Performance and complexity optimization in heterogeneous multiprocessors system on chip

Bouthaina Dammak Masmoudi

Thèse de doctorat

Pour obtenir le grade de Docteur de l'Université de

Valenciennes et du Hainaut-Cambrésis

et de l'Ecole Nationale des Ingénieurs de Sfax

Disciplines :

Ingénierie des systèmes informatiques (ENIS) / Informatique (UVHC)

Présentée et soutenue par Bouthaina DAMMAK MASMOUDI

Le 06/11/2015

Ecole doctorale :
Sciences Pour l'Ingénieur (SPI)
Sciences et Technologies

Equipe de recherche, Laboratoire :
Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH)

Optimisation des performances et de la complexité dans les architectures multiprocesseurs hétérogènes sur puce

Performance and complexity optimization in heterogeneous multiprocessors system on chip

JURY

Président du jury
- Mohamed MASMOUDI (Professeur à l'ENIS)

Rapporteurs
- Ahmed Chiheb AMMARI (Professeur à l'INSAT)
- El-Bay BOURENNANE (Professeur à L'UB)

Examinateur
- Mouloud KOUDIL (Professeur à l'ESI)

Co-directeurs de thèse
- Smail NIAR (Professeur à l'UVHC)
- Mohamed ABID (Professeur à l'ENIS)

Co-encadrants
- Mouna BAKLOUTI (Maître-assistante à l'ENIS)
- Rachid BENMANSOUR (Maître de conférences à l'UVHC)

Acknowledgements

Purshing a PH.D. project is both painful and enjoyable experience. It's just like climbing a high pick, step by step, accompanied with bitterness, hardship, frustration, encouragements and trust and with so many people's kind help. When I found myself at the top enjoying the beautiful scenery, I realized that it was, in fact, teamwork that meet me there. Though it will not be enough to express my gratitude in words to all these people who helped me, I would still like to give my many,many thanks to all these people.

First of all, I would like to thank my supervisor Pr. Smail NIAR for his continuous support of my Ph.D study and related research, for his patience, motivation, immense knowledge and for the nights we were working before papers deadlines. His guidance helped me in all the time of research and writing of this thesis. It was a honour for me to share of his scientific knowledge but also of his extraordinary human qualities. I could not have imagined having a better advisor and mentor for my Ph.D study.

I also would like to thanks my co-supervisor Mouna BAKLOUTI for her continuous support, proposed ideas, valuable discussions and constructive suggestions. Her cooperation, positive attitude and understanding really deserve an everlasting appreciation. I could not wish for a better or friendlier co-supervisor.

A special thank to Rachid BENMANSOUR from LAMIH laboratory, the first person who taught me what linear programming is. I am grateful for his corporation in the work of space exploration and for his help to revise scientific papers.

My sincere thanks also goes to my supervisor, Pr. Mohamed ABID, director of the CES Laboratory. He has been supportive since the days I began working on CES. Ever since, he has supported me not only by providing a research assistantship over almost six years, but also academically and emotionally through the rough road to finish this thesis. Thanks to him.

To my colleagues and friends of the CES laboratory and LAMIH laboratory (Faten, Rim, Agnès, Manel, Amina, Youmna, Asma, Rim, Zainab), I thank them for their companionship and for providing a so pleasurable and working atmosphere. The moment of leisure shared together helped me to overcome some more difficult moments.

I would like to acknowledge all the teachers I learnt from since my childhood, I would not have been here without their guidance, blessing and support.

Some of my best friends are family, some I've known since many years, and others are newer friendships that continue to grow stronger by the day. Although they are all very different, every one of them is extraordinary.
I wish to thank, Mariem and Nihel, for their love, care and moral support. Thank you for being there for me when I call you and need someone to just listen. You were always beside me during the happy and hard moments to push me and motivate me. The journeys in France was easier when we travel together.
Thank also goes to my freinds Saousan, Faten, Emna for their love and moral support.
Thanks to my freind and my cousin Kods to be close to me to change my mood when I am down. Tanks Kods for your help to have nice pictures for my manuscript.
Thanks to my freind and my aunt Fatma for her advices and her encouragement over rough spots.
Thank you doesn't seem sufficient for everyone of you but it is said with appreciation and respect to all of you for your precious friendship.

I am very grateful to my mother, Najoua, and my father Adel. They continue to learn, grow and develop and they have been a source of encouragement and inspiration to me throughout my life. A very special thank you for providing a 'writing space' and to look after may baby 'chahead' through the months of writing my thesis.

For my sisters and my brother ,Dora, Yousra and Wajih, I am grateful for the myriad of ways in which, throughout my life, you have actively supported me in my determination to find and realise my potential. Thanks for never giving up on me and being my best friends.

For my Finally, and most profoundly felt indebtedness, is my husband Rached. Over the fours years, you have been steadfast in your support of my research even when it involved long periods apart. A very special thank you for your practical and emotional support as I added the roles of wife and then mother, to the competing demands of work, study and personal development. And thanks my dearly beloved daughter, Chahead. I'm sorry that left you off care. Even you are so young month, your smile gives me chance to complete this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Context

In the last few years, the number of different consumer electronics products supporting multimedia applications have rapidly grown. Digital TVs, DVD players, game consoles and multimedia-enabled mobile phones are a few examples of such products. The vast majority of these products have specific-purpose processors embedded in them. The computation imposed on these embedded processors are dominated by multimedia applications including digital processing of media streams, such as audio, video, image, as well as other kinds of streaming data. A typical multimedia application consists on receiving data streams from the environment and processing these streams using various algorithms. In order to deliver processed media streams with a good quality, timing constraints have to be met.

The need to implement the compute-intensive multimedia applications under timing constraints suggests that the embedded system have to be designed to handle complex computations. In addition, the wide spread deployment of multimedia embedded systems in the consumer electronics products has exercised competitive pressure for optimizing their energy consumption and cost. In addition, continuous emergence of new multimedia standards, coupled with ever increasing complexity of multimedia applications, motivate flexible architectures. All these requirements increase the pressure on designers to constantly search for architectural solutions for multimedia devices.

Figure 1.1 shows the trends for multimedia embedded approaches like Application Specific Integrated Circuits (ASICs), Digital Signal Processors (DSPs), Application-Specific Instruction Set Processors (ASIPs), and Heterogeneous Multi-Processor System-on-Chip (Ht-MPSoCs). These approaches do not necessarily provide the above mentioned requirements of multimedia embedded systems. Each approach has its its own advantages and drawbacks [9]. In the last few years, Commercial vendors have turned to the use of Ht-MPSoC solution that offers the more adequate solution to next generation complex mobile multimedia applications [10] [11].

Heterogeneous Multiprocessor system-on-chips (MPSoCs)

For decades, Multiprocessor Systems-on-Chip (MPSoC) [12] was adopted as suitable platforms for multimedia systems. In fact, on one hand the data-flow nature of multimedia applications favours the use of multiple processors which operate on different

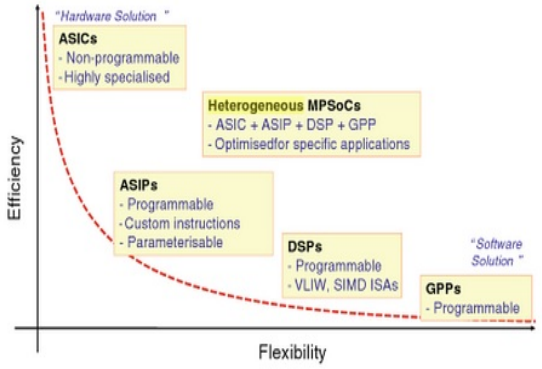

FIGURE 1.1: Comparison of trends for multimedia embedded systems [1]

data streams [13] [1], enabling a pipelined execution for high performance. This solution combines the flexibility of general purpose processor along with a convincing speed-up. These systems consist of a number of general purpose processors, memory units and interconnection subsystem. On the other hand, multimedia applications are complex and heterogeneous in nature [13] [14]; that is, the type and complexity of computations vary across applications tasks. For example, in H264 application, motion estimation task performs correlation on macro-blocks while Discrete Cosinus Transform (DCT) performs a large number of multiplications and additions. Therefore, processor customization has been emerged in the last few years as a solution to bridge the gap between the general-purpose aspect of traditional MPSoC architectures and the ever-increasing complexity of multimedia algorithms in each successive generation. The processor customization consists on coupling the general purpose processors along with customized functional units to execute particular functions aiming to deliver a best performance and a minimal power consumption. These customized functional units are a hard-wired solution ranged from a simple operation to an embedded processor such as DSP. These architectures are called heterogeneous MPSoC (Ht-MPSoC). There are mainly two modes to integrate the customized functional units: loosely coupled and tightly coupled modes. In the tightly coupled mode, the customized functional unit is a part of the processor data path. The loosely coupled mode integrates the customized functional unit as a peripheral and is called a hardware accelerator. Figure 1.2 is an overview of such architecture with a loosely coupled hardware accelerator.

In Ht-MPSoC architecture, the critical portions of the applications take advantage of the hardware implementation and are executed on customised functional units.

- **Flexibility of multimedia Ht-MPSoCs**
  The flexibility of a Ht-MPSoC architecture for multimedia domain implies the ability to implement multiple multimedia standards so that several variants of a product can be quickly deployed. This requirement implies the use of programmable

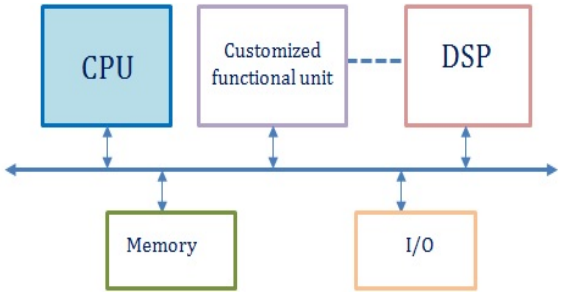

FIGURE 1.2: Ht-MPSoC architecture with loosely coupled hardware accelerator

processing elements such as general-purpose processor and DSPs as building blocks of the Ht-MPSoC.

- **Scalability of multimedia Ht-MPSoCs**
  The Ht-MPSoCs are scalable enough to allow easy addition of components in future, processors, memories and/or customized functional units, to handle the complexity of new multimedia generation without major redesign effort.

- **Performance, area and energy trade-offs of multimedia Ht-MPSoCs**
  Ht-MPSoC architectures use general-purpose processors to allow flexibility and integrate customized functional units to provide extreme customization to match processors to applications tasks and thus provide high performance. However, Ht-MPSoCs for multimedia domain are designed to be deployed in embedded devices, and thereby favours minimal area usage and lowest possible power consumption. Thus, a search of the design space is required for the minimum area usage and under an execution time constraint which is often imposed in multimedia applications.

1.2 Problem

The key advantage to increase performance of multimedia embedded systems is to increase the number of customized functional units in the architecture. However, the exploitation of the full potential of hardware customization lead to an immense increase in die area. Each task that will be customized will add a substantial area overheads in cost and power. Based on profiling results, traditional design flows of hardware/software partitioning consist on implementing the most computational tasks on hardware accelerators. Indeed, most computational tasks may bloat the die area without providing the required performance. In these cases, the most computational tasks may provide a minor speed-up with a high area overheads in cost and power. In addition, for multimedia embedded systems, the number of computations tasks is higher as the number

of applications to be executed increase. The problem of the optimal set of tasks to be customised can be solved based on space exploration methodology. A typical selection approach would include the tasks that have the best area-performance trade off. Integer Linear Programming (ILP) is a widely used technique for optimising MPSoC architectures, and has been already used in several works. Such selection would be more efficient if we consider a resource sharing approach to implement the same computational tasks to the same functional units. The benefit of a resource sharing methodology is well understood. When two tasks execute the same functionality, the same functional unit can perform the computation of these two tasks. If the selection process considers hardware sharing approach, this latter needs to be aware of the effects of resource sharing.

1.3 Our Work

This thesis presents a novel technique in the field of optimizing the complexity and performance for Ht-MPSoC architectures for multimedia domain. The main contributions of this thesis are summarized as follows:

- **A hardware accelerators sharing methodology which is based on the identification of similar tasks between the different applications executed on the different processors in order to be shared between processors.** Similar tasks are implemented with a reduced number of hardware accelerators. The proposed methodology allows an intelligent exploitation of hardware resources in order to deliver the best area performance trade off.
- **Novel classes of Ht-MPSoC architecture are proposed.** The first class is a symmetric Ht-MPSoC architecture where all the processors execute the same application, thereby they have the same number and types of hardware accelerators. The second class is an asymmetric Ht-MPSoC where all the processors execute different applications and they may have different numbers and types of hardware accelerators.
- **A mixed integer linear programming approach is proposed to explore the space of configurations of tasks which are candidates for hardware implementations.** The proposed model takes in consideration the hardware accelerators sharing to generate the efficient architecture. The impact of the delay problem when a task is shared between two or more processors is controlled.
- **A technique that identifies, in short times, computational tasks to be executed on HW accelerators.** Our technique is based on the proposed MILP

model to identify the patterns to be customised and the configuration of their HW accelerators. This is performed via measuring the area usage and performance gain of different possible configurations of the space of solutions in order to find the optimal one. To reduce the time to search the optimal (local optimum) solution, the technique is based on an iterative approach.

This chapter presents experimental results obtained during the validation of the contributions presented in chapters 3 and 4. In order to validate and evaluate the proposed HW accelerators sharing methodology, we present in section 5.3 a case study based on real application. A discussion on the impact of HW accelerators sharing on performance, area and energy trade-off is presented. The proposed technique for the selection of optimised Ht-MPSoC architecture is evaluated in section 5.4. Section 5.5 concludes this chapter.

- **An experimental evaluation of the thesis contributions is performed with XILINX FPGA board.** Implementation results of different Ht-MPSoCs configurations are presented and a discussion on the impact of HW accelerators sharing on performance, area and energy trade-off is performed. The effectiveness of MILP model of the proposed technique is evaluated based on experimental results of synthetic and real applications. For each case study, we explore the design space configurations for different performance constraints and we compare our technique solutions to real FPGA measurements.

1.4 Thesis Organization

This thesis is organized as follows:

Chapter 2 presents an overview of the technologies and techniques considered relevant to this thesis. We focus on giving a background materials in the domain of embedded systems and multiprocessor architectures. This is followed by presenting a study of related works in the field of Ht-MPSoC architecture and space exploration for application specific instructions extension.

Chapter 3 presents the hardware accelerators sharing methodology that can be promising for multimedia applications. We present also the methodology to migrate a c/c++ task to a HW accelerator. Section 3.5 describes the proposed symmetric Ht-MPSoC and asymmetric Ht-MPSoC architectures and the interconnection network for these proposed

architectures.

Chapter 4 details the proposed technique for the selection of hardware accelerators. The MILP models proposed for SHt-MPSoC architecture and AHt-MPSoC architectures are detailed.

Chapter 5 presents the experimental results to validate the proposed methodologies. XILINX FPGA platforms have been used to implement the designed architectures.

Chapter 6 presents conclusions and opens new opportunities for future work.

Chapter 2

Background and Related Works

Contents

2.1 Introduction

This chapter presents an overview of techniques and tools considered relevant to this thesis. We give a background materials in the domain of reconfigurable embedded systems and multiprocessor architectures. This is followed by presenting a study of related works in the field of Ht-MPSoC architecture and space exploration for application specific instructions extension.
The chapter starts with an introduction to multimedia embedded systems in section 2.2. This is followed by an introduction to multi-processor systems. Section 2.4 provides a focus on FPGA-based multiprocessor architectures. Techniques for processor customization is discussed in section 2.6. Section 2.7 presents the existing works that deals with resource sharing for embedded systems on-chip. Finally, in section 2.8, we present prior works considered relevant for design space exploration for optimising area/performance trade-off when customizing an MPSoC architecture.

2.2 Multimedia and Embedded Systems

An embedded system is a computing system which is designed for specific functions and is embedded as part of the complete device which may include hardware and mechanical parts. Thereby, in contrast with general-purpose computers, an embedded system performs a few pre-defined tasks, with very specific requirements. Typical examples of embedded systems include MP3 players, smart cameras and cellular phones. In the last few years, there has been a widespread deployment of embedded systems in a wide range of electronic and communication systems [15]. The combination of embedded systems and multimedia communications is the key reason of the on-going evolution of modern high-tech electronic equipment, ranged from mobile phone to set up boxes [15][16].
The efficiency of embedded multimedia systems is primarily shaped by performance and power concerns [17]. Due to the huge amount of processing for multimedia applications and in order to run with sufficient performance and from inexpensive batteries, better speed-up and lower power is the challenging requirement for multimedia embedded systems [17] [18]. In the following section, we discuss the background of the technologies commonly used for multimedia embedded systems that hold a great promise for improving performance and energy efficiency of these electronic devices. Such technology include GPUs, ASICs, FPGAs and DSPs.

2.2.1 Technologies Used to Implement Multi-media Embedded Systems

Timm et al. [19] compare the performance and energy efficiency of CPU with those of GPUs for several multimedia benchmarks. They proved that GPU offers significant performance advantage over CPU and hence it outperforms CPU in energy efficiency. In [20], Mu et al compare the energy efficiency of GPUs with that of DSPs for a broad range of signal processing applications. They have observed that GPU provides a better performance than the DSP, however, its energy efficiency is less optimised compared to that of the DSP. Mencer et al. [21] compare the energy efficiency of FPGAs with that of DSPs for data encryption algorithm. They have concluded that the FPGAs provide better energy efficiency than DSPs. In [22], the authors provide a comparison between FPGA, ASICs and DSPs and conclude that ASICs and FPGAs are more suited for high efficient multimedia systems.

TABLE 2.1: Comparison of technology used for multimedia embedded systems
Legend: +++: excellent, ++: good, +: moderate, -: poor

Technology	Flexibility/ programmability	Performance	Power usage
GPU	++	++	-
DSP	++	+	+
FPGA	+	++	+
ASIC	-	+++	+++

Table 2.1 summarizes the comparison of technologies commonly used for multimedia embedded systems. From this table, we conclude that hardware solutions (ASIC, FPGA) would yield the most efficient multimedia design. In the following subsection, we justify the use of FPGA technology for our work.

2.2.2 Discussion on FPGA and ASIC Technologies

Many on-line resources compare FPGA and ASIC design flows in order to release the benefits of each technology From [23] [24][23], we note that ASIC and FPGA design flows are somewhat similar. The main difference is that the whole FPGA flow is GUI (Graphical User Interface) driven through CAD tools while ASIC flow cannot be only performed by user. The logic design of an ASIC flow is driven by scripts and is made by user, however, within the physical design, the ASIC must be sent to the foundry for manufacturing. The intervention of the manufacturer raises the main disadvantages of ASIC design, which are a slow time-to-maket (TTM), the non-recurring engineering (NRE) cost and a high manufacturing cost [25] [26]. However, the primary advantages

of manufactured ASICs over programmable FPGAs are the optimized performance and the reduced power and area consumption [25] [26]. In [27], authors prove that FPGA requires almost 20 to 35 times more logic area than an ASIC and has a speed performance 3 to 4 times slower than an ASIC. It was also proved that FPGAs consume approximately 14 times as much dynamic power.
For these reasons, the traditional multimedia systems target ASIC platforms. However, with the advent of FPGAs integration capabilities and for the interest of fast programmability, prototyping and short TTM, FPGA are used for systems where TTM and flexibility are in concern. Tanks to these features, FPGAs can now play in applications and markets that were previously "owned" by ASICs and other devices [28].

In the context of test system, and in order to save time and cost, in this thesis we target FPGA platform as a substitute to ASIC platform. We used FPGA platform from Xilinx Inc. [29], specifically devices XC5VFX70T from the Virtex-5 family [7].

2.3 Multiprocessor System-on-Chip

The multiprocessor System-on-Chip (MPSoC) is a system in a single chip (SoC) which uses multiple processors, usually designed for embedded applications [30]. The MPSoC architecture is a promising trend in recent multimedia embedded applications. The parallel execution paradigm of this architecture allows to take advantages of data, instruction or thread parallelism aspect of multimedia applications [31] and thereby to meet real-time performance and low power consumption demands of these applications. Apple A5X is an example of MPSoC integrating a quad-core Power VR, which drastically increases its video processing capabilities [32].
subsection we present the widely classification of MPSoC architecture (homogneous and heterogenous).

Commonly, according to their architecture type, MPSoCs are classified by two approaches: homogeneous and heterogeneous [33] [34] [35]. It is much easier to develop a MPSoC following the first approach. Indeed, homogeneous approach consists on identical processing elements while the heterogeneous approach consists on different types of processing elements communicated through a hardware communication system.

Homogeneous MPSoC

Homogeneous MPSoCs use generally the paradigm of SMP [36] (Symmetric MultiProcessor) and embed two or more homogeneous soft-cores with main shared memory. In

an SMP system, a pool of homogeneous processors working independently on different data are tightly coupled through a hardware interconnection mechanism.

The inherent architecture of homogenous MPSoC is flexible and scalable. However,

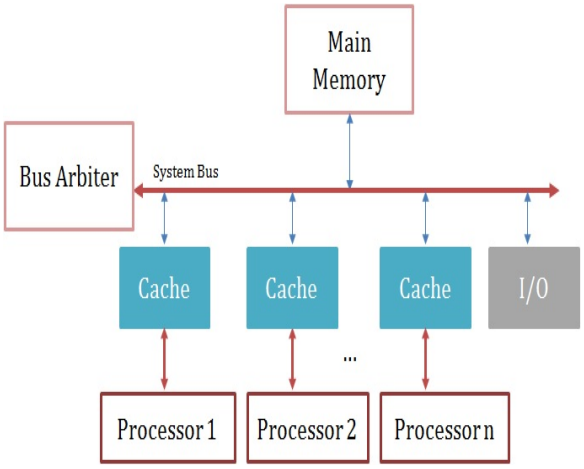

FIGURE 2.1: Homogeneous MPSoC architecture

for multimedia application, the number of processing units is limited by the level of parallelism of the application. Thereby, the increase of the number of processors is not straightforward; scalability could be limited due to several factors such as the level of parallelism ofthe applications, organization of the memory, the interconnection infrastructure, etc [37]. Thus, the performance of this class of architecture, for multimedia applications, is limited. Also, due to the general purpose aspect of homogeneous MPSoC, the power consumption of these architectures is not optimised.

Heterogeneous MPSoC

Heterogeneous MPSoCs consist on several processing units of different types, such as soft-cores, hard-cores, HW accelerators, etc., communicated through hardware interconnection mechanism. This type of architecture is typically designed to deliver best-case performance. To take up the challenges imposed by multimedia processing (high performance, low power consumption), designers are turned to the use of heterogeneous MPSoCs [38] [**?**]. Figure 2.2 illustrates a survey performed in [2] and shows that more than 50% of MPSoC are heterogeneous.

In heterogeneous MPSoC, processors difference may vary from higher level details such as instruction set to architectural details such as memory size and clock frequency.

A simple heterogeneous MPSoC can be designed using multiple copies of the same core. This mean that the cores execute the same instruction set, but have different capabilities and performance levels. Such technology include ARM big.LITTLE architecture

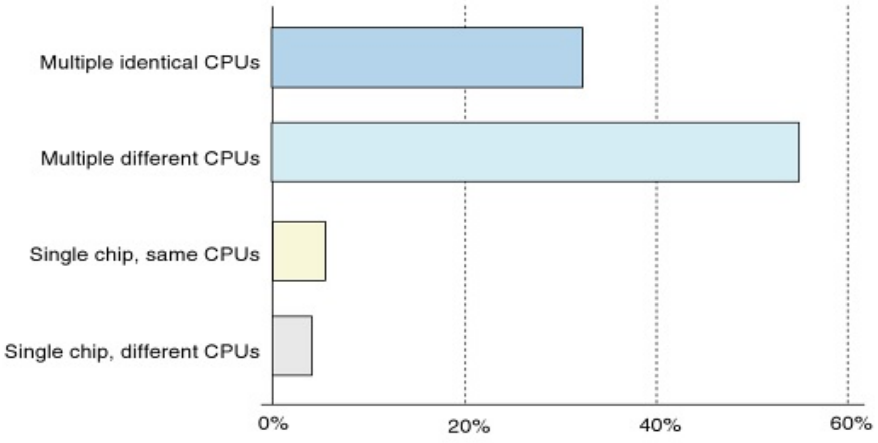

FIGURE 2.2: Types of processors in SoC [2]

[39], which combines relatively low-power processor cores (LITTLE) with relatively more powerful and power-hungry ones (big). This model of architecture has been implemented in the Samsung Galaxy S mobile phones.
Recent research prove that heterogeneous architectures gain performance not just by adding cores, but also by incorporating specialized processing capabilities to handle particular tasks. In [40], the authors show that heterogeneous-ISA architecture outperforms the same-ISA architecture by 21% with 23% energy savings. The benefits of using specialized processing elements with different ISE in heterogeneous architecture have lead to the deployment of specialized processors. Examples of these specialized processing elements include:

- Vector processors: A Vector processor is a processor that can operate on an entire vector in one instruction. The operand to the instructions are complete vectors instead of one element[41].
- Digital Signal Processors (DSPs): A DSP is a specialized microprocessor that has an architecture which is optimized for the fast operational needs of digital signal processing.
- Specialized coprocessors: A special-purpose processing unit that assists the main processor in performing certain types of operations. Coprocessors can deliver noticeable improvements on mathematically intense functions, such as multiplying or inverting matrices or solving n-body problems.

In the last few decades, thanks to their significant evolution of integration capacity [42] [43] [44], FPGA platforms have become feasible to host a complicated MPSoC system. Now, FPGAs are used not only for prototyping, but also for implementing final

design. For such MPSoC, processing elements consist on soft-core processors, DSP, Hw accelerator, etc. These processing elements are used as Verilog or VDHL description that can be extended or reconfigured and that are afterwards synthesized for the target FPGA. The design of FPGA-based MPSoC architecture presents some advantages that compensate the use of ASIC in some way.

- Flexible and reconfigurable: The number of soft-core processors depends on the target FPGA and it can reach 80-100 processors. Moreover, the configuration of each processor could be modified (on-chip memory size, enable FPU, etc.) and the designer has only to re-synthesize to implement the new design.
- Less TTM: The considerable reduced TTM is the primary advantage of using FPGA .
- Less cost: The cost of the design process is relatively cheap. Also, an error occurred during the design process can be altered for no additional cost.

The Cray X1E [45] supercomputer is an example of heterogeneous-ISA architecture. It incorporates both vector processing and scalar processing, and a specialized compiler that automatically distributes the workload between processors. The Cell processor architecture is a second example of heterogeneous-ISA architecture. This architecture combines a general-purpose Power Architecture core of modest performance with streamlined coprocessing elements which greatly accelerate multimedia and vector processing applications. The Cell processor is designed by IBM, Sony and Toshiba to accelerate gaming applications on the Playstation 3).

2.4 FPGA-based MPSoC: Architecture Background

Both homogeneous and heterogeneous MPSoC are mainly composed of three subsystems: processing elements (soft-cores, hardcores, DSP, Hw accelerators), memory hierachy and the hardware interconnection mechanism. In the following subsections, we describe in detail architecture background of these subsystems commonly used in FPGA-based MPSoCs.

2.4.1 Processing Elements

In FPGA-based multiprocessor systems, most used processing systems are either soft-cores, hardcores, HW accelerator or DSP. In the following, we detail each processing system and we give examples of the most used ones.

