groups are partitioned into different modules. Then, the tasks per node within each group are divided into real-time and non-real-time tasks, which are mapped onto the PL and PS, respectively. According to the groups, the CDAS architecture has been designed with three main hardware/software modules: the Control-Synchronization Module, the Data-Flow Module, and the Data-Processing Module, while the architecture has been designed on three layers according to the partitioning: the Hardware layer, the OS layer and the Application layer. Each module is implemented along these layers in relation to the timing constraints.

In the CDAS architecture, the Control-Synchronization Module implements the business logic, the server, and the different synchronization and communication units. The Data-Flow Module consists of a lightweight configurable dataflow architecture, which collects data, reduces, and forwards them on the fly. In contrast to related work, this architecture does not require external memory because the timing constraints are met by using different clock domains between the collected and processed data to send. In addition, this architecture can be extended to increase the number of receivers/transmitters at design time and supports different data rates and acquisition modes at run time. The collected data can also be processed within the CDAS architecture, in the Data-Processing Module. This consists of a lightweight architecture that can integrate processing steps implemented with a dataflow paradigm, as it is designed to process data on the fly. This architecture has been designed to integrate data processing steps and configure them for different data formats. In fact, it has also been used to explore the design space for the pixel processing part of the FDK reconstruction algorithm. The DSE aimed to find the best data format for interventional CT procedures considering computing performance, hardware cost, and image quality analysis metrics. In addition, to accelerate the FDK algorithm various

optimizations have been proposed to perform the pixel processing part on the fly. Finally, the CDAS architecture integrates the LPU, which enables the task isolation for MCS implemented low-cost MPSoC-FPGAs, such as the KIDS-CT scanner.

For the implementation in the KIDS-CT scanner, the AMD-Xilinx ZC706 Evaluation Board including the XC7Z045 MPSoC-FPGA was chosen. After successfully implementing the CDAS architecture on the MPSoC-FPGA, the proposed work was validated and evaluated both before and after its integration into the KIDS-CT scanner.

## 14.2 Discussion Of Results

This work has been started with the aim to answer research questions presented in Chapter 4 and to reach a main research object:

*"Find out how CS and DAS architectures can be combined into a CDAS architecture for MPSoC-FPGA in order to provide real-time support and plug-and-play capability in complex CPSs such as the KIDS-CT scanner"*

This research work can be concluded by stating that the research objective has been achieved. The research work has effectively addressed the posed questions through the proposed methodologies and approaches. Specifically, the following results that have been obtained from the various contributions permit to answer the proposed questions:

- **Plug-and-play capability:** The System Architecture, the Communication Infrastructure, and the CDAS provide flexibility and interoperability at the system, communication, and control level to add an additional component. This functionality has been evaluated by comparing how an additional detector can be plugged into the existing KIDS-CT scanner and into a generic CT scanner which is based on the AMD-Xilinx solution. A critical metric that has been considered to enable the CDAS architecture for plug-and-play is resource utilization since a new component in the KIDS-CT scanner requires additional modules (i.e., hardware units in the PL) on the CDAS architecture to control it and/or collect/process/send data. For this reason, the proposed optimizations for the pixel processing part were crucial to implement the CDAS into the KIDS-CT scanner with the target MPSoC-FPGA due to the limitations of the DSPs. In fact, while the standard implementation requires 39.13% of LUTs, 20.21% of FFs, 153% of DSPs, and 14.31% of BRAMs, leading to implementation failure, the architecture implemented with all the proposed optimizations requires only 7.81% of LUTs, 5.82% of FFs, 5% of DSPs, and 7.89% of BRAMs.
- **Real-time support:** The centralized solution of the System Architecture enhances the estimation of the WCET, particularly when communication between tasks as-

sociated with different components is involved. This improvement is due to the elimination of external communication requirements. Additionally, the Communication Infrastructure, with its various classes per task tailored to their specific time and data size constraints, effectively preventing the “convoy effect”. This is crucial, as the “convoy effect” can lead real-time communication tasks to miss deadlines. By statically mapping real-time tasks onto the PL of the CDAS architecture, the execution time can be predicted with greater accuracy compared to mapping them onto the PS, where a scheduler is also required. Furthermore, the proposed lightweight architectures for the Data-Flow Module and the Data-Processing Module acquire and process data on the fly without using any external memory. As a result, the Data-Flow Module achieves an estimated latency of 205.8 *ns*, while the Data-Processing Module achieves an estimated latency of 260 *ns*, also thanks to the proposed optimizations of the FDK pixel processing part. As a result, the total latency from pixel acquisition to the reconstruction system is approximately 468.8 *ns*, excluding the link latency, which is also in the order of nanoseconds. Since the *integration period* is on the order of 200  $\mu s$ , the CDAS does not add any latency because the reconstruction system must wait for the “integration period” to receive the successive projection. This means that the proposed solution performs the pixel-processing without adding latency to the acquisition time. In addition, by pre-processing this part of the algorithm during the acquisition, also the final reconstruction algorithm running on GPU has been improved because it only performs the remaining part of the FDK algorithm.

- **Isolation in low-cost MPSoC-FPGAs:** The resulting LPU architecture implemented on the XC7Z045 MPSoC-FPGA and the XC7Z020 MPSoC-FPGA achieved better resource utilization compared to the MPPU solution and also better latency than the HMM solution which has a range between 220 *ns* and 35000 *ns*. While other solutions store the police in the external memory, which has a low access prediction, the proposed LPU stores it in FFs and uses only LUTs to implement the decision logic. In this way, it also achieves a solution where no latency is added because the decision path is shorter than the critical path in the implemented design. Finally, the low resource utilization makes it the best solution for low-cost MPSoC-FPGAs, since it uses only 0.17% of LUTs and 0.05% of FFs.
- **Data format in interventional CT:** The 16-bit fixed point is the most resource-efficient, but it does not meet the requirements for CT reconstruction due to its low image quality. The 32-bit fixed point, on the other hand, balances computing performance, resource utilization, and acceptable image quality for these procedures. It uses fewer resources than other suitable formats and is twice as fast as other suitable formats. Alternatively, the 16-bit floating point is excellent for optimizing

image quality and hardware cost efficiency, but while it has the same DSP resource utilization as the 32-bit fixed-point representation, its latency is higher in comparison.

- **KIDS-CT scanner:** Finally, all these results have contributed to the realization of the KIDS-CT scanner, which is the first open-interface CT scanner assembled in Academia and designed for exploring multimodality techniques and iCT procedures.

### 14.3 Future Work: Adaptive Computing Acceleration Platforms For CDAS

As this thesis is close to the end, several future explorations emerge, particularly in relation to the novel Adaptive Computing Acceleration Platforms (ACAPs) such as the Versal architecture [130]. These advanced platforms present exciting opportunities for enhancing and extending the proposed CDAS architecture, especially in the field of medical devices and automotive applications within autonomous CPSs.

One particularly promising application lies in the development of autonomous CT scanners. The integration of complex reconstruction algorithms and autonomous functionalities in CT scanners represents a significant challenge due to the current computing limitations of targeted MPSoC-FPGA platforms. However, recent advancements in ACAPs, which have begun to appear in both academic literature and the market, offer a potential solution. These platforms are rapidly evolving to support the acceleration of real-time, embedded applications, including the complex CPSs targeted in this thesis.

Although currently, these platforms are sometimes cost-prohibitive and not widely adopted in the market, they are fascinating architecture to explore in future research. In fact, this work has identified these architectures as promising platforms for extending the CDAS architecture. A key application could be in the reduction of X-ray doses during interventional procedures. By integrating autonomous algorithms into the CDAS, it may be possible to adjust scanning parameters in real time, based on tumor and needle positioning, as also described by the author of this thesis in the patent in Ref. [DP 8].

Furthermore, these advanced platforms could be leveraged to enhance the CDAS within the KIDS-CT scanner. They offer the potential to support new reconstruction algorithms based on AI or to further explore the FDK algorithm, potentially enabling its full implementation within the CDAS.

However, these opportunities are not without their challenges. The complexity and heterogeneity of these new platforms present obstacles that require thorough investigation. Future research should focus on understanding and overcoming these challenges to optimize and expand the CDAS architecture. This would not only contribute to the field of CDAS architecture and ACAPs but also significantly advance the capabilities of autonomous medical and automotive CPS applications.