2.4.1.1 Soft-core processors

A soft-core processor is a microprocessor described in an HDL language, which can be synthesized in programmable hardware, such as FPGAs. These processors implemented in FPGAs can be easily configured to the needs of the target application. FPGA manufacturers provide commercial soft-core processors. Xilinx offers its MicroBlaze processor [5], while Altera has Nios and Nios II processors [46]. If the designer is company-independent, there is a wide range of soft-cores that can be used in FPGA-based MPSoC [47]. Such company-independent soft-cores are the LEON3 from Aeroflex.

- **NIOS II processor**

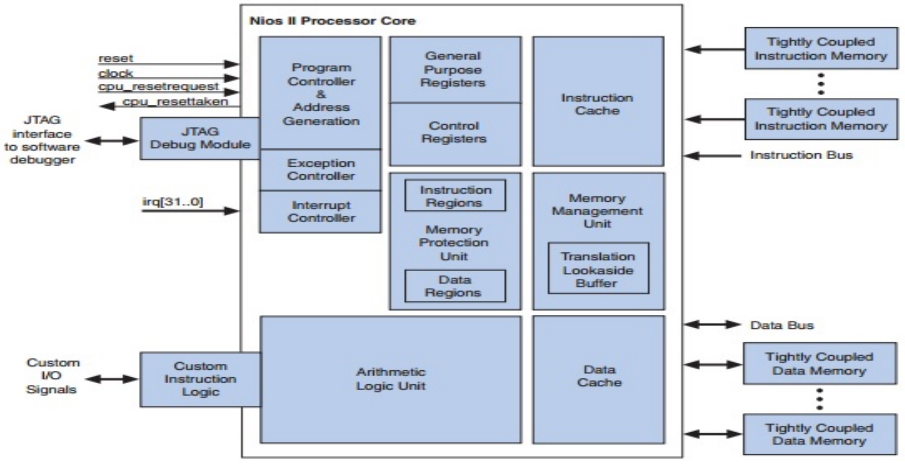

FIGURE 2.3: NIOS processor architecture [3]

The Nios II embedded processor (Figure 2.3 has a Reduced Instruction Set Computer (RISC) architecture. Its arithmetic and logic operations are performed on operands in the general purpose registers. The data is moved between the memory and these registers by means of Load and Store instructions. The word length of the Nios II processor is 32 bits. All registers are 32 bits long. Byte addresses in a 32-bit word are assigned in little-endian style, in which the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word. The Nios II architecture uses separate instruction and data buses, which refers to the Harvard architecture [48].

- **OpenRISC**

  OpenRISC (Figure 2.4) is a soft-core processor that is distributed under the GNU License and it has been used in various industrial applications.
  OpenRISC 1200 is a 32-bit RISC processor core compliant to the Harvard architecture [49] with 32 general purpose registers that implements ORBIS32 ISA.

OpenRISC implements the standard RISC scalar processor with five stage single-issue pipeline. It also supports a 32x32 Multiply-Accumulate Unit (MAC), digital Signal Processing operations and on-chip debug.

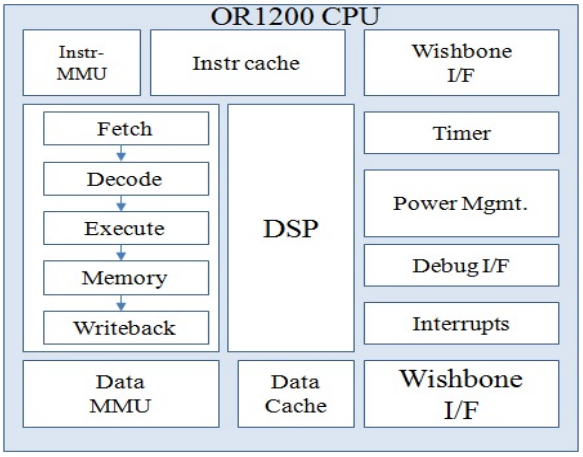

FIGURE 2.4: OpenRISC processor architecture [4]

- **Microblaze processor**

Microblaze is the soft-core processor of Xilinx, the most widely FPGA used in MPSoC. The microblaze architecture, shown in Figure 2.5, is highly configurable and parametrizable [5]. Examples of configurable and parametrizable features include cache size, pipeline depth (3-stage on 5-stage), memory management unit and bus interfaces.

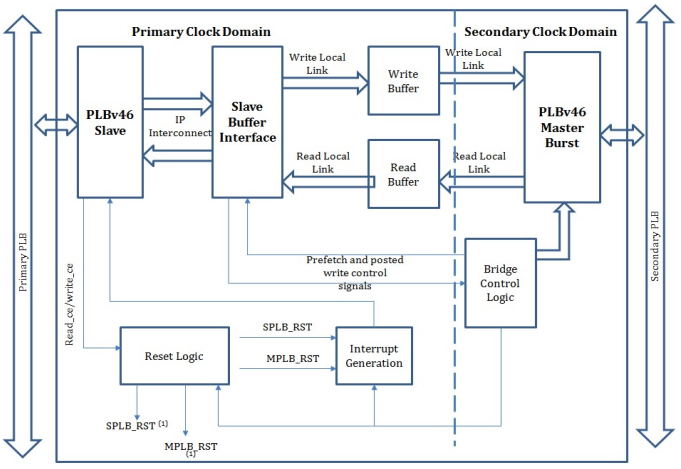

FIGURE 2.5: Microblaze processor architecture [5]

The MicroBlaze support different interconnect systems. The primary used system is the PLB bus, which is a traditional system-memory mapped transaction bus with master/slave capability. For communicating to local-memory, MicroBlaze uses a dedicated LMB interconnect. The user defined hardware accelerators or

peripherals use the FSL (Fast Simplex Link) bus [50], a special dedicated FIFO connection.

- **Leon 3 processor**

  The LEON3 [6] (see Figure 2.6) is a synthesisable VHDL description of a 32-bit processor compliant with the SPARC V8 extension set [51] developed by Aeroflex Gaisler.

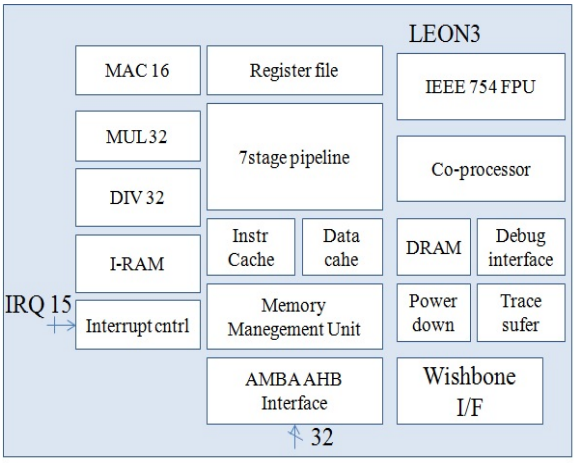

FIGURE 2.6: Leon 3 processor architecture [6]

The model is highly configurable, and essentially suitable for system-on-a-chip designs. The source code is available under the GNU GPL license, allowing free and unlimited use for research and education. The LEON3 processor has several features such as advanced 7-stage pipeline, fully pipelined FPU, Hardware multiply, divide and MAC units, etc.

In table 2.2, we summarize the main features of soft-core processors described above. From this table, we note that NIOS II and Microblaze processors present the best efficiency in term of maximum frequency and resource usage. In this thesis, in our laboratory XILINX FPGA are provided, thereby we use Microblaze processors.

TABLE 2.2: Main features of Soft-cores

	Leon3	Open RISC	Microblaze	NIOS II
Open source	No	Yes	No	No
Hardware FPU	Yes	No	Yes	Yes
Bus standard	AMBA	Wishbone	Core connect	Avalon
Coprocessors	Yes	Yes	Yes	Yes
Maximum frequency (Mhz)	210	47	200	290
Resources	4000 (slices)	2900 (slices)	1450(slices)	1400(Logic Element)

2.4.1.2 Hardware Accelerator

The integration of custom instruction in FPGA-based MPSoC increases the performance gain by incorporating hardware components to handle computational tasks [52][53][54][55]. Modern platforms, including FPGAs and ASICs support different couplings of hardware components with the processor. In [56], coupling schemes are classified into two principal modes: closely coupled mode and loosely coupled mode (Figure 2.7).

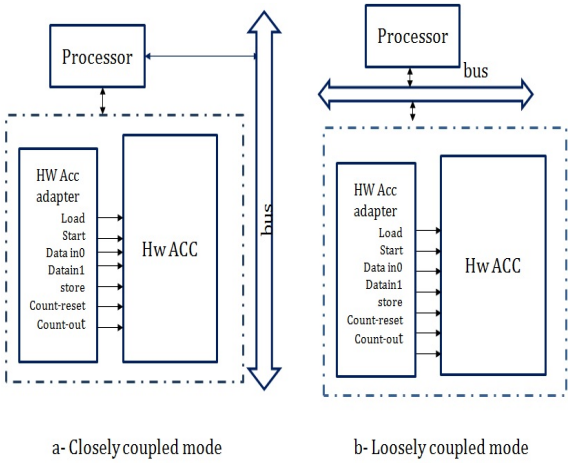

FIGURE 2.7: Hardware accelerator architectures classification: Closely coupled and loosely coupled

In the first mode, the hardware accelerator is part of the processor data path and has direct access to the processor memory. At the opposite, in the second mode, the accelerator is placed outside the processor on a dedicated bus [54][56][57]. A group of closely coupled hardware components operates at a single clock cycle fixed by the slower components. At the opposite, each loosely coupled hardware component runs at its fastest possible individual frequency. Loosely coupled mode is quite popular in multi-media applications like image encoding/decoding applications. Nomadik [10], Freescale i-Mx35 [58] and S3C6400 [59] are examples of multi-media architectures designed with loosely coupled accelerators. These platforms embed on the same die an ARM [60] processor and different multi-media accelerators for video, audio, imaging, and graphics processing.

2.4.2 Memory Infrastructure

There are two basic types of memory in MPSoC architectures, commonly named shared memory and distributed memory. Figure 2.8 shows block diagrams of these two types, which are differentiated by the way in which processors exchange information. In a shared memory, all processors uniformly share the same memory [61]. Processors communicate information by accessing the same memory location. The primary advantage

of the shared memory is their easy programmability, since there are no communications between the processors [62] [61]. However, due to collisions, MPSoCs with shared memory are generally limited to 32 processors.
With distributed memory architectures, since memory is not shared, inter-core communication between processors is required, and interconnection network performance becomes important.

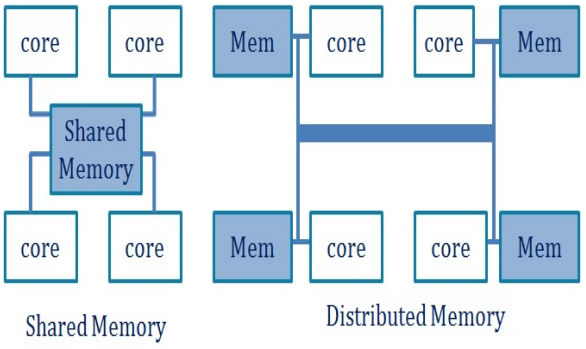

FIGURE 2.8: Shared and distributed memory in MPSoCs

2.4.3 On-Chip Interconnection Mechanism

As noted earlier, an MPSoC consists on a set of processing elements connected together by means of an interconnection mechanism. To meet the performance requirements of modern applications like multimedia applications, the design of hardware interconnection mechanism became a major focus of research in MPSoC design.

In the following subsections, we present a short survey on the existing interconnection approaches in MPSoC, and present the characteristics of each one.

2.4.3.1 Not Communicating Processor

This is the most basic topology. As shown in Figure 2.9, the architecture is composed of a duplication of a tile of components. For this architecture, the processors of the different tiles cannot communicate to divide a computation of the same task. Each one of them performs a specific computation.

2.4.3.2 Communication over a shared bus

The shared bus topology is a single communication path to which all processing elements and peripherals are connected (see Figure 2.10). For this topology, when only one

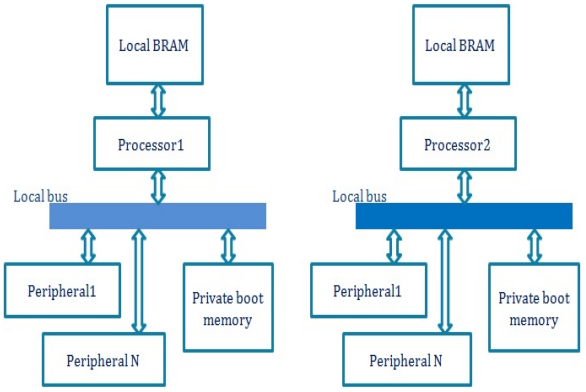

FIGURE 2.9: Not communicating architecture

master is used, no connection problem arises. However, when two or more masters are connected to the shared bus, an arbitration policy has to be considered. It is obvious that the principal advantage of a shared bus communication is its simplest interconnection structure and consequently its reduced design time. However, the arbitration policy occurs a limited bandwidth and a throughput proportional to the number of processing elements [63].

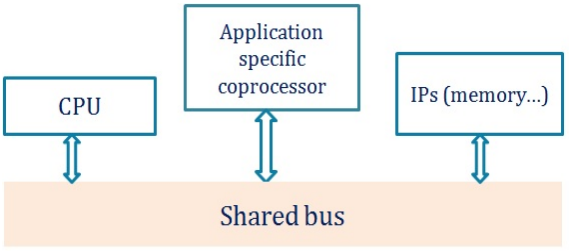

FIGURE 2.10: Communicated architecture over a shared bus

2.4.3.3 Point-To-Point Communication

The Point To Point communication is a direct communication between two communicating units (Figure 2.11). The data exchange over a Point To Point communication is efficient. However, for this topology, the complexity of the interconnection increases exponentially with the increase of the communicating units. In [63], the authors show a complex design of Point To Point interconnection despite the reduced number of connections (10 connections).

2.4.3.4 Crossbar Communication

In a crossbar communication (or also named bus matrix), every processing element on the architecture is connected to all others (See Figure 2.12). This communication is

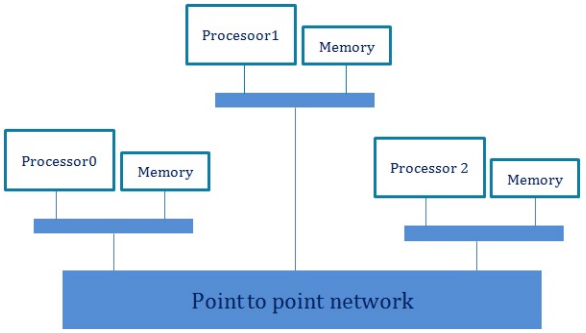

FIGURE 2.11: Communicated architecture over a PoinT To Point Topology

characterised by a non-blocking aspect, because each processing element can perform simultaneous data exchange with every other processing element without conflicts. In addition, the direct connections allow a direct communication of each sender-receiver couple of processing elements. Thereby, the crossbar communication delivers a high speed and a large bandwidth.

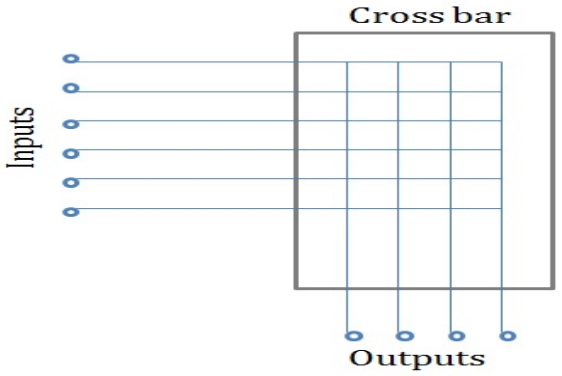

FIGURE 2.12: Crossbar communication

The crossbars are limited by their high cost due to their wiring complexity.

2.4.3.5 NoC communication

Network-on-Chip (NoC) is a general purpose on-chip communication concept, which tackles the problems of wire density in SoCs. As shown in Figure 2.13, typical NoC based system consists of processing elements (PE), network interfaces (NI), routers (R) and inter-router communication channels. It offers better scalability than on-chip buses-based interconnection, because as more resources are used, more routers and links are introduced to connect them to the network.

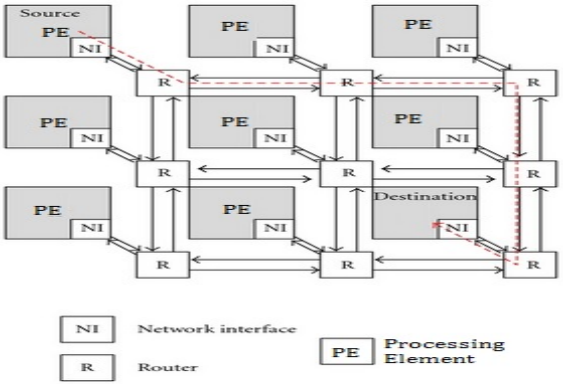

FIGURE 2.13: NoC with mesh topology

2.5 Processor Customization

The complexity of recent embedded applications and their deployment in our daily life have increased their demand on high performance and minimal power consumption. These requirement have reached a point where software execution can no longer follow these requirements. To tackle this problem, especially in multimedia embedded system, a common method is to use application-specific accelerators added to the general purpose processors. In the context of this thesis, we use the term task customization to denote the execution of the computations of specific task on application-specific accelerator. While the aim of our thesis is to create a Ht-MPSoC system through the integration of application-specific accelerators, it is important for us to present a background of techniques used in single-processor to customize a task. In the following subsections, we will expose common used processor customization techniques.

2.5.1 Fine-Grained Processor Customization

In fine-grained processor customization, the accelerators are tightly coupled to the processor data path as custom functional units or loosely coupled to processor as HW accelerator. For ASIPs (Application-Specific Instruction set Processor), these functional units are implemented in ASICs. For more flexibility, an ASIP can be coupled to reconfigurable functional units, which are implemented on reconfigurable hardware resources such as FPGA.

Static Fine-Grained Processor Customization

It is obvious that ASIP reaches the best performance when custom functional units are implemented in ASICs. The performance improvement depends on the number of implemented functional units and their area-performance trade-off. Thus, the complexity

to design an ASIP relies on custom instructions identification and selection. For this reason, efficient algorithms have been proposed to accelerate this process. These algorithms identify the computational tasks directly from an application graph satisfying architectural constraint imposed by processor target [64] [65] [66]. The imposed constraints include a number of inputs/outputs, number of operators and a delay of critical path. The selection process in [64], [65] and [66] is based on tree search algorithms and is further improved in [67] by ILP (Integer Linear Programming) and in [68] by a novel maximal convex subgraph enumeration algorithm.

Dynamic Fine-Grained Processor Customization

ASIPs integrate static custom functional units and thereby they suffer from limited flexibility. In contrast, reconfigurable ASIPs are flexible as they integrate reconfigurable functional units but with a performance trade-off. Many research have been achieved for the efficient designs of the reconfigurable fabric. Several papers survey the contributions of prior works on a single processor core extended with reconfigurable fabric [68] [69]. Theses architectures include Chimara [70], One-chip [71] and Stretch [72].

2.5.2 Identification of Custom Instructions

A distinguishing aspect of cited customization approaches is the identification process of the custom instructions. Instructions can be identified statically or dynamically during the execution. The primary drawback of dynamic identifications such as [73] [74], is to induce a large overhead to the processor, which can negate all the speedup provided by using custom instructions. In order to reveal the identification delay during run-time, [75] [76], customization process is exploited using an approach of static identification and dynamic realization. A static approach identifies computational subgraphs during compilation and replaces these latter with custom instructions at run-time using a translation table. Other works have proposed a dynamic reconfiguration of coarse grained hardware accelerators such as RISPP [77], which dynamically reconfigures FPGA resources to implement custom accelerator functions. Recently, the work presented in [78] tackle a challenging problem, as all mapping steps, from compiler analysis and optimizations to hardware generation, are considered to be both efficient and fast. Their approach extends a general purpose processor (GPP) with a reconfigurable processing unit (RPU), both sharing the data memory. Repeating sequences of GPP instructions are mapped to an RPU composed of functional units and interconnect resources, and able to exploit instruction-level parallelism through loop pipelining .

2.5.3 Coarse-Grained Processor Customization

A primary debate of processor customization covers the granularity of the accelerators: should it be designed at fine grained level [71] [70], or should it be coarse grained, i.e., an array of ALUs which communicated through programmable interconnect [79] [80] [81]. Each customization approach has its advantages and drawbacks. In general fine grained designs are more flexible. However, fine grained designs have a large overhead mainly in speed up and power consumption.

2.6 MPSoC Customization

Over the last decade, according to Moore's law, the number of raw transistors increased at 58% per year [82], whereas the capability of chip designers to design system on chip increased only at a rate of 20% per year. On the other hand, the complexity of recent embedded applications and their deployment in our daily life have increased their demands on high performance and short TTM. These requirements have reached a point where traditional homogeneous MPSoC architectures can no longer follow their demand. For the cited reasons, improving MPSoC with application specific instructions is a challenging solution. We denote by MPSoC customization the customization of a multi-processors design. Depending on whether the MPSoC system supports run-time reconfiguration or not, the MPSoC customization approaches could be divided into static MPSoC customization and dynamic MPSoC customization.

2.6.1 Dynamic Customized MPSoC

Dynamic MPSoC customization is realized by coupling the processors to application-specific functional units implemented in partial reconfigurable fabrics. To support partial reconfiguration, it is interesting to share large reconfigurable fabrics between cores in spatial or temporal sharing manner. However, minor researches have addressed how dynamic customization can benefit future MPSoCs. Many research efforts have been investigated to the integration of reconfigurable functional units on a single-processor architecture, including Chimaera [70] and DPGA [83] which tightly integrate reconfigurable fabric with the processor as application-specific functional unit.

The architectures proposed in [84] and [85] are ones of the minors research that explore the resource sharing of reconfigurable fabrics. Remap (Reconfigurable Multicore Acceleration and Parallelization) [84] is a run-time reconfigurable architecture for accelerating applications executed on the different processors of Ht-MPSoC architecture.

In ReMAP, the reconfigurable fabric is partitioned between clusters of processors. In each cluster, reconfigurable fabric is temporally shared between the different processors in a round robin manner. In [85], the authors present novel approach to minimize reconfigurable fabrics usage by resource sharing for closely coupled application-specific architectures. They develop an algorithm to select the ISEs to be mapped on the same fabric to optimize the fabric sharing between cores leading to the best execution time.

2.6.2 Static Customized MPSoC

Some research have been interested in developing design automation tools for single processor architecture customizations such Tensilica Xtensa [86] and CoWare [87] tool chain. Designing such tools for MPSoC customization is a much more tedious problem. Complex problems arise while exploring the design space such custom extension selections and other architectural constraints such as processing elements, memory hierarchies and chip interconnect mechanism. In term of MPSoC customization, a recognized work is [88] , where a formulation of design space exploration problem is proposed. They focus on extensible processors that combine base processor with application-specific instructions, to provide a good trade-off between flexibility, TTM, and performance. This work motivates the need for such an integrated approach by demonstrating that application-specific instructions selection has a serious assignment and scheduling problems. The exploration is based on an iterative improvement algorithm to a) partition tasks on processors and then b) select custom instructions along the critical path. It uses expected execution time to connect these two steps. A generally used technique for design space exploration in MPSoC design is based on Integer Linear Programming (ILP). Lately, the static MPSoC customization problem is formalized in [89] as a Mixed ILP (MILP) problem. They propose a formal approach based on MILP exploration and its implementation within a CAD tool for the optimization of Ht-MPSoC architectures. These heterogeneous systems, consisting of application-specific as well as of programmable processors, are highly suitable for performing complex schemes of image processing algorithms under real time constraints, which have an intractable running time when the number of processors scales. More recently, in [90] [91], the authors propose to partition the applications tasks onto a set of available processing elements. [92] looks for the optimal solution based on ILP formulas and presents a case study using JPEG application.

2.7 Resource Sharing in Embedded System Designs

MPSoC customization problem is quite challenging due to the complexity of optimising the area/performance trade-off. The requirements of a resource consumption/performance trade-off complicate this challenging problem. Meanwhile, resource sharing is a new research axis and the few prior works have investigate this problem. In this section we expose prior research in the field of resource sharing for embedded systems. For recent multi-media applications, a large number of custom instructions can be identified to be executed in hardware components. In order to avoid an excessive area usage of hardware components, previous works propose to identify and exploit commonality between identified custom instructions and to share hardware resources.
Resource sharing has already been studied in earlier work for closely coupled customization in uni-processor architecture. In [93], the authors propose a polynomial-time heuristic that uses resource sharing to minimize the area required to synthesize a Set of custom Instruction Extension (ISEs). Their resource sharing approach transforms the set of ISEs into a single hardware data path. Nevertheless, their proposed heuristic minimizes the ISEs area usage without a control on latency constraint. Zuluaga et al. [94] introduce latency constraints in the merging process of the ISEs to control the performance improvement. Their proposed parametric algorithm combines a path-based resource sharing algorithm, similar to the ones presented in [93], with a timing budget management scheme. More recently, the work presented by Stojilovič et al. [95] aims at a pragmatic increase in flexibility to integrate different ISEs from different applications. This work is motivated by data path based algorithm. While [93] aims at minimizing the area cost, [95] increases HW accelerators flexibility for a moderate cost. Their approach ensures that all Instruction Set Extensions (ISEs) from an application domain map on the same proposed domain-specific coarse-grained array. The architectures proposed in these papers belong to loosely coupled application-specific architectures. The cited works propose tools to share HW logic between different custom instructions for several tasks mapped on the same and single processor. Their proposed heuristics select the custom instructions to be mapped on logic providing a more area saving with operation sharing.

Recently, some researches have investigated the hardware resources sharing for MPSoC architectures. For the best of our knowledge, these research have only attempted the problem to share partial reconfigurable resources between different processors. In [85], Chen et al. investigated the problem of resource sharing for run-time reconfigurable multi-processor architectures. They develop an algorithm to select the ISEs to be mapped on the same fabric to optimize the fabric sharing between processors leading

TABLE 2.3: Survey on Resource Sharing for Embedded Systems

	Closely coupled	loosely coupled	MPSoCs	Static HW sharing	Run-time HW sharing	Exec time const
P.Brisk 2004	*	-	-	*	-	-
M.Zuluaga 2009	*	-	-	*	-	*
M. Stojilovic 2013	*	-	-	*	-	-
L.Chen 2011	*	-	*	-	*	*
M. Watkins 2010	*	-	*	*	*	-
Our work	-	*	*	*	-	*

to the best execution time. In [96], the authors propose a pseudo-polynomial time algorithm to explore the design space of Multi-Application Specific Instruction Processor (or M-ASIP). Their algorithm identifies the appropriate application-partitions and identifies custom instructions satisfying the area-performance trade-off.

In [96], Watkins et al. [21] proposed ReMap, a shared reconfigurable architecture for accelerating and parallelizing applications in a heterogeneous CMPs. In this architecture, reconfigurable fabric is shared spatially between clusters of processors. Processors of the same cluster can share temporally or spatially their reconfigurable fabric.
Table 2.3 summarizes the features of the different cited works. From this table, we note that all cited works are interested in closely coupled customization. However, in the context of FPGA based hybrid architectures, loosely coupled customization are much more efficient for recent applications in terms of performance/complexity trade-off. In [93] [94] [95], the proposed techniques optimize the hardware resource usage to customize a single-processor architecture by sharing hardware resources between different tasks. In [97] [85], the authors propose to share partial-reconfigurable hardware resources for multi-processor architectures. The custom instructions are implemented on runtime reconfigurable hardware resources. However, the primary drawback of using runtime reconfiguration is the significant delay of reprogramming the hardware. Thus, we think that the runtime reconfiguration delay and the sharing delay will dominate the total execution time, especially applications with a small amount of computation between two consecutive hardware accelerators.

2.8 Design Space Exploration of Customizable MPSoC Architectures

Customizing processors in an MPSoC architecture with application-specific instructions can lead to additional hardware resources in the architecture, but potentially significant improvement in performance. A naive integration of application-specific instructions consists on customizing the most computational tasks until the required performance is reached or until hardware resources constraint is overlapped. For such customization methodology, the most computational tasks may only enable minor speed-up but take too much hardware resources, and may prevent other tasks from being accelerated. Thus, designer would be typically interested in identifying how the performance improvement and on-chip consumption change with different choices of application-specific instructions on an MPSoC architecture. The identification of this trade-off necessitates effective exploration of a huge search space.
There are two main types of algorithms used in space exploration for MPSOCs customization and these are described in more detail below. In brief, the first type is based on exact algorithms [98] that use complex mathematical processes to output the entire set of solutions that satisfy the model exactly. The second type, known as heuristic algorithms [98], finds lower and upper bounds on the optimal solution.

2.8.1 Exact Algorithms

When considering exact approaches, the following techniques have had significant success: branch-and-bound, dynamic programming and in particular the large class of integer (linear) programming (ILP) techniques including linear programming.
Exact algorithms have been adapted for processor customization since 1996 (Binh et al. [99], Shrivastava et al. [100], Arato et al. [101]). In [101], two partitioning algorithms for HW/SW partitioning were presented by Arato et al. (2003): one based on Integer Linear Programming (ILP) and the other on Genetic Algorithm (GA). The authors proved that ILP-based solution works efficiently for smaller graphs with several tens of nodes and generates optimal solutions, whereas GA gives near-optimal and works efficiently with graphs of hundreds of nodes. The performance of GA was found to be uniform, whereas the run time of ILP was variable and depends on the number of nodes. More recently, exact algorithms have been adapted to the problem of HW/SW partitioning for MPSoC architectures [97] [91] [102]. In [97], the authors compare the dynamic programming approach and the ILP approach and it has been proved that dynamic programming approach is way faster than ILP approach and still generates a solution whenever the number of tasks increase.

2.8.2 Heuristic Algorithms

Heuristic is a technique designed for solving a problem more quickly when exact methods are too slow, or for finding an approximate solution when exact methods fail to find any exact solution. Many researchers have applied heuristic approach for processor customization. Particularly, genetic algorithms (Wu Jigang et al [103]; Greg Stit et al [104]; He Jifeng et al. [105]) and simulated annealing (Eles et al. [106], Henkel et al. [107] 2001, Lopez-Vallejo et al. [108]) have been extensively used. Other less popular heuristics are tabu search [106] and greedy algorithms [109].
More recently, some researchers used custom heuristics to solve hardware customization for MPSoCs. The proposed algorithm initially searches for the critical path in the task graph, and then assigns the task with the highest benefit-to-area ratio to hardware implementation. In [110], in order to minimize the overall execution time, a heuristic solution is proposed for scheduling and customizing on multi-processor system on chips (MPSOC).The proposed algorithm initially searches for the critical path in the task graph, and then assigns the task with the highest benefit-to-area ratio to hardware implementation. The critical path and the available hardware area are updated during the iteration. The whole calculation process works until the available hardware area is not enough to implement a software task lying in the critical path. Other custom heuristics are proposed in order to share hardware resources between different custom instructions. Such works include [94] [93] [109].
Unlike with exact approach, heuristic algorithms have been demonstrated to yield sub-optimal solutions. Exact algorithms are guaranteed to find an optimal solution and to prove its optimality. The run-time, however, often increases dramatically with a problem instance's size, and often only small or moderately-sized instances can be practically solved to proven optimality. In our work, we are interested to find the optimal configuration of a Ht-MPSoC architecture, where the number of processors is moderated (a maximum of 32 processors). Thereby, we decided to use ILP approach to explore our search space.

Chapter 3

Hardware Accelerators Sharing in Ht-MPSoC Architecture

Contents

3.1 Introduction

The increased demands for high performance and minimal power/area costs for multimedia applications need to find new emerged architectures. Ht-MPSOC architectures have been used in recent years as the promising solution for new multimedia applications. For these architectures, system performance improves as the number of custom instructions is increased. However, the integration of all the potential custom instructions as HW accelerators in these architectures would consume an excessive amount of hardware resources and dissipate a significant static power [111]. The purpose of our proposed hardware accelerators sharing methodology between processors is to reduce circuit complexity in terms of logic elements usage and energy dissipation while optimizing execution time. Our methodology is motivated by the fact that multimedia applications contain a large number of same frequently used kernels and separate private HW accelerators are used for different processors to provide the same computations.
HW accelerator sharing methodology consists on using a reduced number of HW accelerators for the same task executed on different processors. In fact, a traditional implementation of Ht-MPSoC with a common task executed on m different processors consist on coupling each processor to its private HW accelerator. For this example, the architecture uses m hardware accelerators to execute the same task on the m different processors. However, according to area-performance trade-off, different processors, of the m ones, could share a HW accelerator. Thus the number of implemented HW accelerators would be reduced. It is expected that an appropriate level of HW accelerator sharing will extenuate the area and power consumption and will preserve performance. For each task, the HW accelerators sharing is more significant as the number of processors executing this task is more important. Thereby, the identification of similarity between tasks executed on the different processors seems to us to be the key to releasing wider benefits of HW accelerators sharing methodology. For many multimedia MPSoC architectures, all the n processors execute the same computations on different data and thereby the same tasks. For such architecture, the HW accelerators sharing methodology emerges a new class of HT-MPSOC architecture, where all the processors have the same number and type of HW accelerators with the same sharing degree for the same type of HW accelerators. This architecture is named Symmetric Ht-MPSoC architectures (SHt-MPSC).
When the n processors execute different applications, different set of processors may have different set of similar tasks. For these architectures, the emergence of HW accelerators sharing methodology provide a second new class of Ht-MPSoC architecture where the different processors may have different number and type of HW accelerators with different sharing degree. This architecture is named Assymmetric Ht-MPSoC architectures (AHt-MPSC).

This chapter is organized as follows: in section 3.2, we present the hardware accelerators sharing methodology. After that a motivating example of the proposed sharing methodology is presented; the impact on area usage and performance gain is highlighted. In section 3.3, we present the common tasks used in multimedia applications. The usual use of these common tasks motivates the employment of HW accelerators sharing methodology for multimedia applications. In section 3.4, we present the hardware flow to migrate a c/c++ task to a HW accelerator. Section 3.5 describes the proposed SHt-MPSoC and AHt-MPSoC architectures. The interconnection network of the proposed architecture is detailed in section 3.5.3. Finally, we conclude the chapter in section 3.6.

3.2 Hardware Accelerators Sharing Methodology

For multimedia applications, a large number of computational tasks are candidate for instructions extension in embedded systems. Each instruction is implemented as a HW accelerator adding a substantial area usage. However, these tasks contain a range of similar computations. Instructions extension without exploring such similarity may bloat the available hardware resources without exploring all specific-instructions extensions and reaching the desired performance(See section 3.2.1).
In order to achieve maximum profit from the benefits of specific-instructions extensions, we propose a HW accelerator sharing methodology. The proposed sharing methodology enables to share a HW accelerator of a specific task between two or more processors executing this task.

For our proposed sharing methodology, we have to adopt some notions and definitions.

- Pattern : we call a pattern a computational task existing on one or different applications. A pattern computation ranges from one operation (addition, multiplication, ect) to a complex task. In figure 3.1, $T1$ and $T2$ are two patterns executed on different processors. $T3$ is a third pattern executed on only one processor.
- Private HW accelerator: We call a private HW accelerator, a HW accelerator which is coupled to only one processor. In figures 3.1.b and 3.1.c , $T3$ is a private HW accelerator for $P4$.
- Shared HW accelerator: We call a shared HW accelerator, a HW accelerator which is coupled to two or more processors. For example, in figure 3.1.c, $T1$ is a shared HW accelerator for $P1$, $P2$ and $P3$. A synchronization access is integrated within the HW accelerator in order to manage processors access.

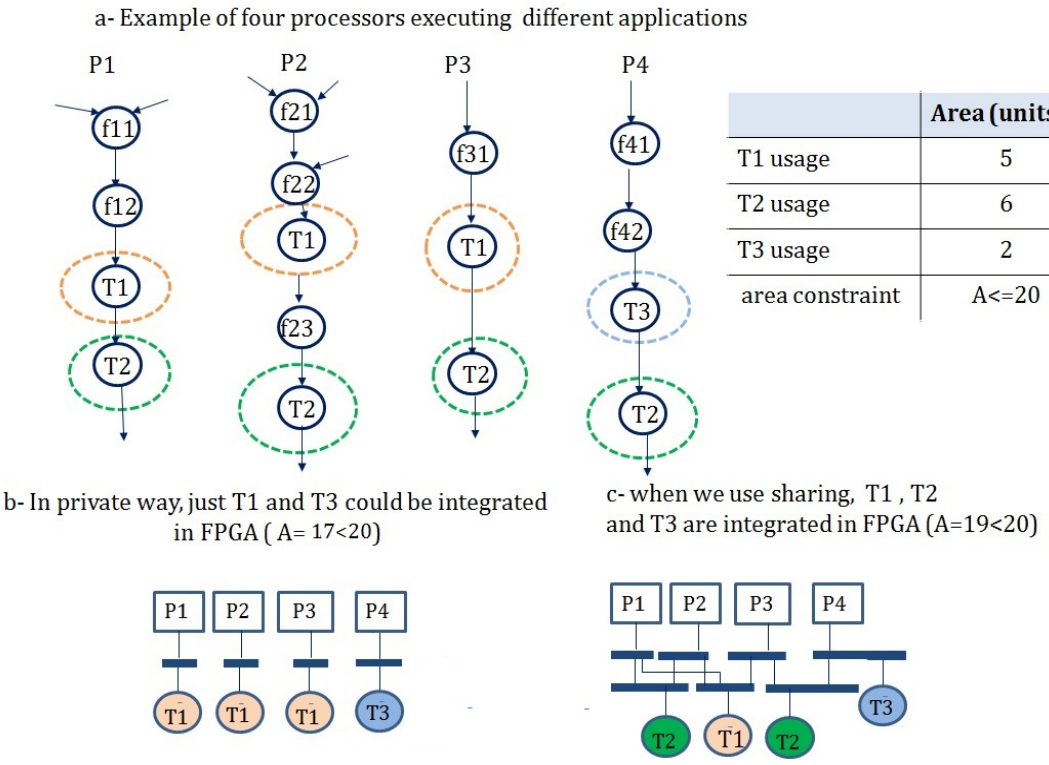

FIGURE 3.1: Illustrative example of benefits of HW accelerator sharing. $T1$, $T2$ and $T3$ are computational tasks executed on $P1$, $P2$, $P3$ and $P4$. The HW accelerator of Tj consumes aj area units in FPGA

- Sharing degree: we define the sharing degree of a HW accelerator, the number of processors that share this HW accelerator. In figure 3.1.c, $T1$ is shared between $P1$, $P2$ and $P3$ and thus its sharing degree is equal to three.
  In the following subsections, we show the benefits of our HW accelerator sharing methodology on area usage and its impact on performance.

3.2.1 Area Saving

Figure 3.1 shows a frequent situation of Ht-MPSoC architecture executing different multimedia applications. In this figure, $P1$, $P2$, $P3$ and $P4$ are four processors executing different applications. Each application contains a number of computational and non-computational tasks. In this example, computational tasks are highlighted with dotted outlines and we note that each computational task is executed by one or more processors. $T1$ is a common computational task executed by $P1$, $P2$ and $P3$. $T2$ is a common computational task executed by $P1$, $P2$, $P3$ and $P4$. $T3$ is a computational task executed by $P4$. For this example, specific instructions extension, if performed without taking advantages of common tasks, cannot be fully explored. As shown in Figure 3.1.a, $T2$ is executed by all the processors. A typical integration of this task

as HW accelerator for all the processors would consume 24 area units. While FPGA hardware resources are limited to 20 area units, $T2$ extension cannot be performed. In figure 3.1.b, only $T1$ and $T3$ are integrated as HW accelerator and consume 17 area units.

When considering HW accelerator sharing methodology, a reduced number of Hw accelerators for each computational task, existing in different applications, can be implemented and shared among the processors. Thus, it becomes more feasible to explore all specific-instructions extensions.
Based on our proposed approach, it would be possible to integrate all computational tasks as HW accelerator. Figure 3.1.c is a possible configuration with shared HW accelerator. For $T1$, one shared-accelerators is used to compute $T1$ instead of 3 private hardware accelerators. For this pattern, area units usage is reduced from 15 area units to 5 area units. For pattern $T2$, two shared-accelerators are used and shared between $P1$, $P2$, $P3$ and $P4$ processors. For $T2$, area-units usage is reduced from 24 units, required to integrate 4 private hardware accelerators, to 12 area units. Pattern $T3$ is executed only on processor $P4$, so it's integrated in private way and consumes 2 area units. The implementation of these HW accelerators requires 19 area units and thereby, the available HW resources could hold this configuration.

3.2.2 Impact of HW Accelerators Sharing Methodology on Performance

When a pattern T_j is executed on many processors, the hardware accelerators sharing methodology can be applied. However, the number and the set of processors that share a HW accelerator may improve or decrease the architecture performance.
Consider the example of figure 3.2, where T_2 is a computational task executed on four processors and can be executed in different ways. Figu 3.2.a shows the software execution of T_2 on the 4 processors. Figure 3.2.b shows the execution of T_2 with a shared configuration. In this configuration, P_1 and P_2 share a HW accelerator of T_2 and P_3 and P_4 share another HW accelerator of T_2. For this configuration, P_2 and P_4 has to wait until the end of the execution of T_1 on processor P_1 respectively on processor P_3. As a result, for P_2 and P_4, delays are created and performance of execution of T_2 on P_2 and P_4 are decreased when compared to the software execution (Figure3.2.a).
In Figure 3.2.c, the delay occurred in b is negated by changing the set of processors that share each HW accelerator. A HW accelerator of T_2 is shared between P_1 and P_3 and an other one between P_2 and P_4. Indeed, for this configuration, the intervals of T_2 execution on processors that share each HW accelerator don't overlap. This means, when each processor has to access the shared HW accelerator, it finds this latter free

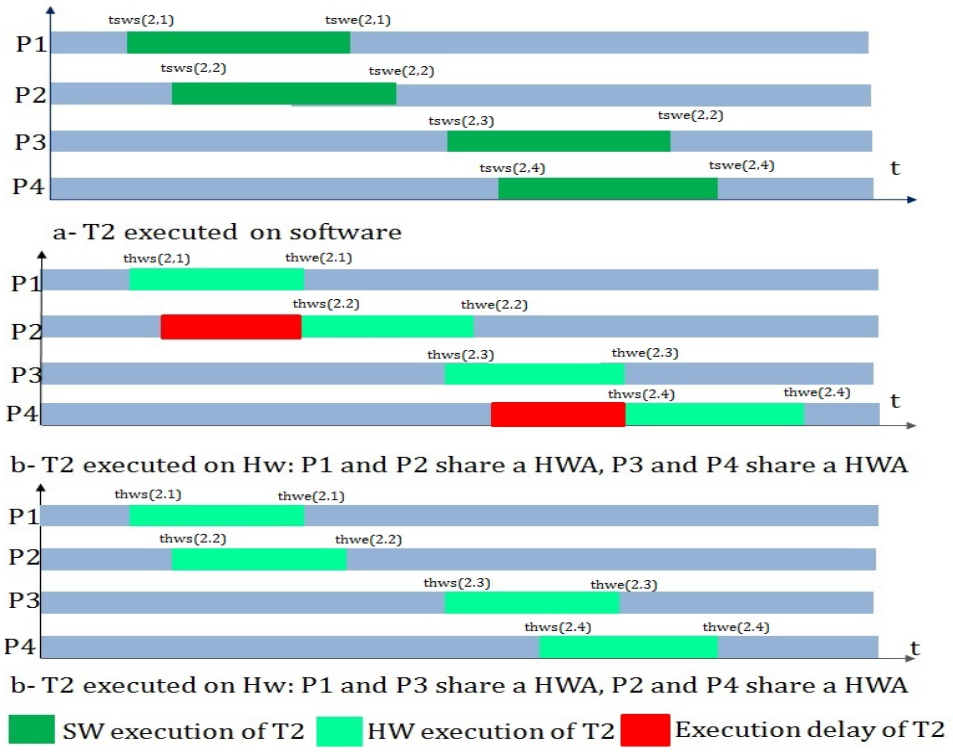

FIGURE 3.2: Example to illustrate the impact of HW accelerator sharing on performance

and executes T_2 without delay.

In this example, we show that hardware accelerators sharing methodology allows an intelligent exploitation of FPGA resources. However, the decisions of the number and the set of processors that share a HW accelerator affect the area-performance trade-off. Therefore, the design space of resource-sharing solutions has to be explored in order to find the optimal solution. Chapter4 presents an original heuristic in order to control the number and the set of processors that share a HW accelerator , thereby permitting the exploration of trade-offs between execution delay and area savings.

3.3 Common Multimedia Kernels

The evermore increasing of computational and communication requirements demanded by recent multimedia applications together with energy constraints are the key challenges to deliver an efficient multimedia device.

These applications are often complex and contain a range of tasks, each of which has to be performed under a real time requirement. For example, a face recognition application on iPhone 3G consumes 11 seconds and consumers may feel it is too slow [112]. In order to improve the performance of such applications, executing the computational tasks

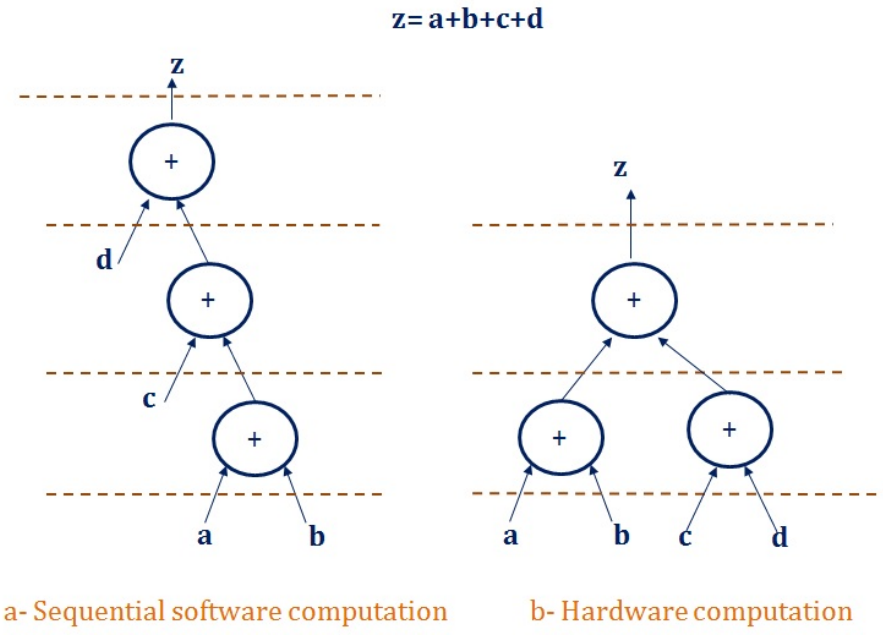

FIGURE 3.3: Illustrative example of parallel aspect of hardware execution

on HW accelerators is a common used technique. In this section, we present a range of time-consuming tasks frequently used in different multimedia applications. These common used tasks motivate our proposed HW accelerators sharing methodology. Our selection covers many types of multimedia applications: image compression, 3-D graphics, audio synthesis, audio compression, video compression. Table 3.1 lists a number of time consuming multimedia tasks commonly used in different multimedia applications.

3.4 Hardware Accelerators Flow Graph

A HW accelerator is a specialized hardware module, which executes a time-consuming task of computationally intensive software code. The HW accelerator is controlled by the software code that requests high performance and minimal power consumption. Performance improvements are reached thanks to the parallel execution of hardware (Figure 3.3). In this section, we describe the different steps of the HW accelerators implementation flow depicted in Figure 3.4. Within the HW accelerator flow, we consider the following steps:

- Pre-synthesis validation
- RTL Synthesis

TABLE 3.1: Examples of usually used multimedia patterns

Common multimedia tasks	Description	Example of applications
Matrix transpose	Common used matrix operation	2D media kernels: Image filtering, Shearsort, DCT, FFT, face recognition, ect
Vector/Matrix Multiply	Common used matrix operation	face recognition, DCT
Repetitive Padding	The pixel values at the boundary of the video object is replicated horizontally as well as vertically	H264, Mpeg-4
RGB-to-YCbCr/ YCbCr-to-RGB	Color space conversion from RGB colors to YCbCr brightness	jpeg encoder, mpeg-2 encoder, mpeg-4 encoder, H264 enoder
2D-DCT	A signal transformation from spatial domain to frequency domain with an elimination of redundant components	Jpeg encoder, H264 encoder, mpeg encoder, mp3 encoder
2D-IDCT	A signal transformation from frequency domain to spatial domain	Jpeg decoder, mpeg decoder, H264 decoder, mp3 decoder
Motion estimation	A video process of determining motion vectors that describe the transformation from one 2D image to another	mpeg-1,mpeg-2, mpeg-4, H264
FFT	A transformation of time (or space) to frequency (or wavenumber) and vice versa	MP3, MPEG-4, H.264

- Post Synthesis verification
- System assembly and logic synthesis

In order to generate the RTL description of the computational task, the RTL synthesis process needs a synthesizable VHDL or Verilog description of this pattern. This means that non synthesizable opertaions, like dynamic allocations, has to be replaced by their equivalent synthesizable operations . The RTL description is then passed to Post Synthesis verification to validate the behaviour of the hardware description. Once the RTL behaviour is verified, the hardware module is connected to the architecture as hardware

accelerator. Different modules and interconnections are added to mange processor and HW accelerators communication and the configuration file is then generated.

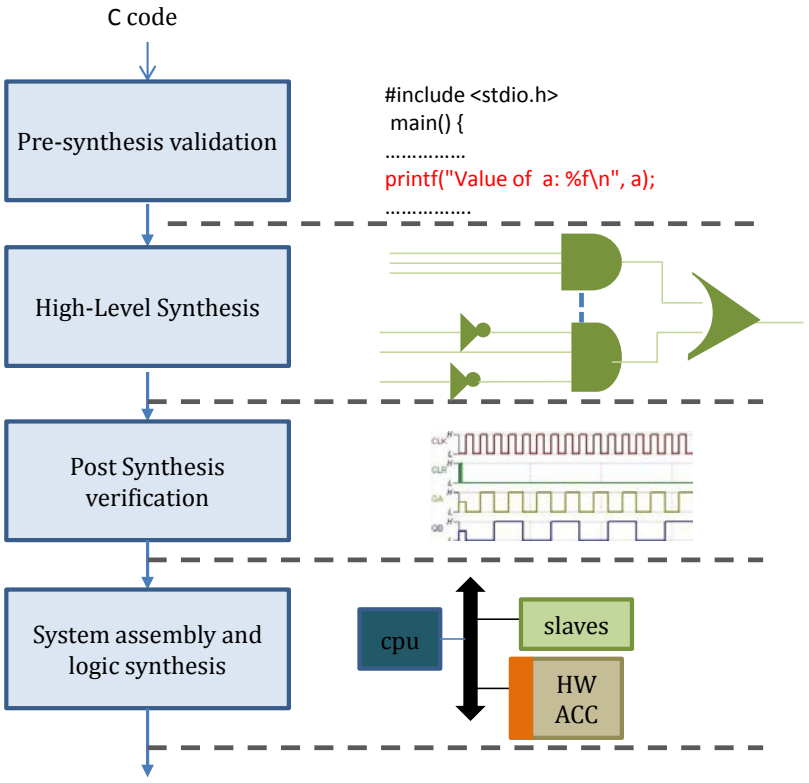

FIGURE 3.4: Design flow of hardware accelerator integration for FPGA-based architecture

3.4.1 Pre-synthesis Validation

The HW accelerator is coded using the VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL) programming language. The VHDL description provides the same computation of the C/C++ code of the computational task. Before we proceed to VHDL description of a computational task and its synthesis, the C code has to be self-checked. In fact not all the VHDL equivalent C/C++ functions can be synthesised. System calls and non-bounded size pointers are examples of non synthesizable C/C++ constructs. For this reason, based on [113] manual, we recommend an update of non synthesizable constructs when describing the VHDL behavioural . For example, VHDL language doesn't have the type float. However, we can define this data as std_logic_vector, then we can use IEEE-754 package. We enumerate in the following a list of non synthesizable data types and constructs.

3.4.1.1 Synthesizable and Non-synthesizable C/C++ Data Types

In C/C++, all ports, variables, signals, ect are declared with a data types. If a C/C++ code has to be converted to VHDL description, data types have to be modified to use synthesizable ones.

TABLE 3.2: Non Synthesizable C/C++ data types

C/C++ type	Equivalent in HDl
floating point	use of std_logic_vector and IEEE-754 package
pointers	access to an array
file type: file	replace or remove
I/O streams: stdout and cout	replace or remove

TABLE 3.3: Synthesizable C/C++ data types

C/C++ type	Description
bool	A single-bit true or false value
int, signed/unsigned int	A signed or unsigned integer, typically 32 or 64 bits
char, signed/unsigned char	8 bits, signed/unsigned character
struct	A user-defined aggregate of synthesizable data types
enum	A user-defined enumerated data type

The appropriate modifications do not affect the desired behaviour. In Table 3.2 and Table 3.3 , we illustrate examples of synthesizable data types and non synthesizable data types and their HDL equivalent.

3.4.1.2 Non-synthesizable C/C++ Constructs

To obtain a synthesizable hardware description, we have to avoid function calls to operating system, dynamic memory allocations, unconditional branching and run-time identification and casting. Table 3.4 summarizes essential non synthesizable C/C++ constructs.

3.4.2 RTL Synthesis

The RTL synthesis process consists on generating the RTL design of the hardware accelerator, from the HDL description. While we are targeting Xilinx FPGA, the design of HW accelerators is generated using Xilinx tools. The EDK software, takes the HDL description of the designed computational task and generates the RTL description of the HW accelerator.
In order to interconnect the HW accelerator and the processor, the former is synthesised

TABLE 3.4: Non synthesizable C/C++ constructs

Category	Construct	Action
Dynamic storage allocation	malloc(), free(), new()	Use static memory allocation
Exception handling	try, catch	Comment out.
Operator, sizeof	size of	Determine size statically
structure type	union	Replace with struct
Dereference operator	* and & operators	Replace with array accessing
Unconditional branching	go to	Replace

with a bus interface and registers or local memory which are mapped into its address space. The following subsections describe the synthesised communication interface and the memory space allocation.

3.4.2.1 Communication Interface

From a communications point of view, a Hw accelerator is a black box controlled by data arrival which receives and sends data possibly at each clock cycle. It has a number of input ports and output ports each of which having a certain bit-width. In addition to the clock, the HW accelerator has a clock enable pin that can freeze its execution. Hence, if the clock enable is not set, everything behave in the HW accelerator as if the clock was not changing. Thus, the HW accelerator is data synchronised, i.e. at each clock cycle, data are presented on the input port and at the raise of the clock (provided that the clock enable is set), the data is read by the Hw accelerator. If all the required data are present and the clock enable is set then the HW can run for a cycle.
As shown in Figure 3.5, in [114], authors classify the processor-accelerators interconnection modes into three categories : Processor driven, external DMA engine and Internal DMA engine. It can be envisaged the use of the DMA in can be made much more efficient if the DMA is designed specifically architecture interface and is directly connected to the HW accelerator as shown on Figure 3.5.c. The interface mechanism is used in various SoC architectures and dependent on the targeted platform. In this thesis, we used XILINX platforms.