## List of Figures

2.1	Integration of computation and physical environment . . . . .	7
2.3	MPSoC-FPGA Micro-architecture: Zynq-7000 SoC and Cyclone V SoC . .	10
2.4	AXI4: channel architecture of read operations [36] . . . . .	12
2.5	AXI4: channel architecture of write operations [36] . . . . .	12
2.6	Representation of the KIDS-CT scanner . . . . .	16
2.7	Example of a 4 voxel object and its projections . . . . .	20
2.8	CT circular trajectory . . . . .	20
2.9	World Coordinate System . . . . .	21
2.10	Pixel Coordinate System and Detector Coordinate System . . . . .	22
2.11	Fan Arc Angle (left) and Fan line height (right) . . . . .	22
2.12	FDK algorithm steps . . . . .	24
2.13	X-ray intersects the circular focal spot twice in full scanning . . . . .	26
2.14	Illustration of the ramp filter $h_{ramp}$ . . . . .	26
2.15	Encodings of the floating-point standard . . . . .	28
2.16	Fixed-point representation . . . . .	29
2.17	Fixed-point representation methods . . . . .	29
3.1	Low-cost C-shaped PET architecture . . . . .	33
3.2	Configurable DAS architecture . . . . .	34
3.3	Tesla FSD: Block diagram architecture . . . . .	35
3.4	Swiss cheese model . . . . .	36
3.5	XPPU functional diagram . . . . .	37
3.6	Network-on-Chip Firewall . . . . .	39
3.7	MPPU: example of deployment [113] . . . . .	39
3.8	Hardware/Software IP management modules overview . . . . .	40
3.9	Altera-Intel CT scanner architecture . . . . .	41
3.10	AMD-Xilinx CT scanner architecture . . . . .	43
5.1	Application classification and requirement definition steps . . . . .	52
6.1	System Architecture of a CPS . . . . .	56
6.2	System Architecture: Physical level . . . . .	57
6.3	System Architecture: Logical level . . . . .	57
6.4	Communication Infrastructure . . . . .	58



6.5	Example of the master-server node in the Communication Infrastructure . . .	59
6.6	Example of node interconnection and interface layer . . . . .	59
6.7	Message structure of the Application Datagram Protocol. . . . .	62
6.8	Handshake protocol in case of no errors . . . . .	63
6.9	Handshake protocol. Message lost and CRC error use case . . . . .	64
6.10	Application Stream Protocol . . . . .	65
7.1	Task partitioning and mapping steps . . . . .	67
7.2	CDAS architecture for MPSoC-FPGAs . . . . .	68
7.3	Control-Synchronization Module associated to a single slave-client node . .	70
7.4	Control-Synchronization Module: Communication Unit instance . . . . .	70
7.5	Synchronization Unit . . . . .	72
7.6	FSM of the <i>Execute</i> module . . . . .	74
7.7	UML sequence diagram: server modules in the CDAS architecture . . . . .	76
7.8	CDAS: Data-Flow Unit . . . . .	82
7.9	CDAS: Data-Processing Module . . . . .	85
7.10	Matching steps between AXI ID and domain ID . . . . .	86
7.11	Matching steps between a Transaction Memory Address and MR . . . . .	87
7.12	Architecture of the Lightweight Protection Unit . . . . .	88
7.13	Policy check functionality . . . . .	90
7.14	Exemplary of LPU placement . . . . .	90
7.15	Exemplary Policy Configuration and Decision . . . . .	91
8.1	Definition of the KIDS-CT requirements . . . . .	94
8.2	KIDS-CT System Architecture: Physical level . . . . .	95
8.3	KIDS-CT System Architecture: Logical level . . . . .	96
8.4	KIDS-CT: Communication Infrastructure . . . . .	97
8.5	Assembled KIDS-CT scanner . . . . .	98
8.6	Structure of four typical message data sections, used in the KIDS-CT scanner	99
8.7	Acquisition datapath for CT scanners without real-time support . . . . .	99
8.8	Acquisition interval between consecutive projections . . . . .	100
8.9	Image reconstruction: Algorithm steps . . . . .	101
8.10	Optimized datapath for the KIDS-CT scanner . . . . .	101
9.1	CDAS architecture for the KIDS-CT scanner . . . . .	102
9.2	KIDS-CT Synchronization Unit . . . . .	105
9.3	KIDS-CT: Synchronization steps for making CT scans . . . . .	105
9.4	KIDS-CT: Synchronization errors during a CT scan . . . . .	106
9.5	KIDS-CT: Data-Flow Module instantiated in the CDAS . . . . .	108
9.6	KIDS-CT: Data-Processing Module . . . . .	109

9.7	Implementation of the Pixel-processing stage for the KIDS-CT scanner . . .	110
9.8	Standard I0-correction . . . . .	112
9.9	Optimized I0-correction step . . . . .	112
9.10	First approach of the weighting steps . . . . .	113
9.11	Second approach of the weighting steps . . . . .	113
9.12	Third approach of the weighting steps . . . . .	114
9.13	Example of pixels with same weights within a projection . . . . .	114
9.14	Optimization of the weighting steps . . . . .	115
10.1	Design configurations for the LPU validation . . . . .	124
10.2	Validation settings of the LPUs . . . . .	124
10.3	Validation of the Data-Flow Module . . . . .	125
11.1	Resource utilization of the Data-Flow Module . . . . .	131
11.2	Exemplary deployment of LPU . . . . .	137
12.1	I0-image: Image without object . . . . .	140
12.2	CATPHAN® 500 [163] phantom . . . . .	141
12.3	Section of the CTP515 Low-Contrast Module [177] . . . . .	142
12.4	CTP486 Uniformity Module [177] . . . . .	142
12.5	CTP401 Slice Geometry and Sensitometry Module [177] . . . . .	143
12.6	Placements of the ROIs . . . . .	144
12.7	Projection of the CTP401 module, before and after the pixel processing part	145
12.8	Reconstructed image of the CTP401 module for human visual analysis . .	146
12.9	Reconstructed image of the CTP486 module, for noise and uniformity analysis	147
12.10	Reconstructed image of the CTP515 module, for low-contrast analysis . . .	148

## List of Tables

2.1	Detector Geometrical Parameters . . . . .	23
2.2	Projection Geometry Parameters . . . . .	23
2.3	Volume Geometry Parameters . . . . .	24
7.1	Example of AXI ID associated with different PDs . . . . .	87
9.1	Design-time parameters of the Data-Flow Modules for KIDS-CT scanner .	107
9.2	Runtime parameters of the Data-Flow Modules for KIDS-CT scanner . . .	107
11.1	Resource utilization of the CDAS architecture for the KIDS-CT scanner . .	128
11.2	Clock domains CDAS architecture for the KIDS-CT scanner . . . . .	129
11.3	Latency CDAS architecture for the KIDS-CT scanner . . . . .	129
11.4	Resource utilization of the Data-Flow Module . . . . .	130
11.5	Lightweight re-configurable dataflow architecture latency . . . . .	132
11.6	Resource utilization: Data-Processing Module using the standard solution .	134
11.7	Resource utilization: Data-Processing Module using the optimized solution	135
11.8	Latency of the Data-Processing Module . . . . .	136
11.9	Resource utilization for the test design . . . . .	137
11.10	Resources' utilization and latency (c.c. means clock's cycle) . . . . .	138
12.1	KIDS-CT scanner: scanning parameters . . . . .	140
12.2	MSE of the 2D projections . . . . .	146
12.3	MSE of the 3D volumes . . . . .	147
12.4	Image quality analysis . . . . .	149
12.5	Resource utilization and latency used for the DSE . . . . .	150
13.1	Related work comparison . . . . .	156

## List of Acronyms

<b>2D</b>	2-Dimensional
<b>3D</b>	3-Dimensional
<b>A</b>	Analog
<b>ACAP</b>	Adaptive Computing Acceleration Platform
<b>ACK</b>	acknowledge
<b>ADAS</b>	Advanced Driver-Assistance System
<b>ADC</b>	Analog-to-Digital Converter
<b>ADD</b>	Addition
<b>AI</b>	Artificial Intelligence
<b>AIB</b>	AXI isolation blocks
<b>ALU</b>	Arithmetic Logic Unit
<b>AMBA</b>	Advanced Microcontroller Bus Architecture
<b>AP</b>	Access Policy
<b>APB</b>	Advanced Peripheral Bus
<b>APU</b>	Application Processing Unit
<b>ASIC</b>	Application-Specific Integrated Circuit
<b>AXI</b>	Advanced eXtensible Interface
<b>BRAM</b>	Block RAM
<b>CAST</b>	Casting
<b>CDAS</b>	Control-Data Acquisition System
<b>CDC</b>	Clock Domain Crossing
<b>CNR</b>	Contrast Noise Ratio