A XILINX designed architecture is based on PLB or OPB bus, so a designed HW accelerator must be a PLB or OPB compliant peripheral. This compliance is provided in the synthesis process with an IPIF interface. Figure 3.7 shows the top level of a HW accelerator. In this figure the top level is constituted of the VHDL user logic, which describe the HW accelerator functionality, and the IPIF inteface.

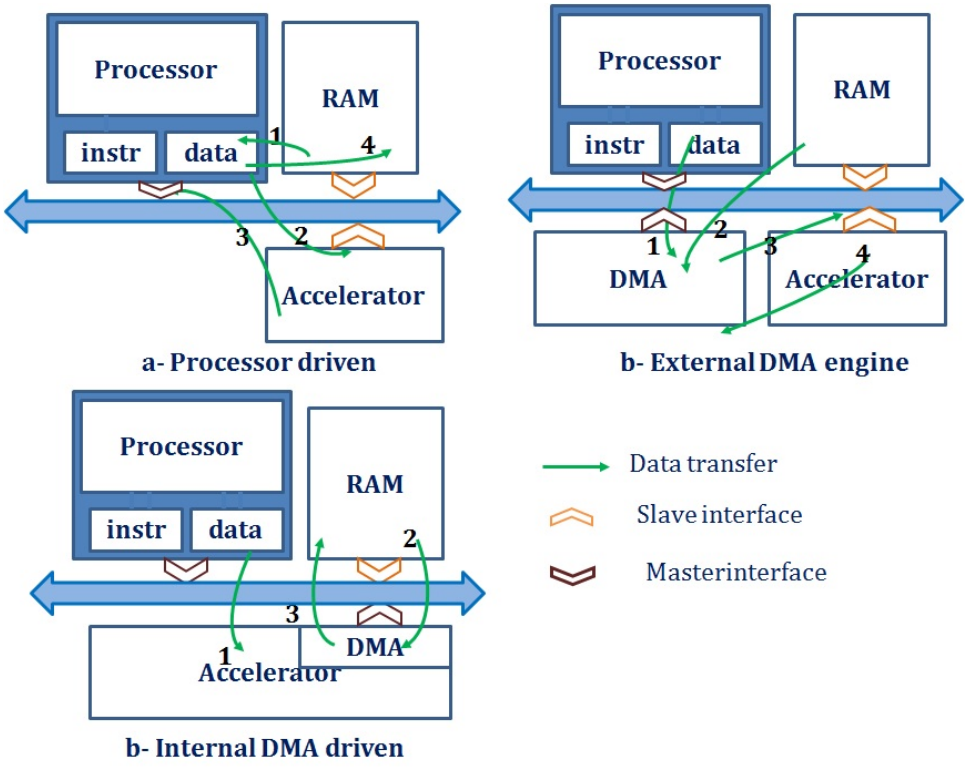

FIGURE 3.5: Interface modes for processor-accelerator interconnection

All designed HW accelerators within Xilinx tool incorporate the Intellectual Property Interface (IPIF). The IPIF module provides three types of IPIF interconnection : OPB bus, PLB version 3.4 and PLB version 4.6. One side of this interface implements the PLB or OPB interface, and the other side implements the Intellectual-Property Interconnect (IPIC) interface. The IPIF provides basic features, such as address decoding, slave attachment, and byte steering [115]. In addition, it provides some optional features that can greatly simplify the task of creating the HW accelerator, either through parameterization of the corresponding IPIF component or direct instantiation of other IP library components. Based on selected functionalities, Xilinx tool automatically creates corresponding OPB or PLB peripheral templates with slave-only operation or master-slave combined operations.

3.4.2.2 Memory space allocation

In order to communicate data between processor and a designed HW accelerator, the top level of the HW accelerator has user registers or memory space addressable through software. The processor issues data to the HW accelerator by sending store instructions to addresses of accessible registers or to addresses within the range, and restores data from a HW accelerator by sending (load instruction from addresses. In case of interconnection through addressable memory space or user register, the size of the address

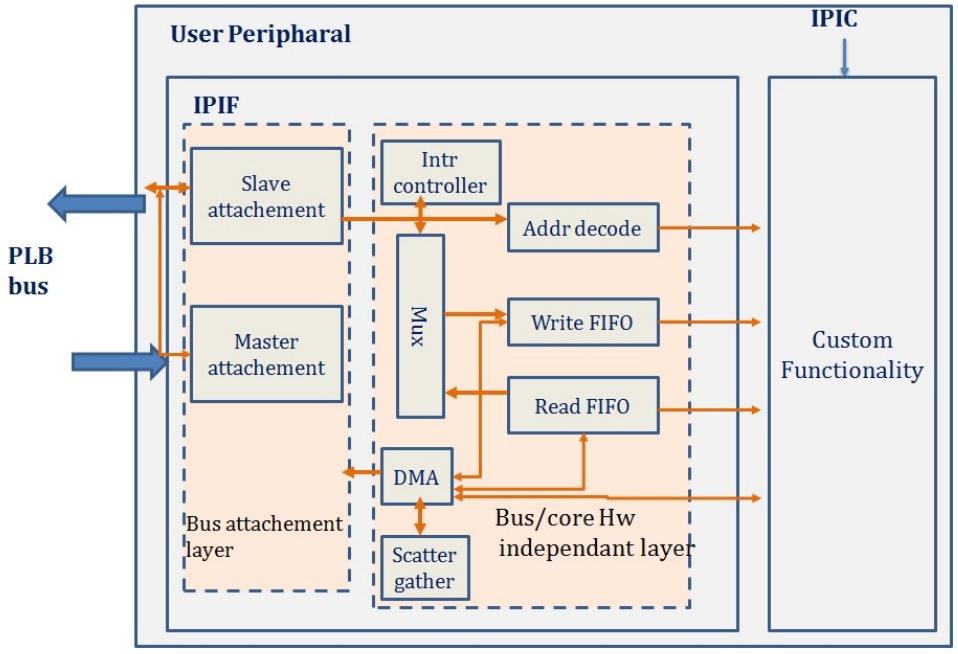

FIGURE 3.6: Communication interface between the processor and the HW accelerator

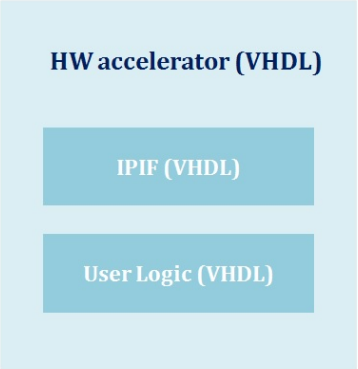

FIGURE 3.7: Top level of PLB-based hardware ACC

range depends on the bit width of processor data bus as well as the width of the Hw accelerator address signal. This address range is calculated as follows:

$$memory\_range = base\_address + \alpha \tag{3.1}$$

$$\alpha = 2^{hw\_addr\_width} * \frac{data\_bus\_width}{8} - 1 \tag{3.2}$$

In Equation 3.1, base address denotes the starting address of the HW accelerator and base $address + \alpha$ indicates its ending address. A bus width is expressed in terms of bits and the memory range is expressed in bytes. For example if the processor bus width is 32 and the HW accelerator has a 3-bit address bus width, value of α will be equal to 0x0000001F.

When a processor needs to send data to a HW accelerator, it activates a write enable signal and sends the address on which data will be stored. This address is decoded within the PLB interconnect and the offset α is transmitted to the IPIF unit. This offset is decoded within the address decode unit and controls how to send data to the functional unit of the HW accelerator.

3.4.3 Post-synthesis Simulation

Post-synthesis simulation of the produced RTL design is performed by means of the co-simulation feature of Xilinx development tool. It uses a HDL test bench which will aid in debugging the design of the HW accelerator before implementation to the FPGA for execution.

3.4.4 System Assembly And Logic Synthesis

The system assembly is performed within the XPS software of XILINX development tools. It is used to configure and connect the HW accelerator to the system architecture.
When the RTL design is successfully verified by the post-synthesis step, the HW accelerator can be exported as an IP, to be connected to the architecture. Once the HW accelerator is connected appropriately, a bitfile of the system architecture is generated and the design can be exported in SDK software of XILINX development tool.

3.5 HT-MPSoC Architecture With Shared HW Accelerators

Processing acceleration in one side and multiprocessing using several cores in the other side are two beneficial paradigms. The combination of these two paradigms in the same architecture offers a sustained performance and could be very efficient for parallel applications involving hot computational kernels like image and video processing applications. These architectures are named Ht-MPSoC architectures. Processing acceleration is provided by dedicated hardware components and corresponding custom instructions.

Moreover, the increase in HW resources in the latest FPGA generation, makes it possible to implement complex Ht-MPSoC architectures. These architectures combine hardware and/or software cores, application-specific HW accelerators and communication units.

The Xilinx Zynq 7000 Extensible Processing Platform (EPP) is an example of such architectures embedding a dual core ARM Cortex A9 processor and tens of thousands of programmable gate arrays [116]. Cyclone V from Altera [117] and SmartFusion2 from Micro-Semi [118] are other examples of FPGA intended to prototype complex Ht-MPSoC. These architectures include one or more hardcores and up to 500K of reconfigurable logic elements to build computational accelerators.

In our thesis, in the proposed Ht-MPSoC architecture, the system consists of multiple processors running software tasks and a group of HW accelerators that execute application specific instructions. The number and the sharing type of HW accelerators can vary from processor to another. The purpose of sharing HW accelerators between processors is to reduce circuit complexity in terms of logic elements while maintaining the performance and reducing the energy consumption.
In this thesis, with the use of hardware accelerators sharing methodology for Ht-MPSOC, we propose two new classes of Ht-MPSoC architecture. The first class is a Symmetric Ht-MPSoC(SHt-MPSoC), in which all the processors have the same number of private and shared HW accelerators. The second class is an Asymmetric architectures(AHt-MPSoC), where HW accelerators attached to the different processors differ from one processor to the other.

3.5.1 Symmetric Ht-MPSoC Architecture

SHt-MPSoC is the architecture where two or more homogeneous processors run the same application or the same set of tasks. All the processors share a main memory, used for data communication. The term symmetric for this class of architecture, lies in the fact that all the processors execute the same set of tasks and they should have the same performance gain, so all the processors should have the same number and type of hardware accelerators. A replication of a hardware accelerator of a computational pattern is an excessive area-consuming solution. The emergence of hardware accelerators sharing methodology for these architectures would moderate the area usage.
Figure 3.8 is an example of 4-processors SHt-MPSoC architecture. All the processors share a main memory and each processor is connected to peripherals over its local bus. In this example, each processor has a private HW accelerators and a shared one. The shared HW accelerator are connected to processors through an interconnection network.

3.5.2 Assymmetric Ht-MPSoC Architecture

AHt-MPSoC is the architecture where two or more homogeneous processors run different applications or different set of tasks. Each processor has a local memory and a

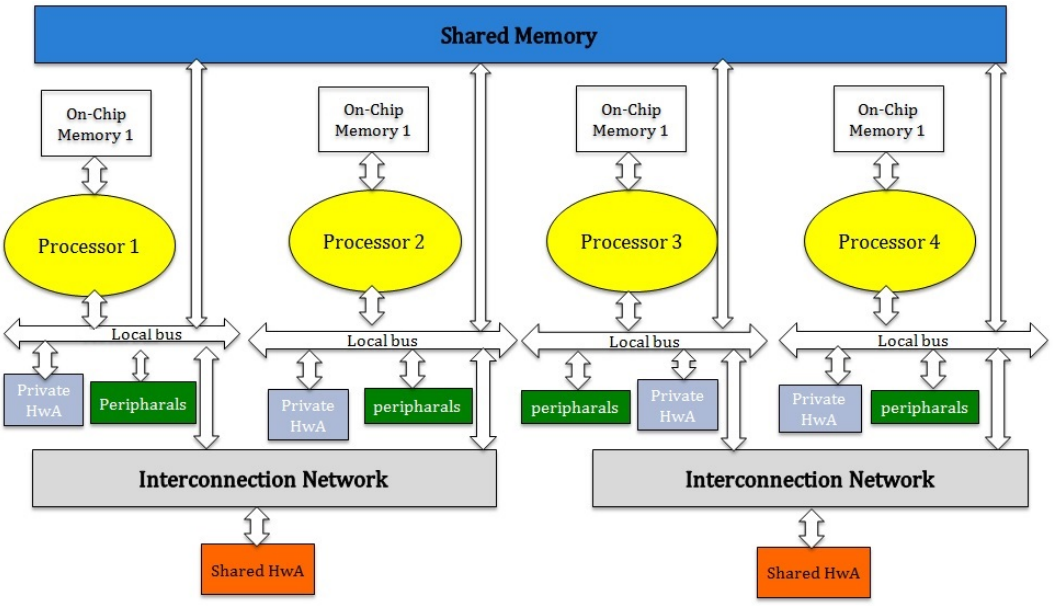

FIGURE 3.8: Example of SHt-MPSoC architecture

local bus. The local bus connects the peripherals and private Hw accelerators to the processor. The Hw accelerator of a common task, executed on two or more processors, can be shared between these processors. An interconnection network is implemented to share the HW accelerators of these common tasks. The number of private and shared hardware accelerators for each processor depends on the performance requirement for each application.

Figure 3.9 is an example of 4-processors AHt-MPSoC architecture. Each processor has a local memory and a private bus. All the processors share a main memory and each processor is connected to peripherals over its local bus. In this example, each processor has a number of private and shared HW accelerators. In this example, Processor 1 and Processor 2 have a common task, thus share its HW accelerator. Likewise, Processor 3 and Processor 4 have two common tasks and they share their HW accelerators. For each processor, the private HW accelerators are placed on its local bus whereas, the access to a shared one is assured through an interconnection network.

3.5.3 Hardware Interconnection For A Shared Hardware Accelerator

In this section, we present how the interconnection network of Figures 3.8 and 3.9 are constructed. The interconnection network on these figures are responsible of data exchange between processors and shared HW accelerators.

There are several ways to construct an on chip interconnection network that acts as a shared bus [119]. For our proposed SHt-MPSOC and AHt-MPSoC architectures, we used an hierarchical bus which is based on several buses interconnected through bridges.

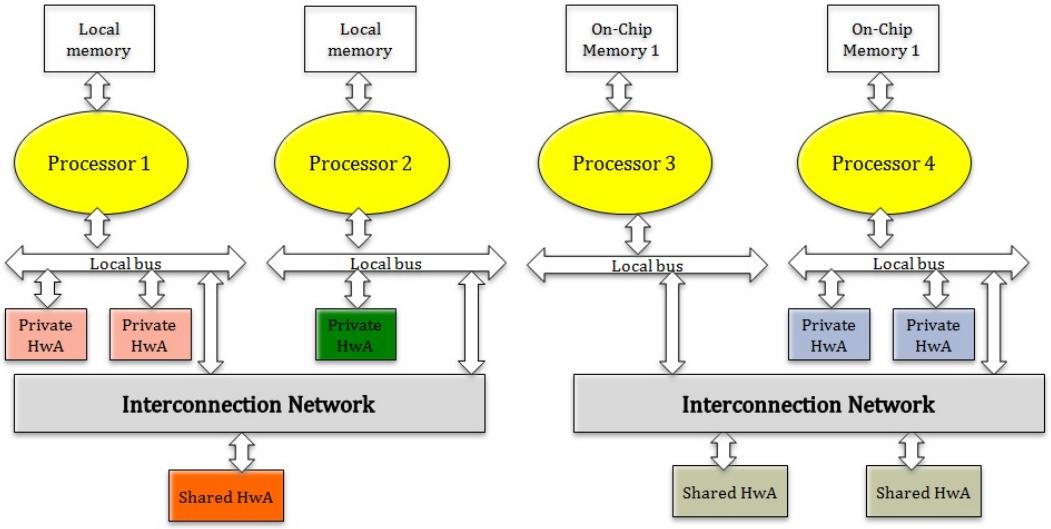

FIGURE 3.9: Example of AHt-MPSoC architecture

Figure 3.10 illustrates a basic architecture of two level hierarchical bus. The primary advantages of an hierarchical bus are the practicality and the performance/energy optimizations [119][120].

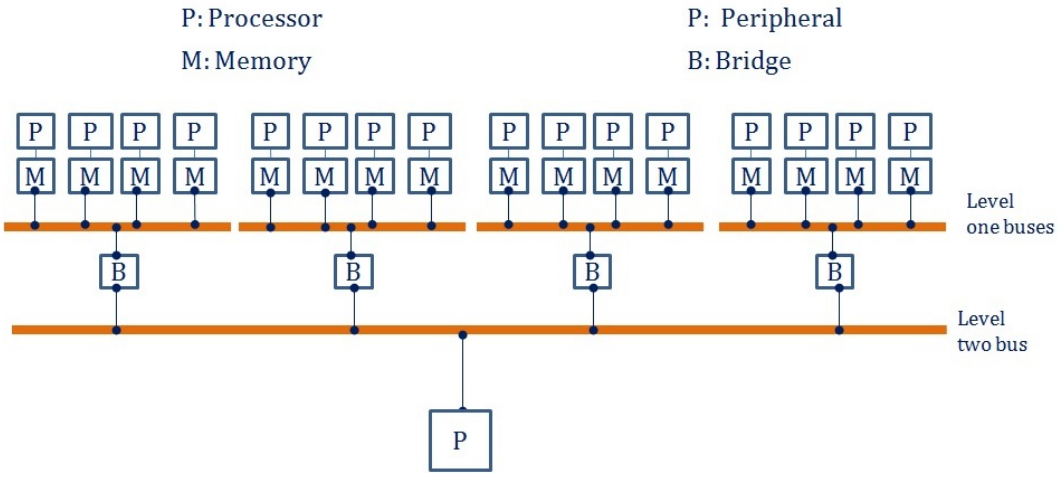

FIGURE 3.10: Interconnection using a two-level bus hierarchy

For SHt-MPSoC and AHt-MPSoC architectures, the level one of buses is constructed of processors local buses. Each processor has its private local bus to interconnect peripherals and private HW accelerators (Figures 3.8 and 3.9). The second level of buses is comprised of a number of buses to interconnect each instance of shared HW accelerator to processors that share this latter. Each shared HW accelerator is placed as a peripheral on a private bus and for each processor accessing this HW accelerator, a bridge is implemented as a master on the HW accelerator bus.

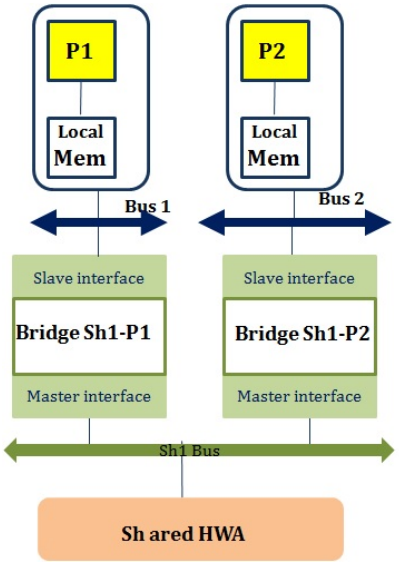

FIGURE 3.11: Example of hierarchical buses for shared HW accelerator interconnection

The bridge passes Figure 3.11 is an example of implemented architecture with two-level shared Hw accelerator. In this figure *Sh1 bus* is the local bus of the shared HW accelerator. Bridge are implemented between the HW accelerator bus and P1 and P2 processors buses. In section 3.5.4, we present the architecture of plb to plb bridge as well as its functioning.

3.5.4 Description of PLB-to-PLB Bridge

Since we are interested in Xilinx FPGA, we present in this section the PLBv46 to PLBv46 bridges [121]. The primary function of the PLBv46 to PLBv46 is the transactions passing from the primary PLB to the secondary PLB. The primary bus is the one that is closer to the processor. The secondary bus is the one that is farther away from the processor. For our proposed architectures, when a HW accelerator is shared between two or more processors, each processor bus is considered as a primary bus and the HW accelerator bus is the secondary bus (Figure 3.11).
The bridge operates as a slave on the primary PLB and as a master on the secondary PLB. When a processor passes a transaction to the shared HW accelerator, the transaction is received by PLBv46 Slave and decoded in the primary interface of the bridge. Then, the secondary interface logic generates the sequence of PLB signals to perform the transaction on the HW accelerator. Figure 3.12 presents the block diagram of PLBv46 to PLBv46 Bridge. The principals blocks of the PLBv46 to PLBv46 Bridge are as follows

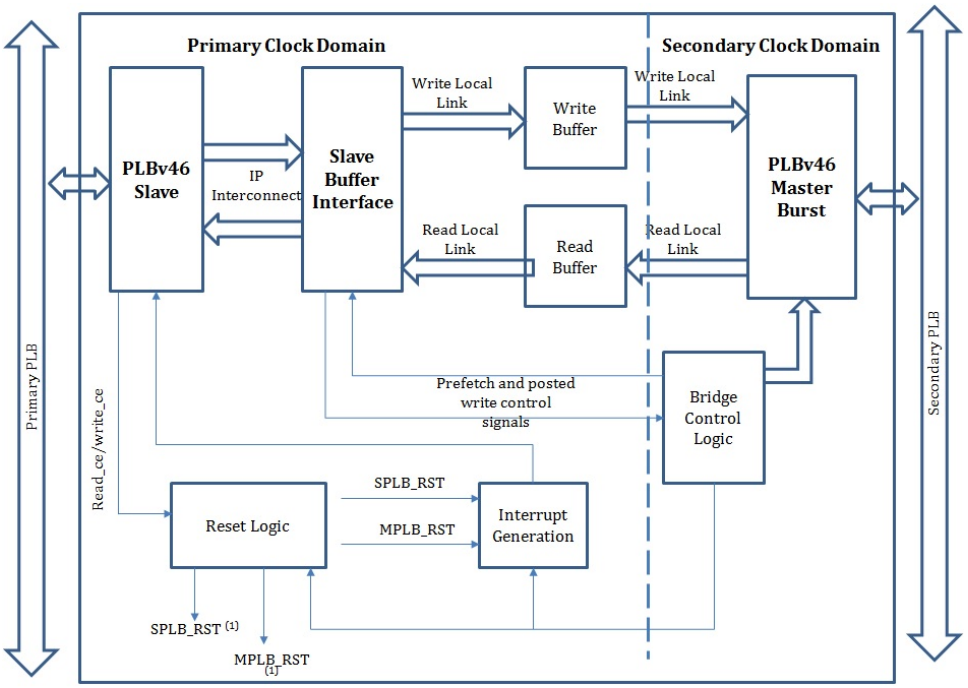

FIGURE 3.12: Block Diagram for the PLBv46 to PLBv46 Bridge

1. The PLBv46 Slave: it provides a bi-directional slave interface to the processor bus. The PLBv46 Slave provides the address decoding for the registers inside the bridge and for the HW accelerators on the secondary PLB.

2. Slave Buffer Interface: it provides the conditional read/write access to the PLBv46 Slave. This block decodes the request from the hw accelerator and passes the request to the processor bus through Control Logic and the data passes through the Write Buffer. This block also have a Xilinx Local Link Interface to communicate with the read buffer and write buffer.

3. Write Buffer: it is a FIFO memory that stores the data from the Slave Buffer Interface during a write transaction.

4. Read Buffer: it is a FIFO memory that stores the data from the PLBv46 Master Burst during a read transaction.

5. PLBv46 Master Burst: it provides the PLB master interface on the HW accelerator bus operations.

3.6 Conclusion

This chapter presents a new HW accelerators sharing methodology for Ht-MPSoC architectures. This methodology enables two or more processors to share hardware accelerators of their similar tasks. The purpose is to explore the available hardware resources in an area intelligent manner. This chapter presented novel classes of Ht-MPSoC architecture on which shared HW accelerators are used to ensure the area-performance trade-off. The shared HW accelerator is interconnected to correspondent processors through two-level hierarchical bus.

We showed through motivating examples that the proposed methodology saves area but may degrade the performance. The increase of numbers of processors that share a HW accelerator save hardware resources but may create significant execution delay. Therefore, in order to provide efficient architecture, we have to explore all configurations between the fully shared one and fully private one.

In the next Chapter, we present a technique to provide the designer a fast solution to find out the configurations of HW accelerators for a given Ht-MPSoC architecture.

Chapter 4

Design Space Exploration in Shared Hardware Accelerators Based Ht-MPSoC

Contents

4.1 Introduction

Hardware Accelerators sharing methodology for Ht-MPSoC architectures aims to provide a smart exploitation of available HW resources to integrate HW accelerators. As more computational tasks are emerged as HW accelerators, the achieved speedup and energy consumption are improved. As explained in Chapter 3, our space exploration is motivated by concurrent aspect of recent mulimedia applications, which have a high potential of HW accelerators customization. For each computational task, we need to determine a) if the computational task will be customized or executed on software and b) the sharing-degree of the HW accelerators of each computational task if it will be customized. The HW accelerators customization and the HW accelerators sharing degree results in a large design space. The design space of HW accelerators sharing is bounded by a fully private solution and a fully shared solution. The fully private solution consumes an excessive number of HW resources than fully shared configuration but preserves maximum speed-up. In contrast, a fully shared Ht-MPSoC configuration consume much less HW resources but it might degrade performance. Between these two extreme solutions, intermediate solutions may provide a best area-performance trade-off. To find one of these solutions, several exact methods or heuristic algorithms are used in the literature depending on the complexity of the problem; Integer Linear Programming (ILP) is one of the earliest exact methods to be used for optimization problems in embedded systems. The ILP formulation is used as it provides an exact solution of the problem. In this chapter, we propose a (Mixed ILP) MILP-based technique that integrates the hardware accelerators sharing methodology to explore the search space of shared and private configurations for Ht-MPSoC architectures. For this purpose, we distinguish two situations: Ht-MPSoC architectures where the different processors execute the same application and situation where the different processors execute different applications. For each architecture, we propose a MILP formulation of the design space of HW accelerators, that emerge the computation of selected application-specific instructions, to find out the best area-performance architecture. The first formulation considers only symmetric configurations while the second one deals with asymmetric and asymmetric configurations. Both MILP models take into consideration a delay parameter to control the impact of HW accelerators sharing degree on performance. Analytical equations, that consider the delay parameter, have been proposed to estimate the gain on execution time. The execution time gain is imposed as a performance-constraint to the objective function that minimize the usage of HW resources.

This chapter is organized as follows. Section 4.2 presents the proposed technique to extend Ht-MPSoC architectures with Hw accelerators. In section 4.3, we present the MILP formulation to explore the configurations of Ht-MPSoC architecture where all the

processors execute the same application. Section 4.4 details the MILP-based formulation of the space of configurations of a Ht-MPSoC architecture where processors execute different applications. Section 4.5 concludes this chapter.

4.2 Proposed Technique for an Area-Performance Trade-off

Most of existing embedded applications, such as multimedia, telecommunication or automotive applications, use the same set of critical tasks. Matrix operations, convolutions and filters are frequently tasks for these applications. For such embedded systems, where different processors execute the same set of computational tasks, the use of HW accelerators sharing approach is beneficial as it avoids bloating the FPGA resources with large number of HW accelerators. However, an excessive level of sharing can degrade performance. In fact, as the sharing degree increases, the delay to access the shared HW accelerator may increase. Depending on the sharing degree and the processors that share the same HW accelerator, the latency may improve or decline performance improvement. So, the space of configurations of a common computational task is bounded by a fully private solution and a fully shared solution. Between these two solutions, each configuration presents an area/performance trade-off. Thus, the complexity to explore this space of configuration increases as the number of common computational tasks increases.
This thesis proposes a technique, presented in Figure4.1, that represents a solution for designers to find out the efficient way to execute the computational tasks of applications executed on the different processors of a Ht-MPSoC architecture. This technique is based on a MILP formulations which identify, through the solution found, how to implement each task. This allows designers to come up, in a short time, with optimal configuration for a Ht-MPSoC architecture where processors execute identical or different applications. For a given set of processors and computational tasks executed on these processors, the generated solution is a Ht-MPSoC configuration that optimises the area usage and satisfies the required performance. This is performed via estimating the area usage and performance gain of different possible configurations of the space of solutions in order to find the optimal one. Depending on the performance/area consumption trade off, each computational task can be executed in software or in shared and/or private HW accelerators.