<b>COS</b>	Cosine
<b>CPS</b>	Cyber-Physical System
<b>CPS PWG</b>	Cyber-Physical Systems Public Working Group
<b>CPU</b>	Central Processing Unit
<b>CRC</b>	Cyclic Redundancy Check
<b>CT</b>	Computed Tomography
<b>CS</b>	Control System
<b>D</b>	Digital
<b>DAS</b>	Data Acquisition System
<b>DCS</b>	Detector Coordinate System
<b>DIV</b>	Division
<b>DMS</b>	Detector Management System
<b>DPS</b>	Data Processing System
<b>DRAM</b>	Dynamic Random Access Memory
<b>DSE</b>	Design Space Exploration
<b>DSP</b>	Digital Signal Processor
<b>ES</b>	Embedded System
<b>EXP</b>	Exponential
<b>F</b>	Fractional
<b>FBP</b>	Filtered Back-Projection
<b>FDK</b>	Feldkamp, Davis, and Kress
<b>FF</b>	Flip-Flop
<b>FFT</b>	Fast Fourier Transform
<b>FIFO</b>	First In First Out
<b>FP16</b>	16-bit floating point

<b>FP32</b>	32-bit floating point
<b>FP64</b>	64-bit floating point
<b>FPGA</b>	Field Programmable Gate Array
<b>FSBL</b>	First Stage Bootloader
<b>FSD</b>	Full-Self Driving
<b>FMC+</b>	FPGA Mezzanine Card
<b>FSM</b>	Finite State Machine
<b>FXP16</b>	16-bit fixed point
<b>FXP32</b>	32-bit fixed point
<b>GCT</b>	Generic Computed Tomography
<b>GFLOPS</b>	Giga FLoating-point Operations Per Second
<b>GPIO</b>	General Purpose Input Output
<b>GPU</b>	Graphics Processing Unit
<b>GTX</b>	Gigabit Transceiver
<b>HDL</b>	Hardware Description Language
<b>HIMM</b>	Hardware IP management module
<b>HLS</b>	High-Level Synthesis
<b>HU</b>	Hounsfield Unit
<b>HV</b>	High Voltage
<b>I</b>	Integer
<b>I/O</b>	Input/Output
<b>I2C</b>	Inter-Integrated Circuit
<b>iCT</b>	interventional Computed Tomography
<b>ILA</b>	Integrated Logic Analyzer
<b>IP</b>	Intellectual Property

<b>J-PET</b>	Jagiellonian-PET
<b>JTAG</b>	Joint Test Action Group
<b>KIDS-CT</b>	Konfigurierbarer, Interfaceoffener, Dosisparender Computertomograph
<b>LDPE</b>	Low-Density PolyEthylene
<b>LOG</b>	Logarithm
<b>LPS</b>	Left, Posterior, Superior
<b>LPU</b>	Lightweight Protection Unit
<b>LSB</b>	Least Significant Bit
<b>LUT</b>	Look-Up Table
<b>M2M</b>	Machine to Machine
<b>MCPS</b>	Medical Cyber-Physical System
<b>MCS</b>	Mixed-Criticality System
<b>MMU</b>	Memory Management Unit
<b>MPPU</b>	Memory Partition Protection Unit
<b>MPSoC</b>	Multi-Processor System-on-Chip
<b>MPSoC-FPGA</b>	Multi-Processor System-on-Chip Field Programmable Gate-Array
<b>MPU</b>	Memory Protection Unit
<b>MR</b>	Memory Region
<b>MRI</b>	Magnetic Resonance Imaging
<b>MSB</b>	Most Significant Bit
<b>MSE</b>	Mean Square Error
<b>MUL</b>	Multiplication
<b>NIST</b>	National Institute of Standards and Technology
<b>NNA</b>	Neural-Network Accelerator
<b>NoC</b>	Network-on-Chip

<b>NoCF</b>	Network-on-Chip Firewall
<b>OCM</b>	On-Chip-Memory
<b>OPC</b>	Open Platform Communications
<b>OPC UA</b>	Open Platform Communications United Architecture
<b>OS</b>	Operating System
<b>OSI</b>	Open Systems Interconnection model
<b>PC</b>	Personal Computer
<b>PCI-E</b>	Peripheral Component Interconnect Express
<b>PCS</b>	Pixel Coordinate System
<b>PD</b>	Protection Domain
<b>PE</b>	Processing Element
<b>PET</b>	Positron Emission Tomography
<b>POW</b>	Power
<b>PL</b>	Programmable Logic
<b>PMU</b>	Platform Management Unit
<b>PS</b>	Processing System
<b>PU</b>	Protection Unit
<b>QoS</b>	Quality of Service
<b>RA</b>	Read Address channel
<b>RAS</b>	Right, Anterior, Superior
<b>RD</b>	Read Data channel
<b>ROI</b>	Region Of Interest
<b>RS</b>	Resconstruction System
<b>RTL</b>	Register-Transfer Level
<b>RTOS</b>	Real-Time Operating System



<b>rx</b>	receiver
<b>SA</b>	System Architecture
<b>SC</b>	Scale Case
<b>SFP+</b>	Small form-factor pluggable
<b>SIMM</b>	Hardware IP management module
<b>SMMU</b>	System Memory Management Unit
<b>SoC</b>	System-on-Chip
<b>SPI</b>	Serial Peripheral Interface
<b>SRAM</b>	Static Random-Access Memory
<b>SUB</b>	Subtraction
<b>tx</b>	transmitter
<b>TPU</b>	Tensor Processing Unit
<b>TRUNC</b>	Truncate
<b>UART</b>	Universal Asynchronous Receiver Transmitter
<b>UDP</b>	User Datagram Protocol
<b>UID</b>	Unique Identifier
<b>UML</b>	Unified Modeling Language
<b>USB</b>	Universal Serial Bus
<b>VCS</b>	Voxel Coordinate System
<b>VHDL</b>	VHSIC Hardware Description Language
<b>VHSIC</b>	Very High Speed Integrated Circuit
<b>VIP</b>	AXI Verification IP
<b>WA</b>	Write Address channel
<b>WCET</b>	Worst-Case Execution Time
<b>WCS</b>	World Coordinate System

**WD** Write Data channel

**WR** Write Response

**XMPU** Xilinx Memory Protection Unit

**XPPU** Xilinx Peripheral Protection Unit

## Bibliography

- [1] S. A. Haque, S. M. Aziz, and M. Rahman. “Review of Cyber-Physical System in Healthcare”. In: *International Journal of Distributed Sensor Networks* 10.4 (2014). Online; Accessed: 23.02.2023, p. 217415. ISSN: 1550-1477. DOI: 10.1155/2014/217415.
- [2] F. Hofer. “Architecture, technologies and challenges for cyber-physical systems in industry 4.0”. In: *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Ed. by M. Oivo, D. Méndez, and A. Mockus. New York, NY, USA: ACM, 10112018, pp. 1–10. ISBN: 9781450358231. DOI: 10.1145/3239235.3239242.
- [3] E. Griffor, C. Greer, D. Wollman, and M. Burns. “Framework for Cyber-Physical Systems: Volume 1, Overview”. In: (June 2017). DOI: 10.6028/NIST.SP.1500-201.
- [4] M. V. Garcia, E. Irisarri, F. Pérez, E. Estévez, and M. Marcos. “OPC-UA communications integration using a CPPS architecture”. In: *2016 IEEE Ecuador technical chapters meeting (ETCM)*. IEEE. 2016, pp. 1–6.
- [5] D. Crosetto. “A modular VME or IBM PC based data acquisition system for multi-modality PET/CT scanners of different sizes and detector types”. In: *2000 IEEE Nuclear Science Symposium. Conference Record (Cat. No.00CH37149)*. Vol. 2. 2000, 12/78–12/97 vol.2. DOI: 10.1109/NSSMIC.2000.949946.
- [6] L. Martí-Bonmatí, R. Sopena, P. Bartumeus, and P. Sopena. “Multimodality imaging techniques”. In: *Contrast media & molecular imaging* 5.4 (2010), pp. 180–189.
- [7] S. A. Wells et al. “Liver ablation: best practice”. In: *Radiologic Clinics* 53.5 (2015), pp. 933–971.
- [8] S. R. Cherry. “Multimodality imaging: Beyond pet/ct and spect/ct”. In: *Seminars in nuclear medicine*. Vol. 39. 5. Elsevier. 2009, pp. 348–353.
- [9] F. C. S. O. von-Guericke University of Magdeburg. *KIDS-CT, Open-Interface-CT*. Online; Accessed: 10.03.2023. 2021. URL: <https://www.forschungscampus-stimulate.de/de/services/open-interface-ct/index.html>.
- [10] E. Alcaín et al. “Hardware Architectures for Real-Time Medical Imaging”. In: *Electronics* 10.24 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10243118. URL: <https://www.mdpi.com/2079-9292/10/24/3118>.