To reduce the time to search the optimal solution, the proposed technique is based on an iterative approach. Based on profiling results, our technique increases iteratively the number of explored tasks. Such process stops when the space exploration process

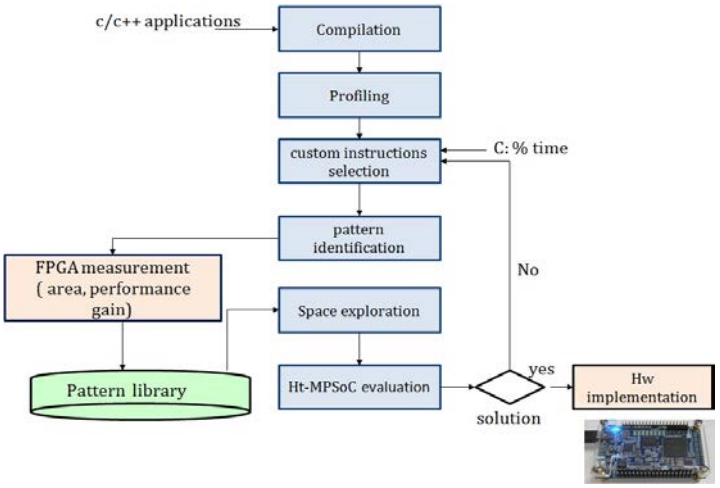

FIGURE 4.1: Proposed technique to extend an AHt-MPSoC with HW accelerators in order to speedup an application with an optimized area usage

generates a solution or when all tasks are explored. Thus, the generated solution is a local optimum MPSoC configuration that satisfies the required performance. In the following subsections, we detail the different process of the proposed technique.

4.2.1 Applications profiling and computational Tasks (CT) identification

Our proposed technique starts with compiling and profiling the architecture applications. Application profiling is an important step since it determines the most computational applications tasks. Embedded System designers are provided with different CAD profiling tools. These profiling tools are classified into three main categories: software-based, HW-based and FPGA-based tools [122][123][124]. For FPGA-based embedded systems, FPGA-based profiling (FPGA-BP) tools have proved better results compared to the other profiling tools [122] [123]. Thereby for our work we use (FPGA-BP) tools to compile and profile applications.

To select the computational tasks to be candidate for customization and the configuration of their HW accelerators, our technique is based on an iterative approach. Based on profiling results of the different applications running on the different processors, the most computational tasks are selected iteratively until the *space exploration* process generates a solution. In each iteration, a task is considered as computational if it consumes more than C% of the overall application execution time. For the first iteration, we compare the highest profiling percentage values of tasks from one application to another. The least value will be considered as the initial value of C. Each new iteration decreases the value of C to the minimum percentage of execution of the next less computational tasks. Thus, for each iteration, at least one more computational task is included. Thereby,

each iteration adds more computational tasks to be explored together with the previous ones until the *space exploration* process generates a feasible solution.

Figure 4.2 is an example of profiling summary of three different applications. Each application is constituted of different functions, which are sorted by percent of total execution time. For this example, the value of C is set to 36. In fact, the largest percentage of execution for application 1, application 2 and application 3 are respectively 50, 36 and 40. The value of C is equal to the minimum value of the three and thereby is set to 36. Each function of Figure 4.2 that consume over C% is selected as computational task. F1, F11, F21 and F22 are selected as initial computational tasks. After a first exploration, and if the tool cannot generate a solution, the value of C is decreased to the minimum value between 30, 35 and 38 which is equal to 30 . Thus F2 and F12 are new computational tasks that may provide additional feasible solutions.

	%time
F1	50%
F2	30%
F3	10%

a-Profiling results of application 1

	%time
F11	36%
F12	35%
F13	22%

b- Profiling results of application 2

	%time
F21	40%
F22	38%
F23	12%

c- Profiling results of application 3

FIGURE 4.2: Example of profiling results of three different applications executed on a Ht-MPSoC architecture

4.2.2 Pattern Identifications

The pattern identification step of Figure 4.1 consists on analysing the identified computational tasks of current iteration to look for similarities with the previous defined patterns and/or to add new patterns. We assume that different computational tasks are similar if they mainly have the same goal. When a set of similar tasks are identified, a generic superordinate pattern is defined.
The pattern library is updated in each iteration to include information of new patterns and/or to update information of existing patterns re-identified in new computational tasks.

In order to update the pattern library with pattern informations, the new identified patterns are described in VHDL language and synthesised and then connected to a processor as HW accelerators. The information of an identified pattern include the processors on which this pattern is executed, performance gain and area consumption if it is implemented on HW accelerator, its software start and end execution times on

the different processors. Information on execution time of the different patterns on the different processors are also obtained by profiling the application and executing it on the correspondent processor. The performance gain is expressed in term of clock cycles gain. The area usage depends on functional modules that constitute the hardware description of the pattern (fabric slice, DSP block, ect). The identification of tasks similarity is carried out manually. This is due to lack of efficient technique for sharing of HW resources for coarse grained application-specific instructions. Such techniques are developed for HW resource sharing of fine grained custom instructions, such as those proposed in [93] [125]. Recent works have developed these technique for the identification of similarity for loop computations [126].

4.2.3 Space Exploration

The main goal of the space exploration process is to find the optimal configuration of a given Ht-MPSoC architecture. This configuration is the one that minimizes the area consumption and, at the same time, satisfies a required performance. More precisely, we propose MILP formulations (see Sections 1.3 and 1.4) in order to solve the problem of customizing the patterns of a given Ht-MPSoC architecture.

Our technique, which is based on MILP formulations, explores iteratively the space of configurations of the patterns stored in the pattern library and stops when it generates the first feasible solution. Since the exploration process is not exhaustive, due to the lack of time to explore all the solution space, the generated solution is a local optimum Ht-MPSoC configuration.

The outputs of space exploration process are decision variables which determine the configuration of each HW accelerator. Once the optimal configuration is generated, the designer can identify if a pattern would be integrated as HW accelerator, and the sharing degree of each HW accelerator. If the model exploration is unable to find a solution, the designer has to decrease the C parameter to increase the number of explored patterns. This step is repeated until the model generates a feasible solution.

4.3 Space exploration in SHt-MPSoC

The aim of this section is to present the space exploration tool to optimise SHt-MPSoC architectures. In section 3.5.1 of previous chapter, an SHt-MPSoC architecture is defined as a Ht-MPSoC architecture where n processors have the same number and type of HW

accelerators. In Ht-MPSoC, when a pattern is added to the pattern library, the space of configuration of its corresponding HW accelerator is bounded by the fully shared and the fully private configurations. For each pattern, the fully shared solution implements only one shared HW accelerator between the n processors. This solution will minimize the added area cost but it might degrade dramatically the performance. In contrast, a fully private solution considers m replications of the HW accelerator. This solution will provide the maximum performance gain but will consume an excessive area usage which is due to the replication of the m HW accelerators. Between these two extremal solutions, there is a wide range of configurations each of which offers a trade-off between performance gain and area cost. This design space exploration has to be carried out by considering only the symmetric solutions, that are likely to share Hw accelerators with same performance constraint on all processors. Thereby the selection process is guided with symmetric configurations consideration.
In the following sections, we present and comment the MILP model proposed to explore the design space of a given SHt-MPSoC architecture. In section 1.3.1, we discuss the input and output of the MILP model corresponding to an instance of the studied problem. Mainly we will precise the data and the variables used in the MILP model in order to produce an optimal solution. Section 1.3.2 is dedicated to the formulation of the area cost of an SHt-MPSoC architecture. The area cost represents the objective function of our MILP model. In other word, in this problem we aim to find a solution (i.e. an architecture) with the minimum value of the area cost. The following section presents the formulation of the performance gain of the SHt-MPSoC architecture. This performance will constitute the constraints of our MILP model

4.3.1 Problem formulation

The architecture is a multi-processor system with n processors running the same applications, i.e. Single Program Multiple Data model. In a typical data parallel architecture, the number of processors is a power of two ($n = 2^i, i \in \mathbb{N}$). Let ($T_1...T_i...T_p$) denotes the sequence of p computational pattern executed on the n processors. For SHt-MPSoC architecture, all the processors have the same number and type of HW accelerators. Thus the performance gain which is provided by the customization process must be identical for all processors.

Let's consider the pattern T_k, $k \in \{1..p\}$, with its n occurrences in the n parallel applications. We denote (C^0..,C^j,..,C^m) the set of the possible configurations of T_k, where $m = \frac{\log n}{\log 2}$. The configuration C^j corresponds to the configuration with 2^j shared HW accelerators. The sharing degree of these HW accelerators is equal to 2^j. Thus, C^0 corresponds to MPSOC configuration with only one HW accelerator shared between the

n processors. C^m corresponds to MPSOC configuration with n private HW accelerators ($2^m = n$). The sharing degree of each configuration is equal to $\frac{n}{2^j}$. Depending on area-performance trade-off, T_i can be implemented in $m + 1$ different ways.

Let A_k be the required area on the FPGA to implement T_k. For the configurations with shared accelerators between cores ($j \in \{0..m-1\}$), an additional area denoted *delta*area added to the total consumed area. This is a predefined value that corresponds to the logic area consumed by the bridges needed to connect shared HW accelerators to their processors. For these configurations, regardless of the number of HW accelerators, each processor is connected to a shared accelerator through a bridge. Thus, for all shared configurations of n-processors architecture, the $\delta area$ is a constant value which is equal to n times the area usage of a bridge.

To evaluate the architecture performance we define different parameters to calculate the performance gain for each processor. These parameters are defined as follows :

- $T0_k$: denotes the software execution time of the pattern T_k on the n homogeneous processors.
- td : denotes the delay of execution between two processors. This delay is due to access to the data inputs stored in shared main memory . The sequential access to this shared memory causes the start of run of each processor to lag slightly behind the number of processors in the architecture.
- $tacc_k$: the execution-time-reduction obtained when implementing T_k ($1 \leq i \leq n$) on HW accelerator.

4.3.2 Objective function

In this section, we describe the area optimization problem to implement computational patterns on the data-parallel MPSoC. Our design space has to explore the software or HW execution of each pattern and the configuration of each pattern implemented on Hw accelerators. These choices can be provided with two decisions variables:

- x_k a binary variable set to 1 if the pattern T_k is chosen to be customized.
  $\forall k \in \{1..p\}$
  $$x_k = \begin{cases} 1, \text{ if } T_k \text{ is implemented on HW accelerator} \\ 0, \text{ otherwise} \end{cases}$$
- y_k^j a binary variable set to 1 if the pattern T_k is implemented on HW accelerator following configuration C^j.

$\forall k \in \{1..p\}, \forall j \in \{0..m\}$

$$y_k^j = \begin{cases} 1, \text{ if } T_k \text{ is implemented on HW with } C^j \\ 0, \text{ else} \end{cases}$$

The objective function is formulated as follows:

$$Area = \sum_{k=1}^{p} \sum_{j=0}^{m} x_k y_k^j a_k^j \tag{4.1}$$

subject to

$$\sum_{j=0}^{m} y_k^j = 1 \quad , \forall k \in \{1..p\} \tag{4.2}$$

Equation 4.2 is used to indicate that for each pattern T_k, only one configuration can be chosen as a solution.

The variable a_k^j denotes the total area required to implement T_k following configuration C^j and is computed is as follows:

$$a_k^j = \begin{cases} A_k * 2^j + \delta area & , \forall j \in \{0..m-1\} \\ A_k * n & , j = m \end{cases} \tag{4.3}$$

In the expression of a_k^j, the private configuration C^m $(j = m)$ doesn't consume additional area units because each processor has each private HW accelerator. However for each shared configuration, a $\delta area$ is added due to bridges consumption.
To linearise the expression of the Area (Equation 4.1), we define a new binary variable z_k^j which is expressed as follows:

$$z_k^j = x_k y_k^j \tag{4.4}$$

In linear form, Equation 4.4 is expressed as follows:

$$\begin{aligned} z_k^j &\leqslant x_k \\ z_k^j &\leqslant y_k^j \\ z_k^j &\geqslant x_k + y_k^j - 1 \end{aligned} \tag{4.5}$$

Now, the objective function is formulated as follows:

$$Area = \sum_{i=1}^{p} \sum_{j=1}^{m} z_i^j a_i^j \tag{4.6}$$

4.3.3 Performance constraint

For an SHt-MPSoC, since all processors are executing the same application, they require the same acceleration. Otherwise, the speedup of the architecture is fixed by the processor with the minimal gain, thereby the performance gain of the overall architecture will be minimal. Let $Tlimit$ denotes the required performance. The total acceleration for each processor i is expressed as follows:

$$acc_i = \sum_{k=1}^{p} x_k (tacc_k - \sum_{j=0}^{m} D_{i,k}^j) \geq Tlimit, \quad \forall i \in \{1..n\} \tag{4.7}$$

The $D_{i,k}^j$ is the contention time required for the processor P_i to share the pattern T_k with the other processors following the configuration C^j. In other words, it represents the waiting time needed by P_i to assure accessing shared accelerators of T_j without conflicts. This variable is expressed in Equation 4.8 as follows:

$\forall j \in \{0..m\}$, $\forall i \in \{1..n\}$, $\forall k \in \{1..p\}$

$$D_{i,k}^j = \begin{cases} 0 \quad \text{for} \quad j = 0 \quad \text{or} \quad i = l * n/2^j \quad , l \in \mathbb{N} \\ x_k(T0_k - y_i^j(tacc_k + td)) \quad , \text{otherwise} \end{cases} \tag{4.8}$$

In this section, we presented a MILP formulation of the design space of application-specific instructions selection and HW accelerators sharing for a SHt-MPSoC architecture. The implementation of each accelerator depends on the area/performance cost. The area cost is modelled in the objective function (Equation4.1) and the required performance is imposed as a constraint (Equation 4.7). Our model explores all the possible accelerators configurations to find the optimal architecture.

4.4 Space exploration in AHt-MPSoC

The aim of this section is to present the space exploration tool to optimise AHt-MPSoC architectures. In previous chapter, we defined an AHt-MPSoC architecture as an Ht-MPSoC architecture where n processors execute different applications and they can have

different number and type of HW accelerators. As in SHt-MPSoC architecture, when a pattern is selected to be executed as application-specific instruction in an AHtMPSoC architecture, the space of configuration of the correspondent HW accelerator is bounded by the fully shared and the fully private configurations. For SHt-MPSoC architecture (section 4.3), we have only considered symmetric solutions of this space. However, for an AHt-MPSoC architecture, the design space exploration has to be carried out by considering symmetric and asymmetric solutions.
In the following sections, we detail the MILP model proposed to explore the design space of a Ht-MPSoC architecture executing different applications. In section 4.4.1, we represent the formulation of the MILP model for the optimization of area-performance cost of AHt-MPSoC architecture. Section 4.4.2 shows how to quantitatively formulate the area cost of an AHt-MPSoC architecture. The area cost represents the objective function of the MILP model. Section 4.4.3 represents the formulation of the performance gain considered as a constraint of the MILP formulation.

4.4.1 Problem formulation

As seen in chapter 3, an AHt-MPSoC architecture execute different multimedia applications, each of which contains several computational tasks. For these applications, we demonstrated that the computational tasks are based on a set of common patterns. Thanks to this communality, the hardware accelerators sharing methodology is beneficial for these architectures. Let $\{T_1...T_j...T_m\}$ denotes the sequence of selected pattern and $P = \{P_1, P_2, ..., P_i, ...P_n\}$ the sequence of n homogeneous processors.

The hardware accelerators configurations of identified patterns are explored to select which ones are good enough to satisfy designer' requirements. Let $N = \{1, 2, \ldots, n\}$ and $M = \{1, 2, \ldots, m\}$.
Each pattern T_j $(j \in M)$ is specified by a number of parameters. These parameters are enumerated as following:

- a_j: the value of which is the number of HW resources which are needed by T_j to be implemented as HW accelerator. For each pattern, the value of a_j is measured with an implementation of T_j as private HW accelerator.
- E_{ji}: is a binary parameter which is equal to 1 if the pattern T_j is executed by the processor P_i , otherwise it is equal to 0.
- ts_{ji} and te_{ji}: are two parameters respectively for the start-time and the end-time of pattern T_j on processor P_i. These parameters are expressed in second and are

determined through profiling step.

- $tacc_j$: the value of which is the execution time gain of T_j pattern. This value is measured with the implementation of T_j as private HW accelerator and is expressed in second.

4.4.2 Objective function

Our optimization problem aims to minimise of an area objective function. Consider the n-processors architecture whose applications can be similar or different. These applications contain a sequence of same patterns $\{T_1...T_j...T_m\}$. The implementation of a pattern on software does not consume HW resources. In contrast, if a pattern is implemented on HW, the HW resources usage depend on the manner of implementing the HW accelerator as private or shared one. Thus, our objective function has to consider two decision variables:

- x_j: a binary variable that denotes whether T_j is implemented on Hw or on Sw .

  $\forall j \in M$

  $$x_j = \begin{cases} 1, \text{ if } T_j \text{ is implemented on HW} \\ 0, \text{ else} \end{cases}$$

- y_{jik} a binary variable that denotes whether the accelerator (Acc) of task T_j is shared between processors P_i and P_k or not.
  $\forall j \in M, \forall (i,k) \in N^2$

  $$y_{jik} = \begin{cases} 1, \text{ if Acc of } T_j \text{ is shared between } P_i \text{ and } P_k \\ 0, \text{ otherwise} \end{cases}$$

The objective function is expressed as follow:

$$Total\_Area = \sum_{j=1}^{m} \sum_{i=1}^{n} E_{ji} x_j \frac{a_j}{\sum_{k=1}^{n} y_{jik}} \tag{4.9}$$

subject to

$$y_{jii} = 1 \quad \text{for } j = 1..m \quad \text{and} \quad i = 1..n \tag{4.10}$$

and

$$y_{jik} \leq E_{ji} \quad \text{for } j = 1..m \quad \text{and} \quad i = 1..n \text{ and} \quad k = \{1..i-1\} \cap \{i+1..n\} \tag{4.11}$$

Equation 4.10 guarantees that denominator in Equation 4.9 is non-zero. Indeed, y_{jik} variable precise if two different processors P_i and P_k share the HW accelerator of the pattern T_j ($j = 1..m$). For $k = i$, if we suppose $y_{jii} = 0$, for each fully private configuration of a pattern T_j, the denominator of Equation 4.9 will be equal to zero for the iteration of the pattern T_j. Thereby the value of y_{jii} has to be set to one. Equation 4.11 guarantees that the pattern T_j will be implemented only for the processors executing this pattern.

$$y_{jik} - y_{jki} = 0 \quad \text{for } j = 1..m \quad \text{and} \quad (i,k) = 1..n \tag{4.12}$$

Equation 4.12 guarantees the symmetry of y_{jik} matrix. In fact, to share a HW accelerator of a pattern T_j between processors P_i and P_k, both y_{jik} and y_{jki} have to be set to one. Otherwise, if P_i and P_k have not a shared HW accelerator of T_j, both y_{jik} and y_{jki} have to be set to zero. Thus, in both cases, y_{jik} and y_{jki} are equal.

$$r_{jikh} = y_{jik} * y_{jih} \quad \text{for } j = 1..m \quad \text{and} \quad (i,k,h) = 1..n \tag{4.13}$$

$$y_{jih} \geq r_{jikh} \quad \text{for } j = 1..m \quad \text{and} \quad (i,k,h) = 1..n \tag{4.14}$$

$$y_{jkh} \leq 1 + 2 * r_{jikh} - y_{jik} - y_{jih} \quad \text{for } j = 1..m \quad \text{and} \quad (i,k,h) = 1..n \tag{4.15}$$

Equations 4.14 and 4.15 denote that if the same processor P_i share a Hw Acc of T_j with P_k and P_h then P_i, P_k and P_h share the same Hw Acc of T_j.

In the objective function (Equation4.9), for each processor P_i , we have implicitly defined the sharing degree for each task T_j. Let sh_{ij} be this variable defined as:

$$sh_{ij} = \sum_{k=1}^{n} y_{jik} \tag{4.16}$$

Figure 4.3 shows an example of decision variables $Y_j = y_{jik}$ for the T_j pattern and an 8-processor architecture. Each row i (respectively column k) in the matrix corresponds to processor P_i (respectively processor P_k) in the MPSoC. The value (1 or 0) on row i and column k determines if processors P_i and P_k share the same Hw Accelerator for T_j.

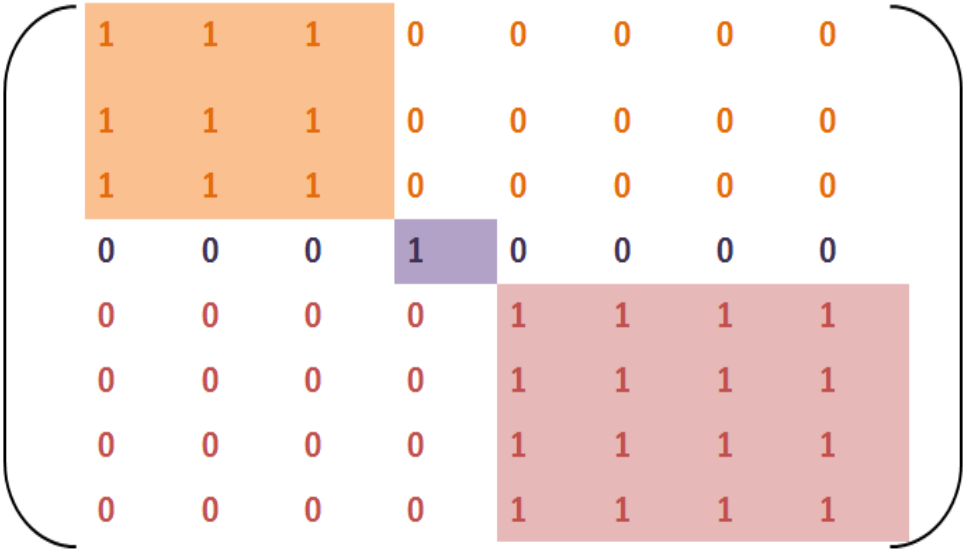

FIGURE 4.3: y_{jik} variables for a T_j pattern. Each row i (respectively column k) in the matrix corresponds to processor P_i(respectively processor P_k) in a 8-processor architecture

For example, from the first three rows (orange region), we deduce that the same HW Accelerator for T_j is shared between P_1, P_2 and P_3. For this region, for each row, the usage of HW resources has been reduced by a factor of 3 (sharing degree for P_1, P_2 and P_3) and is equal to $a_j/sh_{ij} = a_j/3$. Likewise, for the pink region, a 4-shared HW accelerator is shared between P_5 , P_6 , $P7$ and $P8$. For each row i in this region, the HW resource usage is reduced by a factor of 4. The 4th row of the matrix shows that P_4 has a private HW Accelerator for T_j and consumes a_j area units.

To linearise the objective function, we define new continuous variables z_{ij} and w_{ij}

$$z_{ij} = \frac{1}{\sum_{k=1}^{n} y_{jik}} = \frac{1}{sh_{ij}} \tag{4.17}$$

$$w_{ij} = z_{ij} x_j \tag{4.18}$$

The definition of z_{ij} can be expressed in linear form as follows:

$$z_{ij} \sum_{k=1}^{n} y_{jik} = 1 \tag{4.19}$$

$$\sum_{k=1}^{n} \theta_{ijk} = 1 \tag{4.20}$$

Where θ_{ijk} is a continuous variable expressed as follows $\theta_{ijk} = z_{ij}y_{jik}$ and satisfying the following constraints:

$$\begin{aligned} \theta_{ijk} &\leqslant y_{ijk} \\ \theta_{ijk} &\leqslant z_{ij} \\ \theta_{ijk} &\geqslant z_{ij} + y_{ijk} - 1 \end{aligned} \tag{4.21}$$

The objective function (eq. (4.9)) can be re-written as:

$$Total\_Area = \sum_{j=1}^{m}\sum_{i=1}^{n} E_{ji}x_j a_j w_{ij} \tag{4.22}$$

4.4.3 Performance constraint

For many common multimedia-applications, HW accelerators are used to satisfy a required performance. For example, a processor which execute a jpeg decoder has to satisfy a performance of 20 decoded images par second. For our model, we define $limit_i$ and acc_i $(i = 1..n)$, two temporal parameters. $limit_i$ is the required execution-time gain for the processor $P_i.acc_i$ is a variable that calculate the execution-time gain of the generated solution. The expression of acc_i is imposed as a constraint and needs to be upper or equal the required $limit_i$ (Equation 4.23).

$$acc_i \geq limit_i, \quad \forall i \in N \tag{4.23}$$

When a pattern is shared between different processors, access of each processor to the shared Hw accelerator may be delayed. This delay is a crucial parameter to consider in the performance constraint. Let D_{ji} denotes the delay of processor P_i to access a shared Hw accelerator of T_j and is expressed as fllows:

$$D_{ji} = Max(0, d_{ji}) \tag{4.24}$$

$$d_{ji} = te^h_{jk} - ts^h_{ji}, , \quad k = max\{1, ..., i-1\} \text{ and } y_{jik} = 1 \tag{4.25}$$

Now, the performance constraint can be expressed as follows:

$$acc_i = \sum_{j=1}^{m} E_{ji}x_j(tacc_j - D_{ji}) \geq limit_i, \quad \forall i \in N \tag{4.26}$$

In Equation 5.2, the term $(tacc_j - D_{ji})$ is multiplied by x_j to set the acceleration of T_j to zero when it is executed on software ($x_j = 0$). From Equations 4.24 and 4.25, we note that D_{ji} depends on te^h_{jk} and ts^h_{ji}, $j \in M$, $i \in N$, $k \in \{1, 2, ..., i\}$. te^h_{jk} and ts^h_{ji} are continuous variables that define respectively the start-time and the end-time of executing T_j n processors P_k and P_i. These variables are calculated following Equation 4.27 and depends on : software execution time of T_j on P_k and P_i; acceleration provided with T_l patterns ($l = 1..j-1$), on processors P_k and P_i. T and are calculated as follow:

$$\begin{aligned} &\forall j \in M, \forall i \in N, \forall k \in \{1, 2, ..., i\}, \\ &ts^h_{ji} = ts_{ji} - \sum_{l=1}^{j-1} x_l(acc_l - D_{li}) \\ &te^h_{jk} = te_{jk} - \sum_{l=1}^{j} x_l(acc_l - D_{lk}) \end{aligned} \tag{4.27}$$

In Equation 4.24, the expression $k = max\{1, ..., i-1\}$ and $y_{jik} = 1$ denote the last processor sharing the Hw Accelerator of T_j with P_i. In order to linearise this Equation, we define a new binary variable p_{jik} which is defined as follow:

$$p_{jik} = \begin{cases} 1, \text{ if } P_k \text{ is the last processor sharing } T_j \text{ with } P_i \\ 0, \text{ else} \end{cases}$$

Now, Equation 4.24 can be rewritten as follows:

$$D_{ji} = \sum_{k=1}^{i-1} p_{jik} d_{jik} \tag{4.28}$$

4.4.3.1 Calculation of the access-delay to a shared HW accelerator

We presented in Equations 4.24 and 4.25, the expression of the delay D_{ji} of the processor P_i to access a shared HW accelerator of the pattern T_j. The calculation of this delay depends on the start-time of software execution of T_j on $P_i(ts^h_{ji})$ and the end-time of hardware execution of T_j on the last processor P_k (te^h_{jk}), sharing T_j with P_i.
In Equation 4.29, we defined a binary variable p_{jik} to find out the last processor P_k sharing a pattern T_j with the processor P_i.