- [11] J. Hsieh. *Computed Tomography: Principles, Design, Artifacts, and Recent Advances*. 4th ed. Vol. PM344. SPIE, 2022. ISBN: 9781510646872. URL: <https://spie.org/Publications/Book/2605932>.
- [12] X. Han, C. Wang, et al. *Airway Stenting in Interventional Radiology*. Springer, 2019.
- [13] A. K. Jones, R. G. Dixon, J. D. Collins, E. M. Walser, B. Nikolic, et al. “Best practice guidelines for CT-guided interventional procedures”. In: *J Vasc Interv Radiol* 29.04 (2018), pp. 518–519.
- [14] P. Babu and E. Parthasarathy. “Reconfigurable FPGA architectures: A survey and applications”. In: *Journal of The Institution of Engineers (India): Series B* 102 (2021), pp. 143–156.
- [15] S. Gandhare and B. Karthikeyan. “Survey on FPGA Architecture and Recent Applications”. In: *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*. 2019, pp. 1–4. DOI: 10.1109/ViTECoN.2019.8899550.
- [16] W.-T. Zhang et al. “Design of heterogeneous MPSoC on FPGA”. In: *2007 7th International Conference on ASIC*. 2007, pp. 102–105. DOI: 10.1109/ICASIC.2007.4415577.
- [17] P. Marwedel. *Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things*. Springer Nature, 2021.
- [18] E. A. Lee. “Computing foundations and practice for cyber-physical systems: A preliminary report”. In: *University of California, Berkeley, Tech. Rep. UCB/EECS-2007-72* 21 (2007).
- [19] S. Heath. *Embedded systems design*. Elsevier, 2002.
- [20] W. Shi and S. Dustdar. “The promise of edge computing”. In: *Computer* 49.5 (2016), pp. 78–81.
- [21] S. Heinrich and L. Motors. “Flash memory in the emerging age of autonomy”. In: *Flash Memory Summit* (2017), pp. 1–10.
- [22] D. Philips. *Detector CD300 with 64 rows*. Online; Accessed: 22.03.2023. 2018. URL: <https://www.dunlee.com/a-w/ct-solutions/ct-detectors/64-row-detector.html>.
- [23] A. G. Mariño, F. Fons, and J. M. M. Arostegui. “The future roadmap of in-vehicle network processing: A HW-centric (R-) evolution”. In: *IEEE access* 10 (2022), pp. 69223–69249.

- [24] S. Jogwar and P. Daoutidis. “Community-based synthesis of distributed control architectures for integrated process networks”. English (US). In: *Chemical Engineering Science* 172 (2017). Publisher Copyright: © 2017 Elsevier Ltd Copyright: Copyright 2017 Elsevier B.V., All rights reserved., pp. 434–443. ISSN: 0009-2509. DOI: 10.1016/j.ces.2017.06.043.
- [25] S.-H. Tseng and J. Anderson. “Synthesis to deployment: Cyber-physical control architectures”. In: *arXiv preprint arXiv:2012.05211* (2020).
- [26] H. Stähle, L. Mercep, A. Knoll, and G. Spiegelberg. “Towards the deployment of a centralized ict architecture in the automotive domain”. In: *2013 2nd Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2013, pp. 66–69.
- [27] R. V. Chakaravarthy, H. Kwon, and H. Jiang. “Vision Control Unit in Fully Self Driving Vehicles Using Xilinx MPSoC and Opensource Stack”. In: *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. ASPDAC '21. Tokyo, Japan: Association for Computing Machinery, 2021, 311–317. ISBN: 9781450379991. DOI: 10.1145/3394885.3431616. URL: <https://doi.org/10.1145/3394885.3431616>.
- [28] E. Talpes et al. “Compute Solution for Tesla’s Full Self-Driving Computer”. In: *IEEE Micro* 40.2 (2020), pp. 25–35. DOI: 10.1109/MM.2020.2975764.
- [29] W. Wolf, A. A. Jerraya, and G. Martin. “Multiprocessor system-on-chip (MPSoC) technology”. In: *IEEE transactions on computer-aided design of integrated circuits and systems* 27.10 (2008), pp. 1701–1713.
- [30] A. Biondi et al. “SPHERE: A multi-SoC architecture for next-generation cyber-physical systems based on heterogeneous platforms”. In: *IEEE Access* 9 (2021), pp. 75446–75459.
- [31] AMD-Xilinx. *XA Zynq-7000 SoC Data Sheet: Overview (DS188)*. [Online; Accessed: 22.06.2023]. 2018. URL: <https://docs.xilinx.com/v/u/en-US/ds188-XA-Zynq-7000-Overview>.
- [32] A. A. Prince and V. Kartha. “A framework for remote and adaptive partial re-configuration of SoC based data acquisition systems under Linux”. In: *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2015, pp. 1–5.
- [33] J. L. Ayala. *Communication Architectures for Systems-on-chip*. CRC Press, 2018.
- [34] AMD-Xilinx. *Learn the architecture - An introduction to AMBA AXI*. <https://developer.arm.com/documentation/102202/0300/What-is-AMBA--and-why-use-it->. Online; Accessed: 24.07.2023. 2023.

- [35] AMD-Xilinx. *AXI Reference Guide (UG761)*. [https://docs.xilinx.com/v/u/en-US/ug761\\_axi\\_reference\\_guide](https://docs.xilinx.com/v/u/en-US/ug761_axi_reference_guide). [Online; Accessed: 10-July-2023]. 2012.
- [36] “AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite”. In: *Arm* (30.03.2013). Online; Accessed: 20.07.2021. URL: <https://developer.arm.com/documentation/ih0022/e/>.
- [37] A. Burns and R. Davis. “Mixed criticality systems-a review”. In: *Department of Computer Science, University of York, Tech. Rep* (2013), pp. 1–69.
- [38] E. Dubrova. “Fundamentals of Dependability”. In: *Fault-Tolerant Design*. New York, NY: Springer New York, 2013, pp. 5–20. ISBN: 978-1-4614-2113-9. DOI: 10.1007/978-1-4614-2113-9\_2. URL: [https://doi.org/10.1007/978-1-4614-2113-9\\_2](https://doi.org/10.1007/978-1-4614-2113-9_2).
- [39] D. Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K. S. Trivedi. “Fault triggers in open-source software: An experience report”. In: *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 2013, pp. 178–187. DOI: 10.1109/ISSRE.2013.6698917.
- [40] L. Piardi, P. Leitão, and A. S. de Oliveira. “Fault-tolerance in cyber-physical systems: Literature review and challenges”. In: *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*. Vol. 1. IEEE. 2020, pp. 29–34.
- [41] R. Wilhelm et al. “The worst-case execution-time problem—overview of methods and survey of tools”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 7.3 (2008), pp. 1–53.
- [42] S. Altmeyer, B. Lisper, C. Maiza, J. Reineke, and C. Rochange. “WCET and mixed-criticality: What does confidence in WCET estimations depend upon?” In: *15th International Workshop on Worst-Case Execution Time Analysis (WCET 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2015.
- [43] A. Abel et al. “Impact of resource sharing on performance and performance prediction: A survey”. In: *CONCUR 2013—Concurrency Theory: 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings 24*. Springer. 2013, pp. 25–43.
- [44] R. Schneider, D. Goswami, A. Masrur, M. Becker, and S. Chakraborty. “Multi-layered scheduling of mixed-criticality cyber-physical systems”. In: *Journal of Systems Architecture* 59.10 (2013), pp. 1215–1230.
- [45] M. Bottaro and T. Vardanega. “Evaluating a multicore Mixed-Criticality System implementation against a temporal isolation kernel”. In: *Journal of Systems Architecture* 130 (2022), p. 102688. ISSN: 1383-7621. DOI: <https://doi.org/10.1016/j.sysarc.2022.102688>. URL: <https://www.sciencedirect.com/science/article/pii/S1383762122001916>.