For each pattern T_j and for each processor P_k, the search of p_{jik} is guided through the following assumptions:

1. If processors P_k and P_i haven't a shared HW accelerator of pattern T_j ($y_{jik} = 0$) then p_{jik} will be equal to zero.

$$p_{jik} \leq y_{jik} \ \forall j \in M, \forall (i,k) \in N^2 \tag{4.29}$$

2. For a processor P_i and a pattern T_j, the sum of p_{jik} over k is equal to 1 or 0

$$\sum_{k=1}^{n} p_{jik} \leq 1 \ \forall j \in M, i \in N$$

3. If processor P_i is the first processor to access a shared HW accelerator of pattern T_j then p_{jik} (k=0..n) will be equal to zero.

$$\begin{gathered} \forall j \in M, \forall i \in N, \forall k \in \{1, 2, \ldots, i\} \\ \text{if } (\sum_{k=1}^{i-1} y_{jik} \leq 0) \text{ Then } \sum_{k=1}^{n} p_{jik} \leq 0 \end{gathered} \tag{4.30}$$

4. If P_k is the last processor sharing with P_i the HW accelerator of pattern T_j then p_{jik} will be equal to 1. In other words

$$\begin{gathered} \forall j \in M, \forall i \in N, \forall k \in \{1, 2, \ldots, i\} \\ \text{if } (y_{jik} - \sum_{l=k+1}^{i} y_{jil} \geq 1) \text{ Then } p_{jik} \geq 1 \end{gathered} \tag{4.31}$$

   In a linear form, this assumption can be expressed as follows:

$$\begin{gathered} \forall j \in M, \forall i \in N, \forall k \in \{1, 2, \ldots, i\} \\ y_{jik} - \sum_{l=k+1}^{i} y_{jil} + V1.r_{jik} \geq 1; \\ p_{j,i,k} + V2.r_{j,i,k} >= 1; \\ V3(1 - r_{j,i,k}) >= y_{j,i,k} - \sum_{l=k+1}^{i} y_{jil}; \end{gathered}$$

   Where $V1$, $V2$ and $V3$ are large constants and r_{jik} is a binary variable.

5. If P_k is not the last processor sharing with P_i the HW implementation of task T_j then p_{jik} will be equal to 0. This assumption is expressed as an IF-THEN

constraint:

$$\forall j \in M, \forall i \in N, \forall k \in \{1, 2, \ldots, i\}$$
$$\text{if} \quad \sum_{l=k+1}^{i} y_{jil} \geq 1 \quad \text{Then} \quad p_{jik} \leq 0 \tag{4.32}$$

In linear form, this constraint is expressed as follows

$$\begin{aligned} \forall j \in M, \forall i \in N, \forall k \in \{1, 2, \ldots, i\} \\ \sum_{l=k+1}^{i} y_{jil} + V1.q_{jik} \geq 1; \\ 1 - p_{j,i,k} + V2.q_{j,i,k} \geq 1; \\ V3(1 - q_{j,i,k}) \geq \sum_{l=k+1}^{i} y_{jil};; \end{aligned}$$

Where q_{jik} is a binary variable.

Now the period constraint can be re-written as:

$$\forall i \in N$$
$$acc_i = \sum_{j=1}^{m} tacc_j - \sum_{k=1}^{i-1} p_{jik} * (te\_hw_{jk} - ts\_hw_{ji}) >= limit_i \tag{4.33}$$

4.4.3.2 Illustrative example to calculate the access-delay to a shared HW accelerator

In this section, we illustrate an example to find out p_{jik} variables for T_j pattern and to calculate the delay to access the shared Hw accelerator. For each pattern T_j, the MILP model look for the configuration of its HW accelerator that satisfy the required performance and minimize the area usage. If we consider the solution of Figure 4.3, we note that two HW are implemented on 8-processor architecture to execute the pattern T_j. The first Hw accelerator is shared between 3 processors (P_1, P_2 and P_3). Figure 4.4.a shows the execution of pattern T_j for these processors on software. The delay of each processor to access the shared Hw accelerator (Figure 4.4.b) was calculated through the determination of p_{jik} variables.

From Figure 4.24.b, we note that P_1 access to the shared Hw accelerator of T_j without a delay. In fact p_{jik} variables (for $i =$ and $k = 1..8$) are found out using matrix 4.3

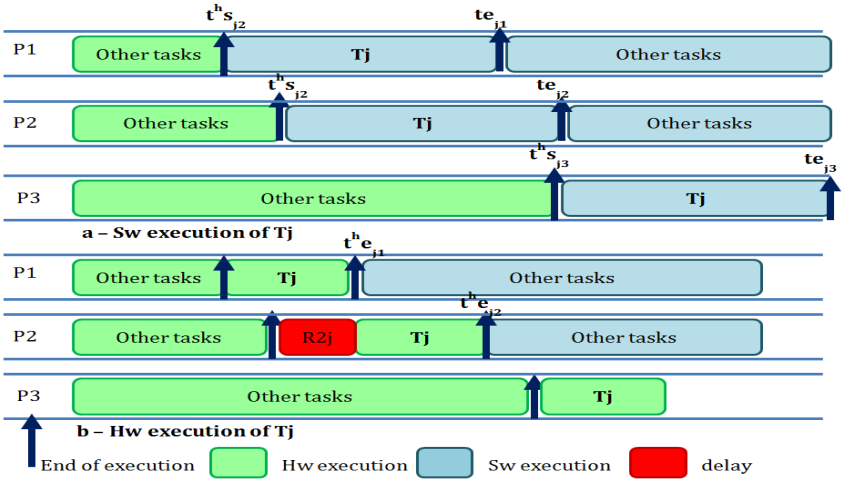

FIGURE 4.4: Illustrative example to calculate an access-delay to a shared HW accelerator

and Equation 4.30, and are equal to 0. This is explained by the fact that P_1 is the first processor executing the pattern T_j and thereby is the first processor accessing to the shared HW accelerator. Thus, D_{1j}, the delay of P_1 to access T_j was calculated from Equation4.28 and was equal to zero.

In Figure 4.24.b the delay of processor P_2 is non zero. In fact, For processor P_2, p_{jik} variables (for $i = 1$ and $k = 2..8$) are given by Equations 4.29 and 4.31 and are equal to zero except for P_1 ($p_{j21} = 1$). D_{2j} was calculated using Equations 4.33 and 4.25 as follows:

$$D_{2j} = p_{j,2,1} * d_{j,2,1} = p_{j,2,1} * (te^h_{j1} - ts^h_{j2})$$

Finally, for processor P_3, D_{3j} was calculated following Equations 4.33 and 4.25 :

$$D_{3j} = Max(0, d_{j32}) = Max(0, te^h_{j2} - ts^h_{j3}) = 0$$

Since $te^h_{j2} - ts^h_{j3}$ is negative, the delay of processor P_2 to access P_2 is negated. This is explained by the late execution of T_j on P_3 when compared to its execution on P_1 and P_2.

4.5 Conclusion

This chapter presents an original technique to extend Ht-MPSoC architectures with HW accelerators in an efficient way. In order to save time and effort, an iterative approach is adopted to select the computational tasks to be explored. In each iteration,

the computational tasks are explored to look for a Ht-MPSOC configuration that minimizes the area consumption and provides the required performance. The exploration process combines the design space of HW customization and HW accelerators sharing for common computational tasks. Such an integration unveils new trade-off between area consumption and performance gain. The exploration process integrates a MILP formulation which is based on linear equations to define the objectives and constraints. In order to guarantee the efficiency of the exploration process, metrics that quantify the execution time overhead when HW accelerators sharing is enabled are used.

Chapter 5

Experimental Results

5.1 Introduction

This chapter presents experimental results obtained during the validation of the contributions presented in chapters 3 and 4. In order to validate and evaluate the proposed HW accelerators sharing methodology, we present in section 5.3 a case study based on real application. A discussion on the impact of HW accelerators sharing on performance, area and energy trade-off is presented. The proposed technique for the selection of optimised Ht-MPSoC architecture is evaluated in section 5.4. Section 5.5 concludes this chapter.

5.2 Target Platform and Application

This thesis uses FPGA devices from Xilinx, Inc [127]. Xilinx was the first company to manufacture FPGAs and they are until now the market leaders [128]. It comprises approximately 47% market share versus 41% to Altera company.
The Xilinx FPGA is made up of different blocks, digital signal processing (DSP48) units, digital clock manager (DCM), block RAMs (BRAM), programmable interconnect points (PIP) and configurable logic blocks (CLB), etc (Figure 5.1). Each CLB is constituted of four interconnected slices. The slices are made up of look-up tables (LUT), flip-flops (FF), multiplexers, and a few gates. Our target platform for this thesis consists of a Xilinx ML507 FPGA board [7]. The Xilinx ML507 board makes use of the XC5VFX70T FPGA chip with other components such as DDR memory, Flash memory, Ethernet and multiple PC interfaces like USB and PS/2. Figure 5.2 shows a schematic overview of the ML507 block diagram.

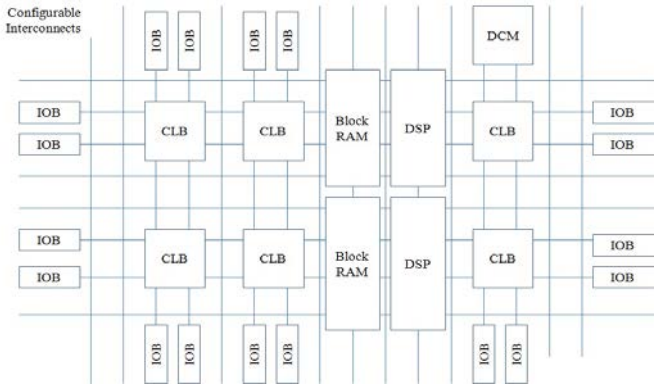

FIGURE 5.1: General structure of an FPGA

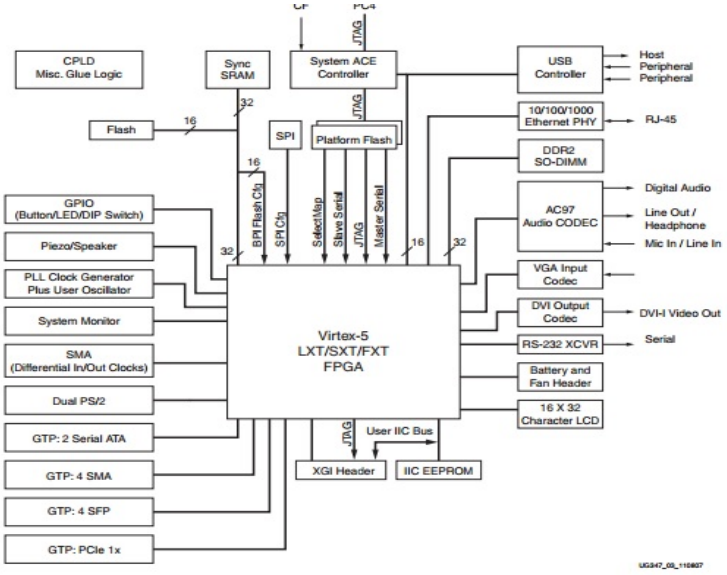

FIGURE 5.2: Block diagram of ML507 board [7]

5.2.1 Xilinx Development Tool

Xilinx recommends ISE (Integrated Synthesis Environment) Design Suite for design starts with Spartan-3 to Virtex-7. In this thesis, we used ISE design suite 12.4. Figure 5.3 illustrates the design flow of Xilinx Development Environment which is essentially based on: SDK, XPS, ISE and PlanAhead. The ISE Project Navigator (ISE) allows the design and the synthesis of Verilog HDL or VHDL modules. The synthesis process generates the netlist file using Xilinx Synthesis Technology (XST). The Xilinx Platform Studio (XPS) is a part of a function of the Embedded Development Kit (EDK). XPS is used to design architectures based on embedded processor, such as Microblaze or PowerPC. The application running on the embedded processor is implemented using Software Development Kit (SDK). The application is compiled to an Executable and Linkable Format (ELF) file. Using Data2MEM utility [129], various files generated by these tools are assembled into a bitstream file for initializing FPGA.

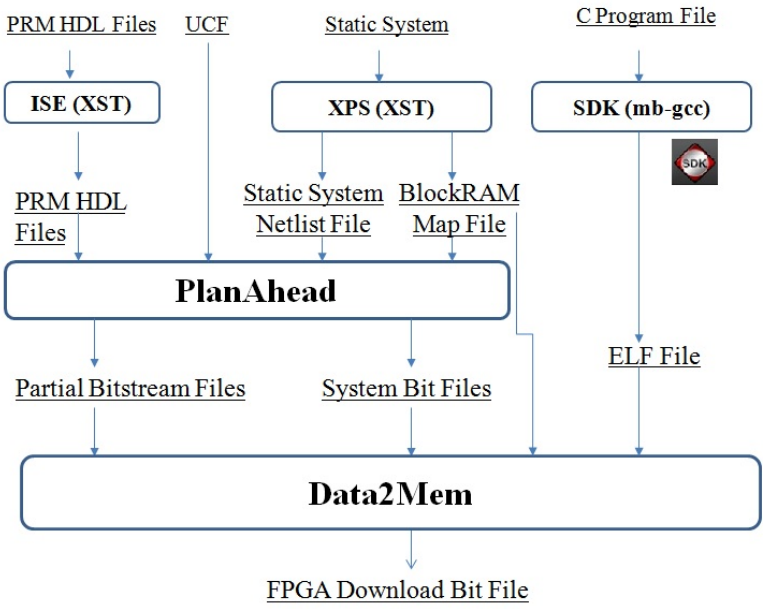

FIGURE 5.3: Design flow of Xilinx development environment

5.2.2 Microblaze Processor

To speed up the design of digital systems, FPGA companies propose their softcore processors and provide designers wizard tools to export the most suitable configuration [130].

The Microblaze is a Xilinx softcore, proposed with the complete programming environment EDK. It is a RISC (Reduced Instruction Set Computer) processor implemented on the FPGA internal resources (arithmetic, logic and memory). The FPGA resource usage of a Microblaze processor depends on its configuration including cache size or not, pipeline depth (3-stage or 5-stage), memory management unit enabled or not, etc.

The Microblaze is a 32-bit Harvard compliant architecture. It uses two Local Memory Buses (LMB) for instruction and data memories, two Block RAMs (BRAM) and peripherals are connected via Processor Local bus (PLB) or On-chip Peripheral Bus (OPB).

5.2.3 XILINX Interconnect System

Xilinx propose versatile interconnect systems to facilitate the design of embedded systems based on Microblaze processor. The first proposed I/O bus is the OPB bus. Later, XILINX has developed another I/O bus and migrates all processor IP cores to this new PLB bus. This migration is motivated by the neeed to increase the system performance [8] . The PLB bus, is a traditional system-memory mapped transaction bus with master/slave capability. PLB bus supports a 32-bit address bus and 32-bit, 64-bit, or 128-bit

data bus. As highlited in Figure 5.4, the PLB transactions are divided into two phases: address phase and data phase. The address phase consists on driving requested address and transaction qualifiers to all slaves. This phase starts with an assertion of valid signal and finishes with an acknowledgement signal.

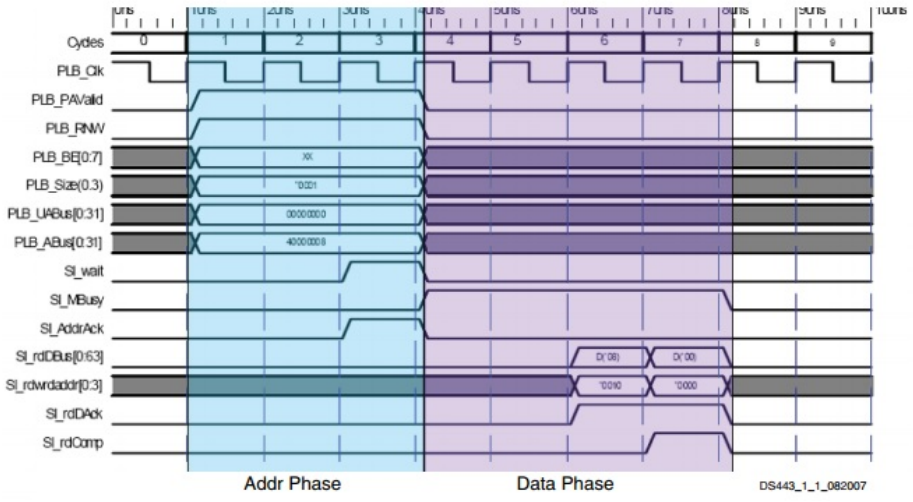

FIGURE 5.4: PLB address phase and data phase [8]

To access to local-memory (FPGA BRAM), Microblaze uses a dedicated LMB bus, which reduces loading on the other buses. User-defined coprocessors are supported through a dedicated FIFO-style connection called FSL (Fast Simplex Link). The coprocessor(s) interface can accelerate computationally intensive algorithms by offloading parts or the overall computation to a user-designed hardware module.

5.2.4 Implementation of a Single-processor Architecture on ML507 Board

In this subsection, we describe the implementation of a single processor architecture on ML507 board. This architecture will be useful along this chapter in order to profile applications and to measure their execution time. The architecture is presented in Figure 5.5 and is composed of a Microblaze processor, on-chip memory (BRAM block in Figure 5.5) and a PLB bus to communicate the processor to different peripherals. The BRAM blocks are connected to processor through data and instruction memory buses (DLMB and ILMB in Figure 5.5). The xps_uartlite is used as peripheral for debugging and I/O purposes. The xps_timer peripheral measures data transfer times and serves to build profiling results. The CF card stores input files for the applications which will be encoded and communicated to the processor via SysACE_compactflash.

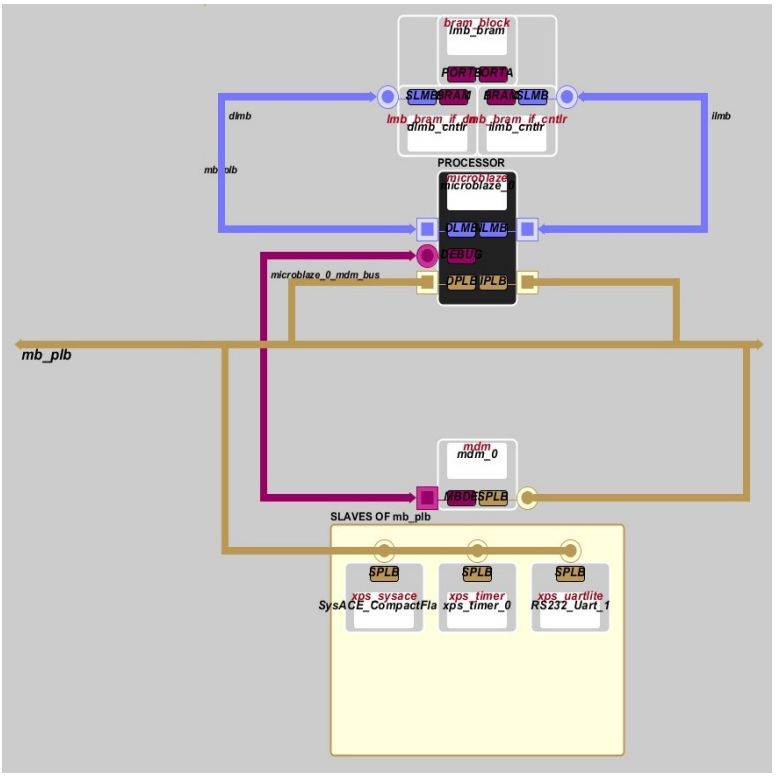

FIGURE 5.5: Implemented Microblaze-based architecture for jpeg-encoder application

5.3 Evaluation of Customized Ht-MPSoC Architecture with Shared and/or Private HW Accelerators

In this section, the efficiency of the proposed HW accelerators sharing methodology will be evaluated using the jpeg encoder application. At the beginning of this section we give an overview of the jpeg encoder application as well as preliminary results of the implementation of this application on a Microblaze processor (total execution time, profiling results and a synthesis summary). Then, the implementation results of jpeg encoder application on different Ht-MPSoC configurations will be presented. The impact of HW accelerators sharing on the efficiency of this architecture will be presented.

5.3.1 Overview of Jpeg Encoder Application

JPEG stands for Joint Photographic Experts Group and it refers to the committee that created the JPEG [131] [132]. JPEG adopts a lossy compression approach based on the discrete cosine transform (DCT). This mathematical algorithm converts each frame of the video source from the spatial (2D) domain into the frequency domain. This algorithm is capable to carry out a high degree of compression with minimal loss of data. Figure 5.6 presents the main five steps of JPEG compression. The first step

consists on converting the image from RGB into a different color space called Y`CBCR. Y component represents the brightness of a pixel, and the CB and CR components represent the chrominance which is divided into blue and red components. The DCT transformation is the second step and aims to remove redundant image data. The third step allows the reduction of the amount of information in the high frequency components. Thus for the quantization process, each component is divided in the frequency domain by an adequate constant and then rounded to the nearest integer. This rounding operation is the only lossy operation in the whole process. The fourth step arranges the image components in a "zigzag" order using run-length encoding (RLE) algorithm that groups similar frequencies together. The final step outputs the DCT block's elements using an entropy encoding mechanism that combines the principles of RLE and Huffman encoding [131].

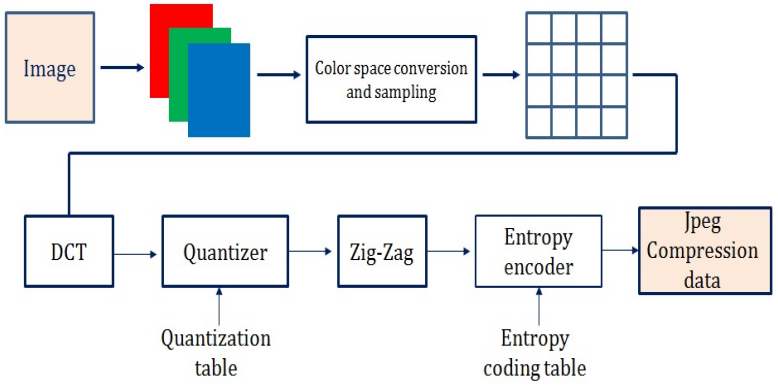

FIGURE 5.6: Jpeg encoder processing steps

5.3.2 Preliminary Implementation Results for Jpeg-encoder Application on Microblaze-based Architecture

In order to obtain preliminary results of the implementation of the JPEG encoder on a Microblaze-based architecture, we configured the BRAM of the architecture of section 5.2.4 to 128 Kbytes to store the executable file of the application. The baseline JPEG encoder application is downloaded from [132]. Tables 5.1 and 5.2 summarize the synthesis report of the implemented architectures.

On this single-processor architecture each bmp image is compressed to jpg format in 0.16 second. Thus, 3.27 seconds are needed to encode 20 images. Profiling results of jpeg encoder application are presented in Figure 5.7. To achieve good video quality, the encoder process has to encode 20 image frames per second. In order to take advantages of the data-level parallelism of the jpeg encoder and to ameliorate its execution time, we implemented this application on a 2 and 4-processor architectures

TABLE 5.1: Logic utilization of baseline components Of Microblaze-based architecture

	Flip-Flop used	LUTs used	Bram used
System	2510	2596	32
Timer	358	287	-
MDM	126	124	-
Sys_ace	210	96	-
RS232	144	130	-
Microblaze	1451	1549	-
BRAM	-	-	32

TABLE 5.2: Synthesis report of Microblase-based architecture

Slice Logic Utilization	Available	Utilization	Pourcentage
Number of slice register	44800	2110	4%
Number of slice luts	44800	2283	5%
Number of occupied slices	11200	890	7%

In our experiments a master processor controls the transfer of images between the CF card and shared memories. Each slave processor have a shared memory with the master processor. When the master processor finishes writing an image in the shared memory, the slave processor is activated by setting its flag. The slave processor starts executing its program and stores the resulting image in shared memory.

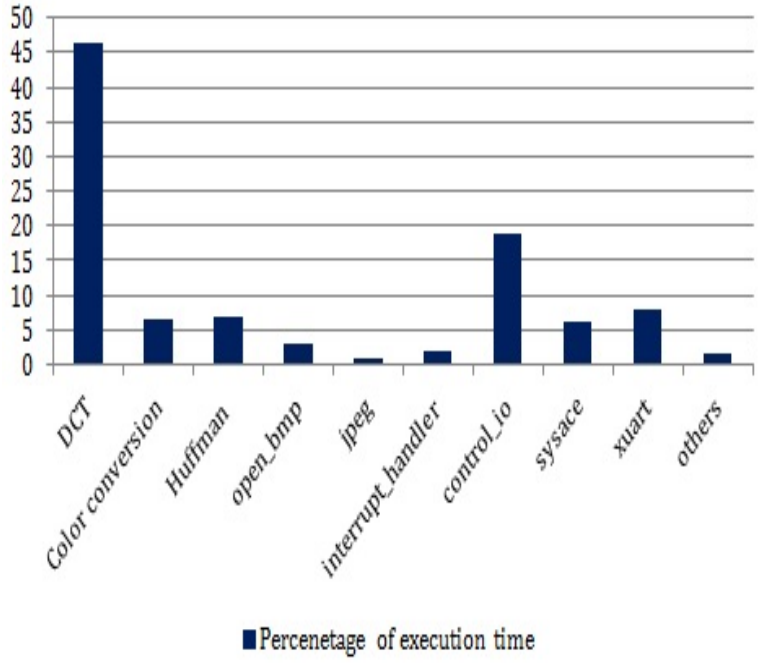

FIGURE 5.7: Profiling results of jpeg encoder application executed on Microblaze processor

The execution time for the 2-processor configuration is measured to 0.0815 sec/image and thus we obtain 1.63 seconds to process 20 images. For the 4-processor configuration, the execution time is measured to 0.065 sec/image and thus 1.3 seconds are needed to process 20 images. We note that increasing the number of processors do not provide the sufficient speed-up to encode 20 images per second. In next section, the execution time will be improved by customizing the computational tasks. Different configurations with different types and sharing degrees of HW accelerators will be evaluated and compared.

5.3.3 Evaluation of Microblaze-based MPSoC Configurations with Private and Shared HW accelerators

In the following, we implement different SHt-MPSoC configurations executing the data-parallel jpeg-encoder application. The integration of HW accelerators is intended to improve performance of the jpeg encoder application. Based on profiling results of Figure 5.7, we select HDCT and VDCT as computational patterns which are candidate for HW customization. In order to improve the performance of the overall architecture, SHT-MPSoC configurations customize the same computational tasks. For each multi-processor architecture, the number of shared configuration depends on the number of processors. For example for a 2-Microblaze architecture, for each accelerator, only one shared configuration is possible. However, for a 4-Microblaze architecture, for each HW accelerator, two shared configurations are possible (2-shared configuration and 4-shared configuration). Figure 5.8 is an example of implemented architecture with 4 Microblazes and two shared HDCT HW accelerators. Each shared HW accelerator has its private plb bus (Sh1 and Sh2 in Figure 5.8). For each shared HW accelerator, two plb bridges are connected to its private bus as masters to communicate the shared HW accelerators to their processors.