- 
- [46] ARM. “Cortex-M4 Technical Reference Manual r0p0”. In: (2009). Online; Accessed: 19.01.2024. URL: <https://developer.arm.com/documentation/ddi0439/b/Memory-Protection-Unit>.
- [47] FreeRTOS. *FreeRTOS-MPU - ARM Cortex-M3 and ARM Cortex-M4 Memory Protection Unit support in FreeRTOS*. Online; Accessed: 19.01.2024. 2023. URL: <https://www.freertos.org/FreeRTOS-MPU-memory-protection-unit.html>.
- [48] Xilinx and Inc. *Zynq UltraScale+ Device Technical Reference Manual*. Online; Accessed: 11.06.2021. URL: [https://www.xilinx.com/support/documentation/user\\_guides/ug585-Zynq-7000-TRM.pdf](https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf).
- [49] G. Heiser. “The role of virtualization in embedded systems”. In: *Proceedings of the 1st workshop on Isolation and integration in embedded systems*. Ed. by M. Engel. New York, NY: ACM, 2008, pp. 11–16. ISBN: 9781605581262. DOI: 10.1145/1435458.1435461.
- [50] B. Bapst, M. Lagadec, R. Breguet, V. Vilgrain, and M. Ronot. “Cone beam computed tomography (CBCT) in the field of interventional oncology of the liver”. In: *Cardiovascular and interventional radiology* 39 (2016), pp. 8–20.
- [51] I. Wisely. *Iterative Reconstruction in CT*. <https://www.imagewisely.org/Imaging-Modalities/Computed-Tomography/Iterative-Reconstruction-in-CT>. Online; Accessed: 16.01.2024. 2023.
- [52] T.-Y. Lee and R. K. Chhem. “Impact of new technologies on dose reduction in CT”. In: *European journal of radiology* 76.1 (2010), pp. 28–35.
- [53] L. L. Geyer et al. “State of the art: iterative CT reconstruction techniques”. In: *Radiology* 276.2 (2015), pp. 339–357.
- [54] H. W. Goo and J. M. Goo. “Dual-energy CT: new horizon in medical imaging”. In: *Korean journal of radiology* 18.4 (2017), pp. 555–569.
- [55] H. Scherl. *Evaluation of State-of-the-Art Hardware Architectures for Fast Cone-Beam CT Reconstruction*. English. 1st ed. 2011. Aktuelle Forschung Medizintechnik – Latest Research in Medical Engineering. Vieweg+Teubner Verlag, 2011. ISBN: 9783834882592. URL: <https://doi.org/10.1007/978-3-8348-8259-2?nosfx=y>.
- [56] Philips. *Philips iCT platform*. Online; Accessed: 22.03.2023. 2021. URL: [https://www.usa.philips.com/healthcare/resources/landing/azurion?npagination=1&\\_ga=2.132416206.1786482620.1679490727-592968462.1679490720](https://www.usa.philips.com/healthcare/resources/landing/azurion?npagination=1&_ga=2.132416206.1786482620.1679490727-592968462.1679490720).
- [57] Siemens. *Siemens iCT platform*. Online; Accessed: 22.03.2023. 2021. URL: <https://www.siemens-healthineers.com/computed-tomography/#somatom-x-platform>.

- 
- [58] Siemens. *Miyabi Angio CT*. <https://www.siemens-healthineers.com/es/angio/artis-interventional-angiography-systems/miyabi>. [Online; Accessed: 22.06.2023]. 2023.
- [59] D. Philips. *Dunlee Xpert bundle with CT6500*. <https://www.dunlee.com/a-w/ct-solutions/ct-product-bundles/ct6500.html>. [Online; Accessed: 22.03.2023]. 2018.
- [60] L. Faggioni, F. Paolicchi, and E. Neri. *Elementi di tomografia computerizzata*. Vol. 4. Springer Science & Business Media, 2011.
- [61] S. GmbH. *Schleifring CT Gantry*. Online; Accessed: 13.03.2023. 2018. URL: [https://www.schleifring.de/fileadmin/08\\_Downloads/CT-Applications\\_January18.pdf](https://www.schleifring.de/fileadmin/08_Downloads/CT-Applications_January18.pdf).
- [62] M. J. Willeminck and P. B. Noël. “The evolution of image reconstruction for CT—from filtered back projection to artificial intelligence”. In: *European radiology* 29 (2019), pp. 2185–2195.
- [63] L. A. Feldkamp, L. C. Davis, and J. W. Kress. “Practical cone-beam algorithm”. In: *Josa a* 1.6 (1984), pp. 612–619.
- [64] H. Scherl, M. Kowarschik, H. G. Hofmann, B. Keck, and J. Hornegger. “Evaluation of state-of-the-art hardware architectures for fast cone-beam CT reconstruction”. In: *Parallel computing* 38.3 (2012), pp. 111–124.
- [65] F. Noo, J. Pack, and D. Heuscher. “Exact helical reconstruction using native cone-beam geometries”. In: *Physics in Medicine & Biology* 48.23 (2003), p. 3787.
- [66] D. L. Parker. “Optimal short scan convolution reconstruction for fan beam CT”. In: *Medical physics* 9.2 (1982), pp. 254–257.
- [67] E. A. Lee. “Cyber physical systems: Design challenges”. In: *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*. IEEE. 2008, pp. 363–369.
- [68] D. Windisch, O. Knodel, G. Juckeland, U. Hampel, and A. Bieberle. “FPGA-Based Real-Time Data Acquisition for Ultrafast X-Ray Computed Tomography”. In: *IEEE Transactions on Nuclear Science* 68.12 (2021), pp. 2779–2786. DOI: 10.1109/TNS.2021.3123837.
- [69] A. D. Pimentel. “Exploring exploration: A tutorial introduction to embedded systems design space exploration”. In: *IEEE Design & Test* 34.1 (2016), pp. 77–90.
- [70] M. Gries. “Methods for Evaluating and Covering the Design Space during Early Design Development”. In: *Integr. VLSI J.* 38.2 (2004), 131–183. ISSN: 0167-9260. DOI: 10.1016/j.vlsi.2004.06.001. URL: <https://doi.org/10.1016/j.vlsi.2004.06.001>.



- [71] M. Gries. “Methods for evaluating and covering the design space during early design development”. In: *Integration* 38.2 (2004), pp. 131–183. ISSN: 0167-9260. DOI: <https://doi.org/10.1016/j.vlsi.2004.06.001>. URL: <https://www.sciencedirect.com/science/article/pii/S016792600400032X>.
- [72] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. “Mapping on multi/many-core systems: Survey of current and emerging trends”. In: *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2013, pp. 1–10. DOI: 10.1145/2463209.2488734.
- [73] “IEEE Standard for Floating-Point Arithmetic”. In: *IEEE Std 754-2008* (2008), pp. 1–70. DOI: 10.1109/IEEESTD.2008.4610935.
- [74] D. L. N. Hettiarachchi, V. S. P. Davuluru, and E. J. Balster. “Integer vs. Floating-Point Processing on Modern FPGA Technology”. In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 2020, pp. 0606–0612. DOI: 10.1109/CCWC47524.2020.9031118.
- [75] A. R. Omondi. *Computer arithmetic systems: algorithms, architecture and implementation*. Prentice Hall International (UK) Ltd., 1994.
- [76] C. Kormanyos. “Fixed-Point Mathematics”. In: *Real-Time C++: Efficient Object-Oriented and Template Microcontroller Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 267–289. ISBN: 978-3-662-56718-0. DOI: 10.1007/978-3-662-56718-0\_13. URL: [https://doi.org/10.1007/978-3-662-56718-0\\_13](https://doi.org/10.1007/978-3-662-56718-0_13).
- [77] AMD-Xilinx. *Getting Started with Vitis HLS*. Online; Accessed: 07.12.2023. 2021. URL: <https://docs.xilinx.com/r/2021.2-English/ug1399-vitis-hls/Getting-Started-with-Vitis-HLS>.
- [78] A. Ahmadi, C. Cherifi, V. Cheutet, and Y. Ouzrout. “A review of CPS 5 components architecture for manufacturing based on standards”. In: *2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*. 2017, pp. 1–6. DOI: 10.1109/SKIMA.2017.8294091.
- [79] J. Jamaludin and J. M. Rohani. “Cyber-physical system (cps): State of the art”. In: *2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*. IEEE. 2018, pp. 1–5.
- [80] L. Hu, N. Xie, Z. Kuang, and K. Zhao. “Review of Cyber-Physical System Architecture”. In: *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. 2012, pp. 25–30. DOI: 10.1109/ISORCW.2012.15.
- [81] S. H. Ahmed, G. Kim, and D. Kim. “Cyber Physical System: Architecture, applications and research challenges”. In: *2013 IFIP Wireless Days (WD)*. 2013, pp. 1–5. DOI: 10.1109/WD.2013.6686528.

- [82] E. Min et al. “Development of Compact, Cost-effective, FPGA-Based Data Acquisition System for the iPET System”. In: *Journal of Medical and Biological Engineering* 37.6 (2017), pp. 858–866. DOI: 10.1007/s40846-017-0245-1.
- [83] G. Korcyl et al. “Trigger-less and reconfigurable data acquisition system for positron emission tomography”. In: *Bio-Algorithms and Med-Systems* 10.1 (2014). DOI: 10.1515/bams-2013-0115.
- [84] G. Korcyl et al. “Evaluation of single-chip, real-time tomographic data processing on FPGA SoC devices”. In: *IEEE transactions on medical imaging* 37.11 (2018), pp. 2526–2535.
- [85] E Fysikopoulos, G Loudos, M Georgiou, S David, and G Matsopoulos. “A Spartan 6 FPGA-based data acquisition system for dedicated imagers in nuclear medicine”. In: *Measurement Science and Technology* 23.12 (2012), p. 125403. DOI: 10.1088/0957-0233/23/12/125403.
- [86] M Traxler et al. “A compact system for high precision time measurements (14 ps RMS) and integrated data acquisition for a large number of channels”. In: *Journal of Instrumentation* 6.12 (2011), p. C12004. DOI: 10.1088/1748-0221/6/12/C12004. URL: <https://dx.doi.org/10.1088/1748-0221/6/12/C12004>.
- [87] *Medical Device "Plug-and-Play" (MD PnP) Interoperability Program*. <https://mdpnp.org>. Online; Accessed: 17.01.2022.
- [88] R. M. Hofmann. “Modeling medical devices for plug-and-play interoperability”. PhD thesis. Massachusetts Institute of Technology, 2007.
- [89] T. Li et al. “From Offline toward Real-Time: A Hybrid Systems Model Checking and CPS Co-design Approach for Medical Device Plug-and-Play (MDPnP)”. In: *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*. 2012, pp. 13–22. DOI: 10.1109/ICCPS.2012.10.
- [90] *About OPC Technologies: OPC UA*. Online; Accessed: 30.11.2023. OPC Foundation. URL: <https://opcfoundation.org/about/opc-technologies/opc-ua/>.
- [91] J. Kim and J. Jeong. “Design and implementation of opc ua-based vr/ar collaboration model using cps server for vr engineering process”. In: *Applied Sciences* 12.15 (2022), p. 7534.
- [92] V. Jirkovský, P. Kadera, and M. Obitko. “OPC UA realization of cloud cyber-physical system”. In: *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. IEEE. 2018, pp. 115–120.

- [93] A. Brusaferrri, A. Ballarino, F. A. Cavadini, D. Manzocchi, and M. Mazzolini. “CPS-based hierarchical and self-similar automation architecture for the control and verification of reconfigurable manufacturing systems”. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE. 2014, pp. 1–8.
- [94] M. Graube, S. Hensel, C. Iatrou, and L. Urbas. “Information models in OPC UA and their advantages and disadvantages”. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2017, pp. 1–8.
- [95] Y. Liu, Y. Peng, B. Wang, S. Yao, and Z. Liu. “Review on cyber-physical systems”. In: *IEEE/CAA Journal of Automatica Sinica* 4.1 (2017), pp. 27–40. DOI: 10.1109/jas.2017.7510349.
- [96] J. Marjanovic et al. “A reconfigurable platform for magnetic resonance data acquisition and processing”. In: *IEEE Transactions on Medical Imaging* 39.4 (2019), pp. 1138–1148.
- [97] C. Yang and M. Chen. “Design of High Performance Image Acquisition and Processing Platform Based on DSP and FPGA”. In: *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications( AEECA)*. 2020, pp. 684–688. DOI: 10.1109/AEECA49918.2020.9213543.
- [98] H. Shi and S. Zhang. “Dual-Channel Image Acquisition System Based on FPGA”. In: *2019 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)*. 2019, pp. 421–424. DOI: 10.1109/ICITBS.2019.00110.
- [99] Y. Xie et al. “FPGA Implementation of High-Speed Data Acquisition System for High-Resolution Millimeter Wave Radar”. In: *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCASST)*. 2020, pp. 1–4. DOI: 10.1109/MOCASST49295.2020.9200252.
- [100] C. Flouzat, E. Piriou, M. Guibert, B. Jovanović, and M. Oussayran. “EVPS: an automotive video acquisition and processing platform”. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 1716–1717.
- [101] S Salgaro et al. “Plug-and-Play High-Speed Communication Protocol for Readout-Systems Network Based on FPGA and Gigabit Optical Fiber Network”. In: *2020 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*. IEEE. 2020, pp. 1–4.
- [102] ELTEC. *PCI Express Frame Grabber for Camera Link Cameras ELTEC systems*. (access:20/01/2020). URL: "[https://www.eltec.de/pdf/datenblaetter/Datasheet\\\_PC\\\_EYE\\\_CL.pdf](https://www.eltec.de/pdf/datenblaetter/Datasheet\_PC\_EYE\_CL.pdf)".

- [103] I. LogiCORE. “LogiCORE IP Aurora 8B/10B v6. 2, User Guide 766”. In: *San Jose, CA, USA: Xilinx* (2011).
- [104] Xilinx Inc. *MicroBlaze Processor Reference Guide*. Online; Accessed: 20.07.2021. URL: [https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2018\\_2/ug984-vivado-microblaze-ref.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug984-vivado-microblaze-ref.pdf).
- [105] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin. “TrustZone Explained: Architectural Features and Use Cases”. In: *2016 IEEE 2nd International Conference on Collaboration and Internet Computing*. Piscataway, NJ: IEEE, 2016, pp. 445–451. ISBN: 978-1-5090-4607-2. DOI: 10.1109/CIC.2016.065.
- [106] Xilinx, Inc. *Isolation Methods in Zynq UltraScale+ MPSoCs Application Note*. Online; Accessed: 02.03.2021.
- [107] J. Coburn, S. Ravi, A. Raghunathan, and S. Chakradhar. “SECA”. In: *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems - CASES '05*. Ed. by T. M. Conte, P. Faraboschi, B. Mangione-Smith, and W. Najjar. New York, New York, USA: ACM Press, 2005, p. 78. ISBN: 159593149X. DOI: 10.1145/1086297.1086308.
- [108] L. Fiorin, G. Palermo, S. Lukovic, and C. Silvano. “A data protection unit for NoC-based architectures”. In: *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis - CODES+ISSS '07*. Ed. by S. Ha, K. Choi, N. Dutt, and J. Teich. New York, New York, USA: ACM Press, 2007, p. 167. ISBN: 9781595938244. DOI: 10.1145/1289816.1289858.
- [109] P. Cotret, G. Gogniat, and M. J. Sepúlveda Flórez. “Protection of heterogeneous architectures on FPGAs: An approach based on hardware firewalls”. In: *Microprocessors and Microsystems* 42 (2016), pp. 127–141. ISSN: 01419331. DOI: 10.1016/j.micpro.2016.01.013.
- [110] F. Siddiqui, M. Hagan, and S. Sezer. “Pro-Active Policing and Policy Enforcement Architecture for Securing MPSoCs”. In: *2018 31st IEEE International System-on-Chip Conference (SOCC)*. IEEE, 9/4/2018 - 9/7/2018, pp. 140–145. ISBN: 978-1-5386-1491-4. DOI: 10.1109/SOCC.2018.8618531.
- [111] A. Sensaoui, D. Hely, and O.-E.-K. Aktouf. “Toubkal: A Flexible and Efficient Hardware Isolation Module for Secure Lightweight Devices”. In: *2019 15th European Dependable Computing Conference (EDCC)*. IEEE, 9/17/2019 - 9/20/2019, pp. 31–38. ISBN: 978-1-7281-3929-6. DOI: 10.1109/EDCC.2019.00018.
- [112] M. LeMay and C. A. Gunter. “Network-on-Chip Firewall: Countering Defective and Malicious System-on-Chip Hardware”. In: *Logic, Rewriting, and Concurrency*. Ed. by N. Martí-Oliet, P. C. Ölveczky, and C. Talcott. Vol. 9200. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 404–426.