TABLE 5.3: Acceleration and area consumption of HDCT and VDCT tasks

	Acceleration (10^6clock cycles)	Area usage	
		Luts slices	Slice registers
HDCT 203	20.25	6126	782
VDCT 188	f 17.75	5688	680

Table 5.3 details the synthesis results of the HDCT,VDCT HW accelerators and their acceleration. The acceleration of each HW accelerator is measured by Equation 5.1 in clock cycles, where $SW\_exec(task)$ and $HW\_exec(task)$ denote the execution of the correspondent task on the processor and on HW accelerator respectively.

$$acceleration(task) = SW\_exec(task) - HW\_exec(task) \tag{5.1}$$

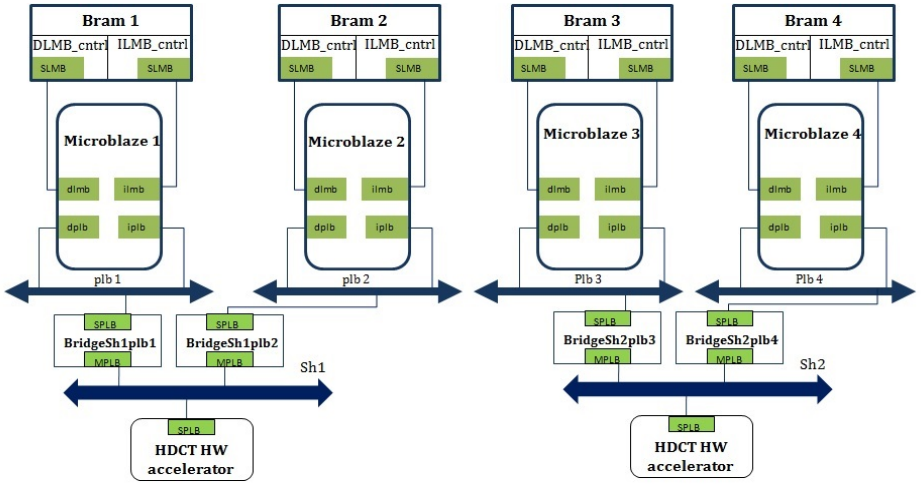

FIGURE 5.8: Example of implemented architecture: Configuration with four processors and two 2-shared HDCT HW accelerators

In the following, we expose results of area consumption, execution time and energy consumption of different SHt-MPSoC configurations executing the jpeg encoder application. Varying the number of processors in the architecture, we implemented different private and shared architectures. In order to synchronize the access of processors sharing the same HW accelerator, a synchronization flag is added to its HW description (See Figure 5.9).

```
//control the syn register, we use test&set op
while (Read HwA(baseaddr, reg _control offset) == 1)
Do
//if shared HwAcc is in use, Pi makes no op and waits
;
end while
//Pi uses HwAcc
write HwA (baseaddr, reg_control offset, 1 );
// processing HwA and read results
//Pi relases the Hw Acc
write HwA(baseaddr, reg _control offset; 0);
```

FIGURE 5.9: Synchronization mechanism for a shared HW accelerator

The area consumption and the execution time of the implemented configurations are depicted in Figures 5.11 and 5.10. In these figures, the x axis corresponds to the different configurations. For example for $p = 4$, the configuration (2,2) corresponds to an SHt-MPSoC with 4 Microblazes, two 2-shared HDCT accelerators and two 2-shared VDCT accelerators.

From Figure 5.10, we note that the implementation of the configuration with 4 Microblaze processors, 4 HDCT HW accelerators and 4 VDCT HW accelerators was not

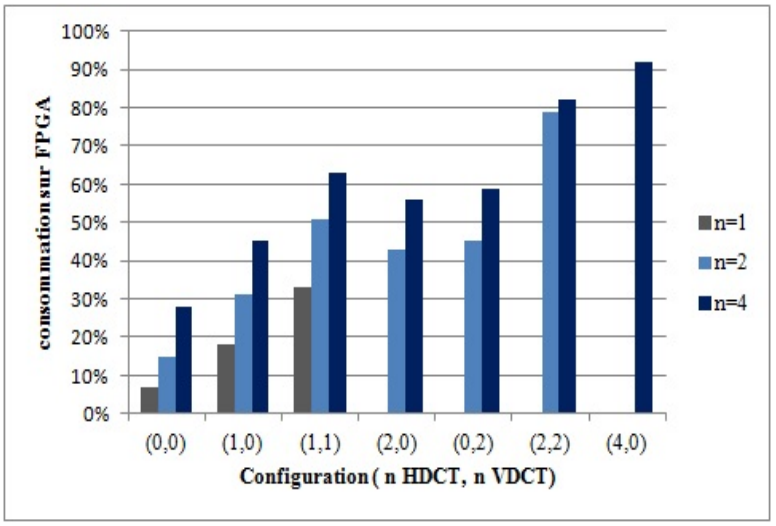

FIGURE 5.10: Slice percentage occupation of different implementations measured on the Xilinx ML507 for different multiprocessor architectures (p=1,2,4) and (HDCT, VDCT) configurations. For p=4 and (4,4) configuration, the area occupation (140%) is estimated based on the other results.

possible due to FPGA resources constraint. So, without HW accelerators sharing, we cannot take advantage of both HDCT and VDCT customization. Indeed, thanks to HW accelerators sharing methodology, the HW accelerators of HDCT and VDCT were implemented on the same architecture, presented by configurations (1,1) and (2,2) in Figure 5.10.

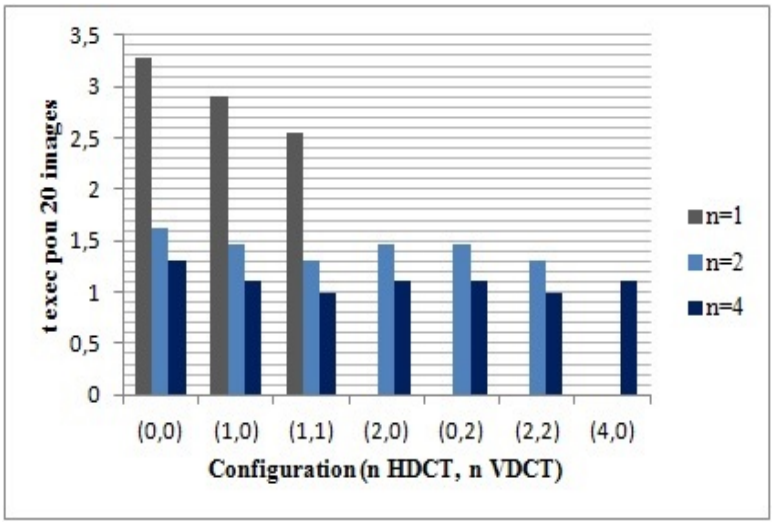

FIGURE 5.11: Execution time in seconds to encode 20 images measured on different multiprocessor architectures (p=1,2,4). For the p=4 and (4,4) configuration, the execution time is estimated based on the other results.

From figure 5.11, we note an overhead between configurations with the same number of processors integrating the same types of HW accelerators but implementing different

number of HW accelerators for the same task. For example, in Figure 5.11, for $p = 2$, both (1,1) and (2,2) configurations used HDCT and VDCT HW accelerators. However, the configuration (1,1) has only one shared HW accelerator for each task whereas the configuration (2,2) has two private HW accelerators for each task. The overhead is due to delay caused by accessing the shared HW accelerator through a bridge (See Figure 5.8). Figure 5.10 confirms that the implementation of private HW accelerators consumes much more area than the configuration with shared accelerators. For instance, the configuration with 1 shared HDCT accelerator reduces the area consumption by 37% compared to the 4-private HDCT configuration. These results show that our proposed technique of sharing hardware accelerators offers a reduced area consumption with a satisfied performance.

The reduced area consumption, provided with the proposed sharing methodology, offers the possibility for the HDCT and VDCT HW accelerators and the processors to be included in the same FPGA chip and so providing better gain in execution time and reduced area cost. This integration on the same chip was not possible using the private approach. Only configurations with 4 Microblaze processors and implementing HDCT and VDCT HW accelerators, configurations (1,1) and (2,2) for $p = 4$, satisfy the requirement of 20 encoded images per second. The configuration (1,1) and $p = 4$ is the most efficient architecture since it satisfies the required performance and consumes less area logic than the configuration (2,2).

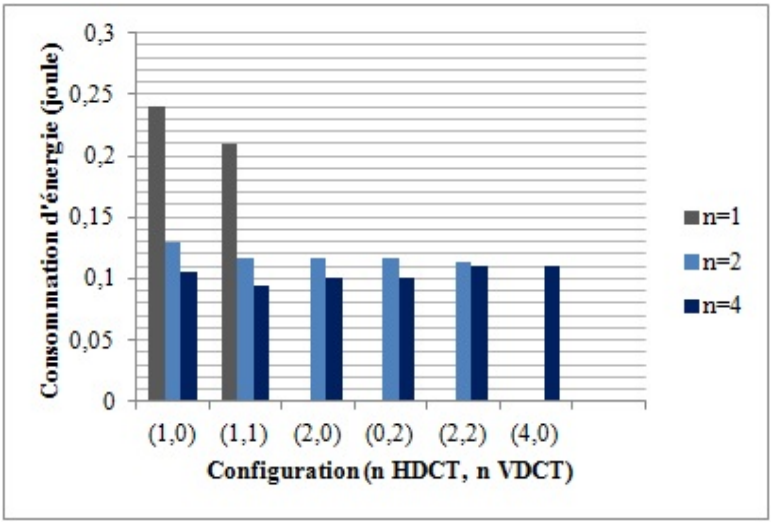

FIGURE 5.12: Energy consumption of the different architectures per encoded image(joules)

Figure 5.12 presents the total energy consumption per encoded image. These values correspond to the static and the dynamic parts of the energy consumption for the whole FPGA. This figure shows that the 1-Microblaze architecture consumes an important energy per image compared to the other architectures. This important difference is

explained by the reduced execution time for multi-Microblaze configurations. We also notice that the different multiprocessor configurations with the same number of HW accelerators have approximately the same energy consumption. For the 2-processor architecture configuration, the implementation of 1 shared HDCT and 1 shared VDCT accelerators, noted (1,1) and P=2 in Figure 5.12, consumes the same energy as the implementation of two private HDCT accelerators, (2,0) and p=2 in Figure 5.12, but with more increased execution time (figure 5.11).

5.4 Experimental Results for the MILP Models

In this section, in order to evaluate the effectiveness of MILP model exploration of proposed technique presented in chapter 4, we give experimental results of synthetic and real applications. For each case study, we explore the design space configurations for different performance constraints and we compare area consumption and execution time of generated solutions to real FPGA measurements.

5.4.1 Case study 1: Synthetic Applications

In this subsection, we use synthetic applications that are produced based on three computational patterns and others non-computational tasks implemented as loop iterations, as illustrated in Figure 5.13. For the different processors, we vary the number of iterations (i, j and k) of the non-computational loops to obtain different applications and different delays between the computational tasks T1 to T3. The three computational tasks consist on:

- T1 implements an inversion of a 16-bit vector.
- T2 implements a multiplication of 8*8 matrices of 32-bits integers.
- T3 implements search the maximum value in a 64-elements vector of 32-bits integers.

Table 5.4 summarizes the execution time of T1, T2 and T3 tasks on the Microblaze processor and their area requirement. Note that the area requirement is presented in term of area unit. Here, an area unit corresponds to 150 slices. In these experiments only the additional area needed for HW accelerator is given, as the number of soft-cores is constant and has been fixed to 8.

Figure 5.14 presents the logic area usage calculated based on our proposed MILP model while varying the required speed-up. The required speed-up is calculated in Equation

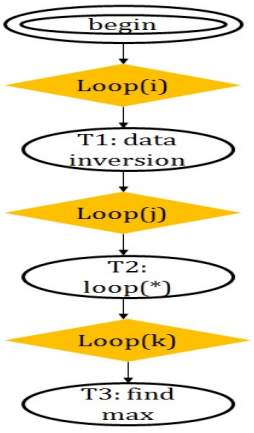

FIGURE 5.13: Generation of different synthetic applications

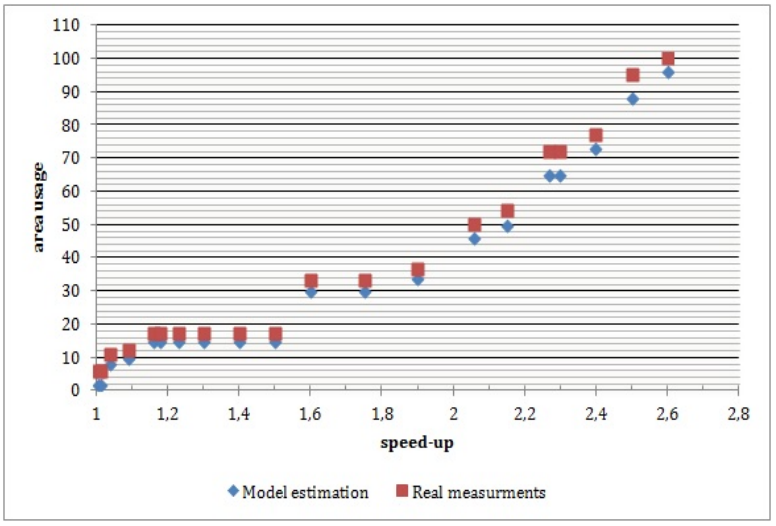

FIGURE 5.14: Area usage (y-axis) of the MILP model generated configurations for different speed-up (x-axis) Vs. configurations 8 processors without HW accelerators. FPGA-based implementation results (real measurements) are also given.

TABLE 5.4: T1, T2 and T3 area and execution time information

	Area usage (area unit)	SW time (cycles)	HW time (cycles)
T1:Data inversion loop	2	440	222
T2:Loop multiplication	15	3815	213
T3:Search maximum	4	2000	1200

5.2 using the required execution time gain $limit_i$ and T_{sw}, the execution time without HW Accelerator.

$$speedup = \frac{T_{sw}}{(T_{sw} - limit_i)} \tag{5.2}$$

In Figure 5.14, the configurations given by the MILP model to satisfy 1.007 and 1.014 speed-ups consume only 2 area units. These solutions have only one shared HW accelerator for T1 ($x_1 = 1$ and $y_{1ik} = 1$), whereas T2 and T3 are executed in software

($x_2 = 0, x_3 = 0$). In contrast, when the speed-up is increased, the generated solutions integrate T2 and T3 on HW accelerators. In Figure 5.14, the solution given for a speed-up of 2.15 consumes 25 additional area units to implement HW accelerators. This solution represents a configuration with HW accelerators for T1, T2 and T3 ($x_1 = 1$, $x_2 = 1$ and $x_3 = 1$).

To illustrate the impact on performance and area consumption when HW accelerators are shared, we compare configurations of different points in Figure 5.14 consuming the same area. In Table 5.5, we compare the MILP outputs for points 1.6 and 1.75 of Figure 5.14. Both solutions need 30 additional area units to satisfy the required speed-ups, but they correspond to different configurations. In fact, for 1.6, the MILP model generates an AHt-MPSoC architecture with two HW accelerators of T2, one shared between (P1, P4, P5, P8) and the second is shared between (P2, P3, P6, P7). Whereas the AHt-MPSoC architecture which provides a speed-up equal to 1.75, has also HW accelerators of T2, but the first one is shared between (P1, P3, P5, P6) and the second one is shared between (P2, P4, P7, P8). We deduce that, different combinations of processors sharing the same HW accelerator could impact the performance of AHt-MPSoC architecture. In fact as noted from Table 5.5, the combination of processors to implement the shared HW accelerators of T_2 impacts the delay matrix. For each constraint, the MILP exploration looks for the combination whose time delay results the required speed-up. Thus, for a fixed area on the FPGA, the designer has several possible configurations and he/she will choose the optimal configuration that provides higher performances.

TABLE 5.5: Comparison of generated configurations for T_2 task for two different speed-ups

	Speed-up=1.6	Speed-up=1.75
Area cost	30	30
Combination of processors	(P1, P4, P5, P8) (P2, P3, P6, P7)	(P1, P3, P5, P6) (P2, P4, P7, P8)
Delay vector	$\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 200 & 0 & 350 & 190 & 520 & 350 \end{Bmatrix}$	$\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 110 & 430 & 120 & 310 \end{Bmatrix}$

Figure 5.14 also demonstrates that the maximum speed-up is provided with a reduced area-usage configuration compared to the configuration with only private HW accelerators. The 8-processor architecture with private HW accelerators for T1, T2 and T3 patterns provides a speed-up equal to 2.6 and consumes 136 area units. To guarantee the same speed-up, our MILP model generates a configuration that consumes only 96 area units. The generated AHt-MPSoC architecture integrates T1, T2 and T3 on HW accelerators ($x_1 = x_2 = x_3 = 1$ as shown in Table 5.6). This architecture has 4 HW accelerators for T1, 4 HW accelerators for T2 and 7 HW accelerators for T3. In Table

5.6, y_{1ik} vector indicates that for T1, processors (P1, P2, P3, P4) and (P5, P7) have two shared HW accelerators and (P6) and (P8) have their private ones.

TABLE 5.6: MILP Model resolution for a speed-up equal to 2.6

	Model Variables	Variables value
Model inputs	N	8
	M	3
	acc[M]	$\{200 \quad 3375 \quad 1000\}$
	a[M]	$\{2 \quad 15 \quad 4\}$
	te[M][N]	$\begin{Bmatrix} 440 & 670 & 910 & 1150 & 1240 & 1360 & 1470 & 1680 \\ 4255 & 4485 & 4725 & 4965 & 5055 & 5175 & 5245 & 5495 \\ 6255 & 6485 & 6725 & 6965 & 7055 & 7175 & 7245 & 7495 \end{Bmatrix}$
	limit[N]	$\{4150 \quad 4100 \quad 4200 \quad 4280 \quad 4300 \quad 4350 \quad 4500 \quad 4575\}$
Decision variables	x_j	$\{1 \quad 1 \quad 1\}$
	y_{1ik}	$\begin{Bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{Bmatrix}$

In Figure 5.14, we also compare the area usage of the MILP-based generated results and the real results obtained with the implementation on the FPGA. From this figure, due to the extra consumed logic-area needed by the bus-bridges, we note a slight overhead difference between real measurement and MILP estimations. The area overhead depends on the number of implemented patterns as shared HW accelerators and is comprised between 3% and 6%. For a speed-up equal to 1.6 only T2 is implemented as shared HW accelerator, thus the area overhead is about 3%. While for a speed up equal to 2.15, all patterns are mapped on shared HW accelerator and the area overhead reaches 6%. These results demonstrate that the proposed MILP model produces results close to those obtained with real implementation.

5.4.2 Case study 2: Jpeg Codec Application

In this section, we evaluate the proposed MILP-based exploration of Ht-MPSoC architectures. Figure 5.15 presents a general overview of the Jpeg codec. The image is decomposed into 8*8 blocks of pixels. Each block is compressed through the encoder process. The array of compressed blocks is stored or forwarded to transmission channels. The image is reconstituted through the decoder process.

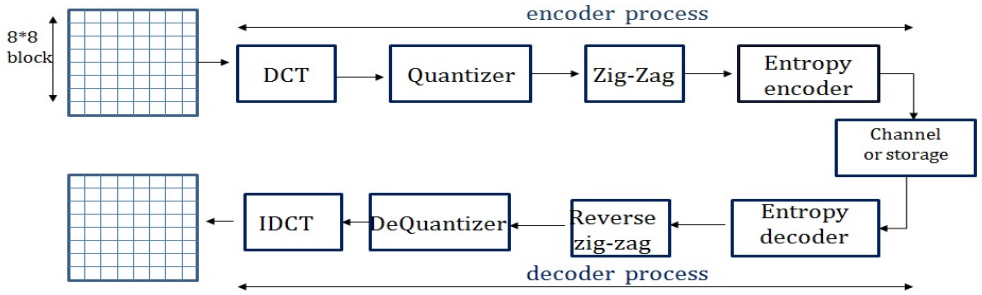

FIGURE 5.15: Jpeg encoder and decoder tasks

In our experiments, we use a 10 Kbytes image and we profile the jpeg-encoder and the jpeg-decoder applications on a Microblaze processor. Profiling results show that DCT (respectively IDCT) is the most time-consuming function for jpeg-encoder (respectively jpeg-decoder) application. The DCT function is mainly composed of two functions: horizontal DCT (noted HDCT) and vertical DCT (noted VDCT). Each function consumes almost 20% of the whole execution time. The IDCT function is composed of horizontal IDCT (IHDCT) and vertical IDCT (IVDCT). Each function consumes almost 15% of the jpeg-decoder execution time. The 2D-DCT and 2D-IDCT computations are detailed in equations 5.3 and 5.4.

a- HDCT function

```
// For i fixed, we need 8 values of f [i ][ k ]
and64 values of Mt[k][j]

for ( i = 0 ; i < N ; i++ )
{
for ( j = 0 ; j < N ; j++ )
 {
       temp[ i ][ j ] = 0.0;
          For ( k = 0 ; k < N ; k++ )
Temp[ i ][ j ] += ( (int) f[ i ][ k ] - 128 ) * Mt[ k ][ j ]; } }
```

b- VDCT function

```
//For i fixed, we need 64 values of temp[ k][ j ]
 and  8 values of M[i][k]
for ( i = 0 ; i < N ; i++ )
{
 for ( j = 0 ; j < N ; j++ )
  {
       temp1 = 0.0;
        for ( k = 0 ; k < N ; k++ )
           temp1 += M[ i ][ k ] * temp[ k ][ j ];
            F[ i ][ j ] = ROUND( temp1 ); } } }
```

c- IHDCT function

```
// For i fixed, we need 8 values of F i ][ k ]
and 64 values of M[k][j]
for ( i = 0 ; i < N ; i++ )
{
for ( j = 0 ; j < N ; j++ )
 {
       temp[ i ][ j ] = 0.0;
       for ( k = 0 ; k < N ; k++ )
           temp[ i ][ j ] +=F[ i ][ k ] * M[ k ][ j ];} }
```

e- IVDCT function

```
//For i fixed, we need 64 values of  temp[ k][ j ]
and 8 values of Mt[i][k]
for ( i = 0 ; i < N ; i++ )
{
 for ( j = 0 ; j < N ; j++ )
 {
      for ( k = 0 ; k < N ; k++ )
       temp1 += Mt[ i ][ k ] * temp[ k ][ j ];

      f[ i ][ j ] = ROUND( temp1 ); } } }
```

FIGURE 5.16: DCT and IDCT functional decomposition

$$F(u,v) = \frac{1}{\sqrt{2N}} * C(u) * C(v) \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} f(x,y) * \cos \frac{(2x+1) * u * \pi}{2N} * \cos \frac{(2y+1) * v * \pi}{2N} \tag{5.3}$$

$$f(x,y) = \frac{1}{\sqrt{2N}} \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} C(u) * C(v) F(u,v) * \cos \frac{(2x+1) * u * \pi}{2N} * \cos \frac{(2y+1) * v * \pi}{2N} \tag{5.4}$$

Where

$$C(u) = \begin{cases} \frac{1}{\sqrt{2N}} & \text{if} \quad u = 0 \\ 0 & \text{if} \quad u \geq 0 \end{cases} \tag{5.5}$$

The 2D-DCT and 2D-IDCT are computed using the separability property of this transform. This means that F(u,v) and f(x,y) can be computed in two separate steps. Each 2-D transform (forward or inverse) is divided in two 1-D transform. The separated transformations can also be expressed in matrix operations (Equations 5.6 and 5.7). Figure 5.16 details DCT and IDCT computations.

$$F = T * f * T^t \tag{5.6}$$

$$f = T^t * F * T \tag{5.7}$$

Where

$$T_(i,j) = \begin{cases} \frac{1}{\sqrt{N}} & \text{if} \quad i = 0 \\ \frac{2}{\sqrt{N}} * \cos \frac{(2j+1)i\pi}{2N} & \text{if} \quad i > 0 \end{cases} \tag{5.8}$$

For each function of Figure 5.16, the computational parts are highlighted to be implemented as custom instructions. The HDCT and IHDCT parts consist on multiplication of 1*8 matrix with an 8*8 matrix while the VDCT and IVDCT parts consist on a multiplication of 8*8 matrix with an 8*1 matrix. Thus, we observe that we can only associate one pattern for HDCT and IHDCT tasks and another pattern for VDCT and IVDCT tasks.

The HDCT/IHDCT and VDCT/IVDCT patterns consume respectively 28 and 26 area units when implemented on a dedicated HW accelerator. In these experiments an area unit corresponds to 70 slices on the FPGA.
The jpeg encoder/decoder applications will be explored on a 8-processor MPSoC architecture, in which four processors compute the encoder application while the four others execute the decoder application. The optimal AHt-MPSoC configuration will be selected through the MILP-based exploration process. The exploration finds out the sharing degree of each pattern to minimize the consumed area while respecting the performance constraint. Thus, the HDCT/IHDCT and VDCT/IVDCT patterns information are inserted into the MILP model. For each processor, the performance constraint is set to ensure the 20 images/second requirement of the jpeg-codec application.
Figure 5.17 shows the area usage of the MPSoC configurations using HW accelerators when varying the performance constraint. For the first five points, the model generates the same solution with a minimum area usage. These points only integrate one pattern ($x_1 = 1$ and $x_2 = 0$) and correspond to a fully shared HDCT/IHDCT HW accelerator between processors. For a slight increase in speed up, the fully shared configuration can

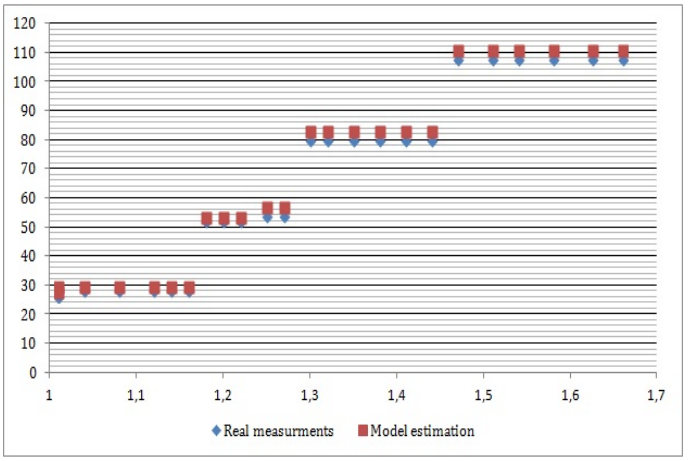

FIGURE 5.17: Area usage (y axis) of the MILP model for the generated configurations for different speed-ups. The speed-ups are calculated relative to the configuration with 8 processors without HW accelerators.

no more satisfy the performance constraint. This is mainly due to delay and conflicts to access shared HW accelerators. In Figure 5.17, for speeds-up equal to 1.18, 1.2 and 1.22, the model generates solutions with a 4-shared HW accelerators. These solutions consume 52 area units and integrate two VDCT/IVDCT HW accelerators. The first one is shared between P1, P3, P6 and P8 processors while the second is shared between P2, P4, P5 and P7.