- ISBN: 978-3-319-23164-8. DOI: 10.1007/978-3-319-23165-5{\textunderscore}19.
- [113] G. Kornaros et al. “Hardware Support for Cost-Effective System-Level Protection in Multi-core SoCs”. In: *2015 Euromicro Conference on Digital System Design*. IEEE, 8/26/2015 - 8/28/2015, pp. 41–48. ISBN: 978-1-4673-8035-5. DOI: 10.1109/DSD.2015.65.
- [114] S. Kumar Saha and C. Bobda. “FPGA Accelerated Embedded System Security Through Hardware Isolation”. In: *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 12/15/2020 - 12/17/2020, pp. 1–6. ISBN: 978-1-7281-8952-9. DOI: 10.1109/AsianHOST51057.2020.9358258.
- [115] I. Polian, F. Regazzoni, and J. Sepulveda. “Introduction to hardware-oriented security for MPSoCs”. In: *2017 30th IEEE International System-on-Chip Conference (SOCC)*. IEEE, 9/5/2017 - 9/8/2017, pp. 102–107. ISBN: 978-1-5386-4034-0. DOI: 10.1109/SOCC.2017.8226017.
- [116] T. Dörr, T. Sandmann, and J. Becker. “A Formal Model for the Automatic Configuration of Access Protection Units in MPSoC-Based Embedded Systems”. In: *2020 23rd Euromicro Conference on Digital System Design (DSD)*. 2020, pp. 596–603. DOI: 10.1109/DSD51259.2020.00098.
- [117] Xilinx, Inc. *Xilinx Reduces Risk and Increases Efficiency for IEC61508 and ISO26262 Certified Safety Applications (WP461)*. Online; Accessed: 08.03.2021.
- [118] Xilinx and Inc. *Zynq-7000 SoC Technical Reference Manual*. Online; Accessed: 11.06.2021. URL: [https://www.xilinx.com/support/documentation/user\\_guides/ug585-Zynq-7000-TRM.pdf](https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf).
- [119] Xilinx, Inc. “AXI Interconnect v2.1 LogiCORE IP Product Guide (PG059)”. In: ().
- [120] Intel. *Cyclone V Hard Processor System Technical Reference Manual*.
- [121] Xilinx, Inc. *Isolate Security-Critical Applications on Zynq UltraScale+ Devices White Paper*. Online; Accessed: 02.03.2021.
- [122] Intel-Altera. *Diagnostic Imaging, FPGA Diagnostic Imaging Applications - Intel FPGA*. Online; Accessed: 27.11.2021. URL: <https://www.intel.fr/content/www/fr/fr/healthcare-it/products/programmable/applications/diagnostic-imaging.html>.
- [123] Intel Corporation. *Cyclone V Hard Processor System Technical Reference Manual*. Online; Accessed: 17.11.2023. Aug. 2023. URL: <https://www.intel.com/content/www/us/en/docs/programmable/683126/21-2/introduction-to-the-hard-processor-system-98309.html>.

- [124] Intel Corporation. *Arria V Hard Processor System Technical Reference Manual*. Online; Accessed: 27.11.2023. Dec. 2022. URL: <https://www.intel.com/content/www/us/en/docs/programmable/683011/21-2/introduction-to-the-hard-processor-system-98309.html>.
- [125] D. Lewis et al. “The Stratix™ 10 highly pipelined FPGA architecture”. In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2016, pp. 159–168.
- [126] M. Parker. “Understanding peak floating-point performance claims”. In: *Technical White Paper WP-012220-1.0* (2014).
- [127] Intel-Altera. *Medical Imaging Implementation Using FPGAs - White Paper (WP-MEDICAL-2.0)*. Online; Accessed: 27.11.2023. URL: <https://cdrdv2-public.intel.com/650313/wp-medical.pdf>.
- [128] *Medical Imaging with CT Scanners and MRI Machines*. <https://www.xilinx.com/applications/medical/medical-imaging-ct-mri-pet.html>, [Online; Accessed: 28.03.2021].
- [129] AMD-Xilinx. *Scale+ MPSoC Data Sheet: Overview, DS891, Rev. 1.9*. 26/05/2021.
- [130] AMD-Xilinx. *Versal: The First Adaptive Compute Acceleration Platform (ACAP)*. White Paper WP505. Online; Accessed: 19.12.2023. Xilinx, 2020. URL: <https://docs.xilinx.com/v/u/en-US/wp505-versal-acap>.
- [131] S. M. Savaresi. “The role of real-time communication for distributed or centralized architectures in vehicle dynamics control systems”. In: *2006 IEEE International Workshop on Factory Communication Systems*. IEEE. 2006, pp. 67–72.
- [132] S. Coric, M. Leeser, E. Miller, and M. Trepanier. “Parallel-beam backprojection: an FPGA implementation optimized for medical imaging”. In: *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. 2002, pp. 217–226.
- [133] I. Goddard and M. Trepanier. “High-speed cone-beam reconstruction: an embedded systems approach”. In: *Medical Imaging 2002: Visualization, Image-Guided Procedures, and Display*. Vol. 4681. SPIE. 2002, pp. 483–491.
- [134] B. Heigl and M. Kowarschik. “High-speed reconstruction for C-arm computed tomography”. In: *Proceedings of the 9th international meeting on fully three-dimensional image reconstruction in radiology and nuclear medicine*. 2007, pp. 25–28.
- [135] J. Deng, B. Yan, J. Li, and L. Li. “Parallel no-waiting pipelining accelerating CT image reconstruction based on FPGA”. In: *2010 3rd International Conference on Biomedical Engineering and Informatics*. Vol. 1. IEEE. 2010, pp. 451–455.

- [136] L. Qiao, G. Luo, W. Zhang, and M. Jiang. “FPGA Acceleration of Ray-Based Iterative Algorithm for 3D Low-Dose CT Reconstruction”. In: *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE. 2020, pp. 98–102.
- [137] Y.-k. Choi and J. Cong. “Acceleration of EM-based 3D CT reconstruction using FPGA”. In: *IEEE transactions on biomedical circuits and systems* 10.3 (2015), pp. 754–767.
- [138] D. Windisch, O. Knodel, G. Juckeland, U. Hampel, and A. Bieberle. “FPGA-based Real-Time Data Acquisition for Ultrafast X-Ray Computed Tomography”. In: *IEEE Transactions on Nuclear Science* 68.12 (2021), pp. 2779–2786.
- [139] B. Wang, L. Zhu, K. Jia, and J. Zheng. “Accelerated cone beam CT reconstruction based on OpenCL”. In: *2010 International Conference on Image Analysis and Signal Processing*. IEEE. 2010, pp. 291–295.
- [140] J. Chen et al. “A hybrid architecture for compressive sensing 3-D CT reconstruction”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 2.3 (2012), pp. 616–625.
- [141] H. Scherl, B. Keck, M. Kowarschik, and J. Hornegger. “Fast GPU-based CT reconstruction using the common unified device architecture (CUDA)”. In: *2007 IEEE Nuclear science symposium conference record*. Vol. 6. IEEE. 2007, pp. 4464–4466.
- [142] X. Zhao et al. “GPU-based 3D cone-beam CT image reconstruction: application to micro CT”. In: *2007 Ieee Nuclear Science Symposium Conference Record*. Vol. 5. IEEE. 2007, pp. 3922–3925.
- [143] B. Shi, S. Chen, F. Huang, C. Wang, and K. Bi. “The parallel processing based on CUDA for convolution filter FDK reconstruction of CT”. In: *2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming*. IEEE. 2010, pp. 149–153.
- [144] O. Dandekar, C. Castro-Pareja, and R. Shekhar. “FPGA-based real-time 3D image preprocessing for image-guided medical interventions”. In: *Journal of Real-Time Image Processing* 1.4 (2007), pp. 285–301.
- [145] M. Nourazar and B. Goossens. “Accelerating iterative CT reconstruction algorithms using Tensor Cores”. In: *Journal of Real-Time Image Processing* 18.6 (2021), pp. 1979–1991.
- [146] R. A. Carrasco Cavieres, R. Vega, and C. A. Navarro. *Analyzing GPU Tensor Core Potential for Fast Reductions*. Oct. 2018. DOI: 10.29007/zlmg. URL: <http://dx.doi.org/10.29007/zlmg>.

- [147] C. Maaß, M. Baer, and M. Kachelrieß. “CT image reconstruction with half precision floating-point values”. In: *Medical physics* 38.S1 (2011), S95–S105.
- [148] I. Dumitrache, I. S. Sacala, M. A. Moisesescu, and S. I. Caramihai. “A conceptual framework for modeling and design of Cyber-Physical Systems”. In: *Studies in Informatics and Control* 26.3 (2017), pp. 325–334.
- [149] W. Mahnke, S.-H. Leitner, and M. Damm. *OPC unified architecture*. Springer Science & Business Media, 2009.
- [150] AMD-Xilinx. *Z7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)*. 14/08/2018.
- [151] V. Q. Rodriguez and F. Guillemin. “Performance analysis of VNFs for sizing cloud-RAN infrastructures”. In: *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2017, pp. 1–6. DOI: 10.1109/NFV-SDN.2017.8169835.
- [152] R. Budruk, D. Anderson, and T. Shanley. *PCI express system architecture*. Addison-Wesley Professional, 2004.
- [153] B. McLaughlin. *Building Java Enterprise Applications: Architecture*. Vol. 1. " O’Reilly Media, Inc.", 2002.
- [154] S. T. Albin. *The art of software architecture: design methods and techniques*. Vol. 9. John Wiley & Sons, 2003.
- [155] A. Kurth et al. “An Open-Source Platform for High-Performance Non-Coherent On-Chip Communication”. In: *IEEE Transactions on Computers* 71.8 (2022), pp. 1794–1809. DOI: 10.1109/TC.2021.3107726.
- [156] Xilinx. *ZC706 Evaluation Board for the Zynq-7000 XC7Z045 SoC*. Tech. rep. UG954, v1.8. Aug. 2019.
- [157] T. Feist. “Vivado design suite”. In: *White Paper* 5 (2012), p. 30.
- [158] I. Xilinx. “7 Series DSP48E1 Slice User Guide”. In: *UG479 (v1. 10) March 27* (2018).
- [159] A. Orailoglu and D. D. Gajski. “Flow graph representation”. In: *Proceedings of the 23rd ACM/IEEE Design Automation Conference*. 1986, pp. 503–509.
- [160] T. Fomby. “Scoring measures for prediction problems”. In: *Department of Economics, Southern Methodist University, Dallas, TX* (2008).
- [161] *Measures of image quality*. [https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL\\_COPIES/VELDHUIZEN/node18.html](https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node18.html).
- [162] Xilinx. *7 Series FPGAs Configurable Logic Block, UG474 (v1.8)*. [Online; Accessed: 27.12.2021]. 2016. URL: [https://www.xilinx.com/support/documentation/user\\_guides/ug474\\_7Series\\_CLB.pdf](https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf).



- [163] T. P. Laboratory. *Catphan 500*. <https://www.phantomlab.com/catphan-500>. Online; Accessed: 23.01.2022.
- [164] AMD-Xilinx. *XILINX 7 SERIES FPGAS*. [Online; Accessed: 22.12.2023]. 2018. URL: [https://www.xilinx.com/publications/prod\\_mktg/7-Series-Product-Brief.pdf](https://www.xilinx.com/publications/prod_mktg/7-Series-Product-Brief.pdf).
- [165] AMD-Xilinx. *Vivado Design Suite User Guide: Designing with IP (UG896)*. [Online; Accessed: 22.12.2023]. 2021. URL: <https://docs.xilinx.com/r/2021.1-English/ug896-vivado-ip/IP-Centric-Design-Flow>.
- [166] Xilinx. *Vivado System-Level Design Flows*. Online; Accessed: 07.12.2023. 2021. URL: <https://docs.xilinx.com/r/2021.2-English/ug892-vivado-design-flows-overview/Vivado-System-Level-Design-Flows>.
- [167] AMD-Xilinx. *Getting Started with Vitis*. Online; Accessed: 07.12.2023. 2021. URL: <https://docs.xilinx.com/r/2021.2-English/ug1400-vitis-embedded/Getting-Started-with-Vitis>.
- [168] Xilinx. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906)*. Online; Accessed: 07.12.2023. 2021. URL: <https://docs.xilinx.com/r/2021.1-English/ug906-vivado-design-analysis/Navigating-Content-by-Design-Process>.
- [169] Xilinx. *Integrated Logic Analyzer v2.0 Data Sheet (DS875)*. Online; Accessed: 07.12.2023. 2012. URL: <https://docs.xilinx.com/v/u/en-US/ds875-ila>.
- [170] *DS-520AN 802.11n Wireless & Gigabit Ethernet USB Device Server Including 802.1X Enterprise Security*. Product brochure. Silex Technology. 2017. URL: <https://www.silextechnology.com/hubfs/Resource/%20PDF/DS-520AN%20Product%20Brochure%20.pdf>.
- [171] Xilinx. *Vivado Design Suite User Guide: Synthesis*. Version v2021.2. UG901. 2021. URL: <https://docs.xilinx.com/v/u/2021.2-English/ug901-vivado-synthesis>.
- [172] S. P. Morgan. “Queuing disciplines and passive congestion control in byte-stream networks”. In: *IEEE Transactions on Communications* 39.7 (1991), pp. 1097–1106.
- [173] A. M. A. Roa, H. K. Andersen, and A. C. T. Martinsen. “CT image quality over time: comparison of image quality for six different CT scanners over a six-year period”. In: *Journal of applied clinical medical physics* 16.2 (2015), pp. 350–365.
- [174] K. Gulliksrud, C. Stokke, and A. C. T. Martinsen. “How to measure CT image quality: variations in CT-numbers, uniformity and low contrast resolution for a CT quality assurance phantom”. In: *Physica Medica* 30.4 (2014), pp. 521–526.

- [175] E. Husby, E. D. Svendsen, H. K. Andersen, and A. C. T. Martinsen. “100 days with scans of the same Catphan phantom on the same CT scanner”. In: *Journal of applied clinical medical physics* 18.6 (2017), pp. 224–231.
- [176] T. D. DenOtter and J. Schubert. *Hounsfield Unit*. StatPearls Publishing, Treasure Island (FL), 2021. URL: <http://europepmc.org/books/NBK547721>.
- [177] T. P. Laboratory. *Catphan 500 and 600 Product Guide*. Online; Accessed: 23.01.2022. URL: <https://static1.squarespace.com/static/5367b059e4b05a1adcd295c2/t/615ef40255dbd2709cd9cfbd/1633612803610/CTP500600ProductGuide20211006.pdf>.
- [178] S. Abdurahman. *Generic Computed Tomography (GCT)*. Online; Accessed: 07.12.2023. 2021. URL: <https://gitlab.stimulate.ovgu.de/shiras-abdurahman/gct>.
- [179] C. Huang et al. “Proceedings of the 17th Virtual International Meeting on Fully 3D Image Reconstruction in Radiology and Nuclear Medicine”. In: *arXiv preprint arXiv:2310.16846* (2023).
- [180] A. Fidler, U. Skaleric, and B. Likar. “The impact of image information on compressibility and degradation in medical image compression”. In: *Medical physics* 33.8 (2006), pp. 2832–2838.
- [181] NVIDIA Corporation. *NVIDIA A100 Data Center GPU Datasheet*. <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf>. Online; Accessed: 19.01.2024. 2024.
- [182] D. C. a varex image company. *XC-THOR PHOTON COUNTING X-RAY DETECTOR*. [https://www.varximaging.com/wp-content/uploads/2022/01/XC-THOR\\_PDS.pdf](https://www.varximaging.com/wp-content/uploads/2022/01/XC-THOR_PDS.pdf). [Online; Accessed: 16.11.2022].

## **Confidentiality Note**

Please note that this thesis contains information that is subject to confidentiality constraints. In compliance with these constraints and to respect the privacy and proprietary rights of the involved parties, certain data, findings, and discussions have been omitted or presented in a generalized form.

These modifications have been made to adhere to ethical guidelines, legal requirements, and confidentiality agreements that govern the use of sensitive information. The omitted or generalized information does not detract from the overall findings and conclusions drawn in this thesis.

## **Declaration**

### **Ehrenerklärung**

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die Hilfe eines kommerziellen Promotionsberaters habe ich nicht in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Verwendete fremde und eigene Quellen sind als solche kenntlich gemacht.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann.

Ich erkläre mich damit einverstanden, dass die Dissertation ggf. mit Mitteln der elektronischen Datenverarbeitung auf Plagiate überprüft werden kann.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

### **Declaration of Honor**

I hereby declare that I produced this thesis without prohibited external assistance and that none other than the listed references and tools have been used. I did not make use of any commercial consultant concerning graduation. A third party did not receive any nonmonetary perquisites neither directly nor indirectly for activities which are connected with the contents of the presented thesis.

All sources of information are clearly marked, including my own publications.

In particular I have not consciously:

- Fabricated data or rejected undesired results

- Misused statistical methods with the aim of drawing other conclusions than those warranted by the available data
- Plagiarized data or publications
- Presented the results of other researchers in a distorted way

I do know that violations of copyright may lead to injunction and damage claims of the author and also to prosecution by the law enforcement authorities. I hereby agree that the thesis may need to be reviewed with an electronic data processing for plagiarism.

This work has not yet been submitted as a doctoral thesis in the same or a similar form in Germany or in any other country. It has not yet been published as a whole.

Magdeburg, November 13, 2024

---

Daniele Passaretti