For a speed-up greater than 1.22, configurations with only one pattern can no more satisfy the required speed-up. For example, in Figure 5.17, for speed-ups of 1.25 and 1.27, the MILP generated solutions integrate HDCT/IHDCT and VDCT/IVDCT patterns

($x_1 = 1$ and $x_2 = 1$). These solutions consume 54 area units and integrate a fully shared HW accelerator for HDCT/IHDCT pattern and a fully shared HW accelerator for VDCT/IVDCT pattern. Table 5.7 summarizes the presented results and highlights

TABLE 5.7: Generated configurations for different speed-ups (S)

	$1.02 \leq S \leq 1.16$	$1.17 \leq S \leq 1.22$	$\leq S \leq 1.45$	$S \geq 1.47$
Area cost	28	52	80	108
x[m]	{10}	{01}	{11}	{11}
nb of HW acceleratorsfor T1, nb of HW accelerators for T2	1,0	0,2	1,2	2,2

the variation of the area cost and the number of integrated HW accelerators to provide the required speed-ups 5.17.

In Figure 5.17, we note a slight overhead between our MILP results and real measurements. As mentioned in 5.4.1, this is due to bridge between processors and shared HW accelerators.
In order to evaluate the area efficiency of our proposed AHt-MPSoC configurations, we compare different model-based solutions to SHt-MPSoC architectures with only private HW accelerators. In Figure 5.17, the AHt-MPSoC configuration that provides a speed-up equal to 1.22 consumes 52 area units. Whereas, without hardware sharing, this speed-up is provided with a fully private SHt-MPSoC architecture that integrates VDCT/IVDCT HW accelerators and reaches 208 units of area usage. Also without hardware sharing, the maximum speed-up requires 432 area units to map HDCT/IHDCT and VDCT/IVDCT patterns on HW accelerators for all the processors. In Figure 5.17, for this speed-up, our model generates an AHt-MPSoC configuration that consumes 108 area units.

5.5 Conclusion

The experiments presented in this chapter have demonstrated that the proposed HW accelerators sharing methodology allows an intelligent usage of available hardware resources. Our approach has been applied and tested with jpeg encoder application in an FPGA emulation platform. The FPGA prototyping allows measuring the performance gain/area usage trade-off of Ht-MPSoC configurations with shared HW accelerators in comparison to private ones. When HW sharing is disabled, the implementation of two types of HW accelerators was not possible. However, configurations that enable HW accelerators sharing integrate more computational tasks as HW accelerators. In addition, the implementation of different configurations with different sharing degree has demonstrated that increasing the level of sharing preserves roughly the same speed up

as private configurations.

In order to evaluate our technique proposed to select the HW accelerators for a Ht-MPSoC architecture two case-studies have been proposed. The first one is based on synthetic applications and the second one is based on jpeg codec applications. The experiments have shown that our technique allows a rapid selection of Ht-MPSoC configurations. Cplex was able to generate solutions in seconds. The Comparison of area consumption and execution time of generated solutions to real measurements on FPGA show the accuracy of proposed technique.

Chapter 6

Conclusion

Although processor performance is paramount for high performance-computing, embedded systems have additional requirements, namely the minimization of area overheads in cost and power. Moreover, an embedded system is a specialized computing system for an application domain. Driven by the same physical laws, every application domain shapes and sizes the computing systems under different goals and constraints. Thus, every application domain has to adjust their embedded systems in order to improve the area overhead of every new generation of products while meeting particular constraints. Multimedia embedded systems are widely used in many areas to provide information service in applications, such as teleconferences, distant learning, movies, and video games. These systems require the processing of signal, image, and video data streams under an execution time constraint. Moreover, these systems require low power and area costs. In this context, Ht-MPSoC architecture is a promising computing system. In such architecture, customization leads to more efficient designs, as resources are consumed to meet the exact requirements of the application. On the other hand, parallelization distributes the computation amongst several processors.

The integration of HW accelerators represents an alternative to customize a processor by providing a hardware execution that exploits the exact level of instruction-level parallelism of a particular computational task. HW accelerators have been used in multimedia embedded systems because they allow to exploit parallelization and reduce power consumption.

Thus, the integration of HW accelerators play an important role in the design of high-performance, energy efficient Ht-MPSoC architectures for multimedia domain. However, this solution is still regarded as an expensive design decision, as area costs are high, and the performance/cost trade-off is complex .

This thesis presented a technique for optimising the area cost of Ht-MPSoC architectures while satisfying performance constraints. Our technique integrates a HW accelerators sharing methodology. This methodology enables two or more processors to share hardware accelerators to execute their similar tasks. Our methodology is motivated by the fact that multimedia applications contain a large number of similar frequently used kernels. A naive exploitation of the available hardware resources implements separate private HW accelerators for different processors to provide the same computations. Previously, proposed resource-sharing techniques share static resources [93] [94] [95] or run-time reconfigurable resources [85] [84] amongst custom functional units that are tightly coupled to the processor data path. However, for multimedia application, loosely coupled hardware accelerators are more suited than tightly coupled ones [133]. Hence, resource sharing for loosely coupled HW accelerators is essential in order to optimise multimedia embedded systems. While executing the Jpeg-encoder application , the proposed HW accelerators sharing methodology achieves an area saving that reaches 50% for a 4-microblazes architecture without impacting the performances.

Our technique is also based on MILP model that is able to quickly explore the design space of optimal trade-off solutions. The search of the trade-off aims to find the right balance between HW accelerators sharing and execution time overhead. The solutions with the optimal trade-off are found by guiding the selection process to favour HW accelerators sharing between tasks that are likely to be executed in a multiplexed manner with low performances losses. This is achieved by using variables that quantify the execution time delay. The search of the design is based on real measurements of area cost and execution time gain of each HW accelerator. This allows our technique to be accurate. The comparison of the area consumptions and performances of generated solution and real FPGA measurements for the jpeg-codec applications shows an area and performance overheads which are respectively below 5% and 2%. Chapter 3 has presented the hardware accelerators sharing methodology for multimedia applications. We presented also the proposed SHt-MPSoC and AHt-MPSoC architectures and their interconnection network. This latter consists of hierarchical buses interconnected through bridges. The choice of hierarchical buses was made for two reasons: the practicality and the performance/energy optimizations. However, such interconnection would be complex and less efficient when the number of processors increases. Thus, a study of performance/energy trade-offs of SHt-MPSoC and AHt-MPSoC architectures with different interconnection networks would be rewarding.

Chapter 4 detailed the proposed technique for the selection of private and/or shared HW accelerators for SHt-MPSoC and AHt-MPSoC architectures. For our technique, the identification of patterns that are candidates for hardware customization is a manual process. This process can be automated by using existing techniques to find maximal common subgraph of different tasks.

Chapter 5 uses two real applications, namely JPEG encoding and JPEG decoding, and synthetic applications in order to demonstrate the benefits of the proposed contributions. The experiments performed were considered sufficient to prove the benefits of HW accelerators sharing methodology as well as the accuracy of the MILP models.

There are many interesting extensions that can be made to the HW accelerators-sharing technique proposed in this thesis. Some of the possible future research perspectives include:

- Power management for the SHt-MPSoC and AHt-MPSoC architectures. Although there is not an explicit relationship between the level of HW accelerators sharing and the resulting power consumption of the architecture, observations highlighted upon experiments could be used to derive power-models. Based on these models, the search of the design space can be guided towards energy-efficient solutions.
- In order to minimize the design time of efficient SHt-MPSoC and AHt-MPSoC architectures, the process for the identification of common tasks, to construct an automated framework, has to be automated. The identification of maximal common sub-graphs based on maximal cliques technique can be used [134].
- The support of dynamically reconfigurable SHt-MPSoC architectures and AHt-MPSoC architectures is another interesting research axis. In fact, some multimedia applications require run-time adaptation. This means that some of the computational tasks of the system may depend on varying conditions imposed by the application or by the user.

Bibliography

[1] J. Haris and P. Sri. *Pipelined Multiprocessor System-on-Chip for Multimedia*, chapter Intruduction. Springer, 2014.

[2] J. Turley. Survey says: Software tools more important than chips. *Embedded Systems Design*, 2005.

[3] Altera. Nios II Classic Processor Reference Guide, .

[4] Opencores. Openrisc 1200 Ip Core Specification, 2011.

[5] *MicroBlaze Processor Reference Guide.*

[6] Aeroflex. Leon3 processor, 2015. URL http://www.gaisler.com/index.php/products/processors/leon3.

[7] *ML505/ML506/ML507 Evaluation Platform User Guide.*

[8] *PLB v3.4 and OPB to PLB v4.6 System and Core Migration User Guide.*

[9] Shafique M. *Architectures for adaptive low-power embedded multimedia systems.* PhD thesis, Karlsruhe Institute of Technology, Germany, 2011.

[10] STMicroelectronics. Stmicroelectronics unveils new nomadik(tm) processor for next-generation mobile multimedia applications. Technical report, STMicroelectronics, 2013.

[11] K. Konin. Nexperia pnx7850. Technical report, Philips Semiconductors, 2002.

[12] W. Wolf, A. Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(10), Oct 2008.

[13] A. Kumar, H. Corporaal, B. Mesman, and Y. Ha. Trends and challenges in multimedia systems. In *Multimedia Multiprocessor Systems*, Embedded Systems. Springer Netherlands, 2010.

[14] A. Nery, N. Nedjah, F. França, L. Jozwiak, and H. Corporaal. A reconfigurable ray-tracing multi-processor soc with hardware replication-aware instruction set extension. In Springer, editor, *Algorithms and Architectures for Parallel Processing*, volume 8285 of *Lecture Notes in Computer Science*. 2013.

[15] F. Luís. A survey on operating system support for embedded systems properties. In *Workshop of Operating Systems, 2009*.

[16] F. Dahlgren. Future mobile phones - complex design challenges from an embedded systems perspective. In *Proceedings of the Seventh International Conference on Engineering of Complex Computer Systems (ICECCS'01)*.

[17] P. Koopman. Embedded system design issues (the rest of the story). In *Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*, ICCD '96, 1996.

[18] R. Ubal, J. Sahuquillo, S. Petit, H. Hassan, and P. López. Power reduction in advanced embedded IPC processors. *Intelligent Automation & Soft Computing*, 15(3):495–507, 2009.

[19] C. Timm and A. Gelenberg. Reducing the energy consumption of embedded systems by integrating general purpose gpus. Technical report, Dortmund University, 2010.

[20] Sh. Mu, C. Wang, M. Liu, D. Li, M. Zhu, X. Chen, X. Xie, and Y. Deng. Evaluating the potential of graphics processors for high performance embedded computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011.

[21] O. Mencer, M. Morf, and M. Flynn. Hardware software tri-design of encryption for mobile communication units. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, May 1998.

[22] O. Mencer, M. Morf, and M. Flynn. Hardware software tri-design of encryption for mobile communication units. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 5, May 1998.

[23] .

[24] REASON. A Modern Hdl-Based Diesgn Flow for Fpga Prototyping of Asics, 2000.

[25] F. Nilson. Fpga vs. asic design flow. Technical report, Xilinx, 2010.

[26] K. Ian, T. Russell, and R. Jonathan. Fpga architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2008.

[27] Kuon Ian and Rose Jonathan. Measuring the gap between fpgas and asics. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2007.

[28] Clive Maxfield. What's the number of asic versus fpga design starts?, 2011. URL http://www.eetimes.com/document.asp?doc_id=1278646.

[29] Copyright 2015 Xilinx Inc. Xilinx all programmable, 2015. URL http://www.xilinx.com/.

[30] wikipedia. Mpsoc, 2015. URL http://en.wikipedia.org/wiki/MPSoC.

[31] N. Maik, B. Herbert, and E. Henri. Supporting reconfigurable parallel multimedia applications. In *Proceedings of the Euro-Par 2008 conference*.

[32] wikipedia. Apple a5x, 2014. URL http://en.wikipedia.org/wiki/Apple_A5X.

[33] L.Carro and M. Beck Rutzig. *Handbook of Processing Systems*, chapter Multicore System On chip. Springer, 2013.

[34] L.Torres, P. Benoit, G. Sassatelli, M.Robertand F. Clermidy, and D. Puschini. *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*, chapter An Intoduction to Multi-core System on Chip - Trendd and Challenges. Springer, 2013.

[35] D. Taho, Jaime J., Jose L. Martin, Unai B., and Armando A. Reconfigurable multiprocessor systems: A review. *International Journal of Reconfigurable Computing*, 2010.

[36] Wikipedia. Symmetric multiprocessor system, 2014. URL http://en.wikipedia.org/wiki/Symmetric_multiprocessor_system.

[37] Lionel T., Pascal B., Gilles S., Michel R, Fabien C., and Diego P. An introduction to multi-core system on chip – trends and challenges. In Springer, editor, *Multiprocessor System-on-Chip*. Springer New York, 2011.

[38] H. Yue, Z. Wang, and K. Dai. A heterogeneous embedded mpsoc for multimedia applications. In Springer, editor, *High Performance Computing and Communications*, volume 4208 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006.

[39]

[40] A. Venkat and D. Tullsen. Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14. IEEE Press, 2014.

[41] M. Pratyusa and Vyas S. Vector processors. Technical report, Carnegie Mellon University, School of Computer Science, 2012.

[42] Altera. Stratix iv fpgas: The world's highest density 40-nm fpga. Technical report, ALTERA, 2015.

[43] Xilinx. Leading fpga system performance and capacity. Technical report, XILINX, 2015.

[44] J. Anderson, W. Qiang, and C. Ravishankar. Raising fpga logic density through synthesis-inspired architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2012.

[45] S. Yerneni. Cray x1e architecture overview. Technical report, Hinditron Cray Supercomputers, 2006.

[46] *Introduction to the Altera Nios II Soft Processor.*

[47] J. Bennett. Softcores for fpga: the free and open source alternatives, 2013. URL http://www.embecosm.com/2013/11/20/softcores-for-fpga-the-free-and-open-source-alternatives/.

[48] *Introduction to the Altera Nios II Soft Processor*, .

[49] Wikipedia. Harvard architecture, 2015. URL http://en.wikipedia.org/wiki/Harvard_architecture.

[50] *LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c).*

[51] *The SPARC Architecture Manual.*

[52] T. Blank. A survey of hardware accelerators used in computer-aided design. *Design Test of Computers, IEEE*, 1(3), Aug 1984.

[53] A.M.Tyrrell N.J.Howard and N.M.Allison. The use of field-programmable gate arrays for the hardware acceleration of design automation tasks. In *The International conference of VLSI DESIGN, 1996, Vol. 4, No. 2, pp. 135-139.*

[54] G.Wei B.Reagen, Y.S.Shao and Da.Brooks. Quantifying acceleration: Power/performance trade-offs of application kernels in hardware. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2013.*

[55] Altera. Hardware acceleration and coprocessing, July 2011. URL http://www.altera.com.cn/literature/hb/nios2/edh_ed51006.pdf.

[56] A. Gangwar A. JeetSingh, A. Chhabra and B.Dwivedi. Soc synthesis with automatic hardware software interface generation. In *Proceedings of the 16th International Conference on VLSI Design, 2003.*

[57] S. Lonardi S. Sirowy, Y.Wu and F. Vahid. Two-level microprocessor-accelerator partitioning. In *Proceedings of the Conference Design, Automation and Test in Europe, 2007.*

[58] Freescale-Semiconductor. i.mx35 applications processors for automotive products. Technical report, Freescale Semiconductor, 2012.

[59] Samsung. Samsung s3c6400 mobile processor. Technical report, Samsung, 2007.

[60] Arm cortex. arm the architecture for the digital world. URL 2014http://www.arm.com/products/processors/index.php.

[61] S. Hovsmith. Getting started with multicore programming. Technical report, CriticalBlue, 2008.

[62] R. Sumit and C. Vipin. Design issues for a high-performance distributed shared memory on symmetrical multiprocessor clusters. In *Proceeding of the High Performance Distributed Computing Conference(HPDC'98).*

[63] G.L. Hyung, C. Naehyuck, and O. Umit Y. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2007.

[64] K. Atasu, L. Pozzi, and P. Ienne. Automatic application-specific instruction-set extensions under microarchitectural constraints. In *Design Automation Conference, 2003. Proceedings*, June 2003.

[65] K. Martin, C. Wolinski, K.f Kuchcinski, A. Floch, and F. Charot. Constraint-driven identification of application specific instructions in the durase system. In Koen Bertels, Nikitas Dimopoulos, Cristina Silvano, and Stephan Wong, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, volume 5657 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009.

[66] L. Pozzi, K. Atasu, and P. Ienne. Exact and approximate algorithms for the extension of embedded processor instruction sets. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(7), July 2006.

[67] C. Ozturan, G. Dundar, and K. Atasu. An integer linear programming approach for identifying instruction-set extensions. In *Hardware/Software Codesign and*

System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on, Sept 2005.

[68] K. Atasu, O. Mencer, W. Luk, C. Ozturan, and G. Dundar. Fast custom instruction identification by convex subgraph enumeration. In *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on*, July 2008.

[69] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, 2001.

[70] Z.A. Ye, A. Moshovos, S. Hauck, and P. Banerjee. Chimaera: a high-performance architecture with a tightly-coupled reconfigurable functional unit. In *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, 2000.

[71] T.J. Callahan, J.R. Hauser, and J. Wawrzynek. The garp architecture and c compiler. *Computer*, 33(4), Apr 2000.

[72] R.E. Gonzalez. A software-configurable processor architecture. *Micro, IEEE*, 26, Sept 2006.

[73] P.G. Sassone and D.S. Wills. Dynamic strands: Collapsing speculative dependence chains for reducing pipeline communication. In *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on*, Dec 2004.

[74] S. Yehia and O. Temam. From sequences of dependent instructions to functions: an approach for improving performance without ilp or speculation. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, June 2004.

[75] N. Clark, J. Blome, M. Chu, S. Mahlke, S. Biles, and K. Flautner. An architecture framework for transparent instruction set customization in embedded processors. In *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, pages 272–283, June 2005.

[76] N. Clark, M. Kudlur, Hyunchul Park, S. Mahlke, and K. Flautner. Application-specific processing on a general-purpose core via transparent instruction set customization. In *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on*, pages 30–40, Dec 2004.

[77] L. Bauer, M. Shafique, S. Kramer, and J. Henkel. Rispp: Rotating instruction set processing platform. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, June 2007.

[78] N. Paulino, J.C. Ferreira, J. Bispo, and J.M.P. Cardoso. Transparent acceleration of program execution using reconfigurable hardware. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, 2015.

[79] N. Clark, M. Kudlur, Hyunchul Park, S. Mahlke, and K. Flautner. Application-specific processing on a general-purpose core via transparent instruction set customization. In *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on*, Dec 2004.

[80] N. Clark, J. Blome, M. Chu, S. Mahlke, S. Biles, and K. Flautner. An architecture framework for transparent instruction set customization in embedded processors. In *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, June 2005.

[81] V. Govindaraju, Chen-Han Ho, and K. Sankaralingam. Dynamically specialized datapaths for energy efficient computing. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, Feb 2011.

[82] L. Torres, P. Benoit, G. Sassatelli, M. Robert, F Clermidy, and D. Puschini. An introduction to multi-core system on chip – trends and challenges. In Springer, editor, *Multiprocessor System-on-Chip*. Springer New York, 2011.

[83] R.E. Gonzalez. Xtensa: a configurable and extensible processor. *Micro, IEEE*, 20 (2), Mar 2000.

[84] M.A. Watkins and D.H. Albonesi. Remap: A reconfigurable heterogeneous multicore architecture. In *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, Dec 2010.

[85] Liang C. and T. Mitra. Shared reconfigurable fabric for multi-core customization. In *Design Automation Conference (DAC), 48th ACM/EDAC/IEEE*, 2011.

[86] R.E. Gonzalez. Xtensa: a configurable and extensible processor. *Micro, IEEE*, 20 (2), Mar 2000.

[87] K. Van Rompaey, D. Verkest, I. Bolsens, and H. De Man. Coware-a design environment for heterogeneous hardware/software systems. In *Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European*, Sep 1996.

[88] F. Sun, S. Ravi, A. Raghunathan, and N.K. Jha. Application-specific heterogeneous multiprocessor synthesis using extensible processors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(9), Sept 2006.

[89] M. Schwiegershausen and P. Pirsch. A formal approach for the optimization of heterogeneous multiprocessors for complex image processing schemes. In *Design Automation Conference, 1995, with EURO-VHDL, Proceedings EURO-DAC '95., European*, 1995.

[90] R. Leupers, K. Karuri, S. Kraemer, and M. Pandey. A design flow for configurable embedded processors based on optimized instruction set extension synthesis. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, March 2006.

[91] C.-L. Sotiropoulou and S. Nikolaidis. Ilp formulation for hybrid fpga mpsocs optimizing performance, area and memory usage. In *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*, Dec 2011.

[92] S. Lin Shee, A. Erdos, and S. Parameswaran. Heterogeneous multiprocessor implementations for jpeg:: a case study. In *Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference*, Oct 2006.

[93] P. Brisk, A. Kaplan, and M. Sarrafzadeh. Area-efficient instruction set synthesis for reconfigurable system-on-chip designs. In *Proceedings of the 41st Annual Design Automation Conference*, 2004.

[94] M. Zuluaga and N. Topham. Design-space exploration of resource-sharing solutions for custom instruction set extensions. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28(12), December 2009.

[95] M. Stojilovic, D. Novo, L. Saranovac, P. Brisk, and P. Ienne. Selective flexibility: Creating domain-specific reconfigurable arrays. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(5):681–694, May 2013.

[96] M.A. Watkins and D.H. Albonesi. Remap: A reconfigurable heterogeneous multicore architecture. In *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, Dec 2010.

[97] L. Chen, N. Boichat, and T. Mitra. Customized mpsoc synthesis for task sequence. In *Proceedings of the 2011 IEEE 9th Symposium on Application Specific Processors (SASP '11)*, 2011.

[98] P. Festa. A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. In *Transparent Optical Networks (ICTON), 2014 16th International Conference on*, July 2014.

[99] N. Bình, M. Imai, A. Shiomi, and N. Hikichi. A hardware/software partitioning algorithm for designing pipelined asips with least gate counts. In *Proceedings of the 33rd Annual Design Automation Conference*, DAC '96, New York, NY, USA, 1996. ACM.

[100] A. Shrivastava, H. Kumar, S. Kapoor, S. Kumar, and M. Balakrishnan. Optimal hardware/software partitioning for concurrent specification using dynamic programming. In *VLSI Design, 2000. Thirteenth International Conference on*, 2000.

[101] P. Arato, S. Juhasz, Z.A. Mann, A. Orban, and D. Papp. Hardware-software partitioning in embedded system design. In *Intelligent Signal Processing, 2003 IEEE International Symposium on*, Sept 2003.

[102] R. Corvino, A. Gamatié, M., and Lech Józwiak. Design space exploration in application-specific hardware synthesis for multiple communicating nested loops. In *2012 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XII, Samos, Greece, July 16-19, 2012*, 2012.

[103] W. Jigang, Q. Sun, and T. Srikanthan. Multiple-choice hardware/software partitioning: Computing model and algorithms. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, April 2010.

[104] G. Stitt. Hardware/software partitioning with multi-version implementation exploration. In *Proceedings of the 18th ACM Great Lakes Symposium on VLSI*, GLSVLSI '08. ACM, 2008.

[105] J. He, D. Van Hung, P. Geguang, Q. Zongyan, and Y. Wang. Exploring optimal solution to hardware/software partitioning for synchronous model. *Formal Aspects of Computing*, 17(4), 2005. ISSN 0934-5043.

[106] P. Eles, K. Kuchcinski, Z. Peng, and A. Doboli. Hardware/software partitioning of vhdl system specifications. In *Proceedings of the Conference on European Design Automation*, EURO-DAC '96/EURO-VHDL '96. IEEE Computer Society Press, 1996.

[107] J. Henkel and R. Ernst. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. *IEEE Trans. Very Large Scale Integr. Syst.*, 9(2), April 2001.

[108] M. López-Vallejo and J.C. López. On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Trans. Des. Autom. Electron. Syst.*, 8(3), July 2003.

[109] K.S. Chatha and R. Vemurl. Magellan: multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs. In *Hardware/Software Codesign, 2001. CODES 2001. Proceedings of the Ninth International Symposium on*, 2001.

[110] H. Han, W. Liu, J. Wu, and G. Jiang. Efficient algorithm for hardware/-software partitioning and scheduling on mpsoc. *Journal of Computers*, 8 (1), 2013. URL http://ojs.academypublisher.com/index.php/jcp/article/view/jcp08016168.

[111] J. Srinivasan. An overview of static power dissipation. Technical report, University of Illinois, 2011.

[112] S. Imai. Task offloading between smartphones and distributed computational resources. Master's thesis, Rensselaer Polytechnic Institute Troy, New York, 2012.

[113] *Describing Synthesizable RTL in SystemC*.

[114] F. Antoine and R. Tanguy. Master interface for on-chip hardware accelerator burst communications. *Journal of VLSI Signal Processing*, 2005.

[115] Xilinx Inc. Selecting intellectual property interface services, 2008. URL http://www.xilinx.com/itp/xilinx10/help/platform_studio/ps_c_ipw_selecting_ipif_services.htm.

[116] Xilinx. *Zynq-7000 All Programmable SoC Technical Reference Manual*. URL http://www.xilinx.com/support/documentation/user_guides/ug-585-Zynq-7000-TRM.pdf.

[117] Altera. *Cyclone V*, . URL http://www.altera.com/devices/fpga/cyclone-v-fpgas/cyv-index.jsp.

[118] Micro-Semi. *Smart Fusion 2*. URL http://www.microsemi.com/products/fpga-soc/soc-fpga/smartfusion2.

[119] *An Overview of On-Chip Buses*, chapter Ser. Elect. and Energet. FUniversity of Nic´, 2006.

[120] D. Reetuparn, E. Soumya, K. M. Asit, N. Vijaykrishnan, and R. D. Chita. Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps. In *Proceedings of the 2009 IEEE 15th Intern. Symp. on High Performance Computer Architecture (HPCA'09)*.

[121] xilinx. Logicore ip plbv46 to plbv46 bridge. Technical report, XILINX, 2011.

[122] J. G. Tong and M. A. S. Khalid. Profiling cad tools: A proposed classification. In *The 19th International Conference on Microelectronics, December 2007*.

[123] J. G. Tong and M. Khalid. Profiling tools for fpga-based embedded systems: Survey and quantitative comparison. *Journal of Computer*, June 2008.

[124] Rajendra P. and Arvind R. A survey of embedded software profiling methodologies. *International Journal of Embedded Systems and Applications (IJESA)*, Decembre 2011.

[125] H. Bunke, G. Guidobaldi, and Vento M. Weighted minimum common supergraph for cluster representation. In *International Conference on Image Processing, ICIP 2003*.

[126] F. Mahmood, F. Mohammad, Z. Mahdy, and Z. Ali. A new datapath merging method for reconfigurable system. In *5th International Workshop on Applied Reconfigurable Computing (ARC'09)*.

[127] xlinx. Xilinx all programmable, 2015. URL http://www.xilinx.com/.

[128] STAFF WRITER. Sourcetech411, 2013. URL http://sourcetech411.com/2013/04/top-fpga-companies-for-2013/.

[129] *Data2MEM User Guide*.

[130] *Microblaze Softcore and Digilent S3 FPGA Demonstration Board*.

[131] J. Ahmad, K. Raza, M. Ebrahim, and U. Talha. Fpga based implementation of baseline jpeg decoder. In *Proceedings of the 7th International Conference on Frontiers of Information Technology*, FIT '09, pages 29:1–29:6. ACM, 2009.

[132] M. Krepa. Jpeg encoder : Overview, 2012. URL http://opencores.org/project,mkjpeg.

[133] E.G. Cota, P. Mantovani, G. Di Guglielmo, and L.P. Carloni. An analysis of accelerator coupling in heterogeneous architectures. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6, June 2015. doi: 10.1145/2744769.2744794.

[134] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1–2):1 – 30, 2001. doi: http://dx.doi.org/10.1016/S0304-3975(00)00286-3.