Portable infrastructure for heterogeneous reconfigurable devices in a cloud-FPGA environment

Arief Wicaksana

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Nano Electronique et Nano Technologies (NENT)**

Arrêté ministériel : 25 mai 2016

Présentée par

Arief WICAKSANA

Thèse dirigée par **Frédéric ROUSSEAU,** Professeur**,** UGA
et codirigée par **Arif SASONGKO,** Professeur associé, ITB

préparée au sein du **Laboratoire Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés (TIMA)**
dans **l'École Doctorale Electronique, Electrotechnique, Automatique et Traitement du Signal (EEATS)**

Infrastructure portable pour un système hétérogène reconfigurable dans un environnement de cloud-FPGA

Portable infrastructure for heterogeneous reconfigurable devices in a cloud-FPGA environment

Thèse soutenue publiquement le **2 octobre 2018**,
devant le jury composé de :

Monsieur Christophe JEGO
Professeur, Institut Polytechnique Bordeaux, Président
Monsieur Kenneth Kent
Professeur, Université Rudgers New Brunswick, Rapporteur
Monsieur Loïc LAGADEC
Professeur, ENSTA Bretagne, Rapporteur
Monsieur Alain FONKOUA
Ingénieur, Synopsys, Examinateur
Monsieur Olivier MULLER
Maître de conférences, Grenoble INP, Examinateur
Monsieur Arif SASONGKO
Professeur associé, Institut Technologique de Bandung, Co-directeur de thèse
Monsieur Frédéric ROUSSEAU
Professeur, Université Grenoble Alpes, Directeur de thèse

Cette thèse a été co-encadrée par **Monsieur Olivier MULLER**, Maître de conférences, Grenoble INP

I'm writing my first draft and reminding myself
that I'm simply shoveling sand into a box
so that later I can build castles.

— Shannon Hale

Dedicated to my parents and brothers who always support me
in every step I take.

You don't write because you want to say something,
you write because you have something to say.

— ***F. Scott Fitzgerald***

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisors without whom it would not be possible to complete this work—Frédéric Rousseau, Olivier Muller, and Arif Sasongko. Many thanks for your guidance, advices, and time during the entire work and beyond.

I would also like to thank the reviewers, Mr. Kenneth Kent and Mr. Loïc Lagadec, for their reviews and suggestions which not only improve the quality of the presentation of this work but also provide me many insights in preparing to be a better researcher, the president of the jury, Mr. Christophe Jego, and the examiner, Mr. Alain Fonkoua, for the precious feedbacks and remarks during the presentation of the defense.

Special thanks to Alban Bourge and Adrien Prost-Boucle which have inspired me in a lot of ways. I am extremely grateful for the tool support which is one of the most important parts of this work. Thank you for the help and suggestions during the development and the experiments.

Many thanks to Robin Rolland-Girod which has provided help and assistance for the experiments that used various FPGA boards. Thank you for giving the technical support necessary to complete this work.

Many thanks to the other members of SLS team of TIMA Laboratory: Frédéric Petrot, Liliana Andrade, Luc Michel, Clément Deschamps, Antoine Faravelon, Thomas Baumela, and Georgios Christodoulis. Your presence and supports have been a great help for me in achieving better results in work.

I would also like to express my deepest gratitude to my parents and my brothers for their supports and encouragements. Thank you for always being there during wonderful as many as difficult moments.

Last but not least, many thanks to all the people who helped me in any way during my thesis.

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) have been gaining popularity as hardware accelerators in heterogeneous architectures thanks to their high performance and low energy consumption. This argument has been supported by the recent integration of FPGA devices in cloud services and data centers. The potential offered by the reconfigurable architectures can still be optimized by treating FPGAs as virtualizable resources and offering them multitasking capability. The solution to preempt a hardware task on an FPGA with the objective of context switching it has been in research for many years. The previous works mainly proposed the strategy to extract the context of a running task from the FPGA to provide the possibility of its resumption at a later time. The communication during the process, on the contrary, has not been receiving much attention.

In this work, we study the communication management of a hardware task while it is being context switched. This communication management is necessary to ensure the consistency in the communication of a task with context switch capability in a reconfigurable system. Otherwise, a hardware context switch can only be allowed under restrictive constraints which may lead to a considerable penalty in performance; context switching a task is possible after the communication flows finish and the input/output data have been consumed. Furthermore, certain techniques demand homogeneity in the platform for a hardware context switch can take place.

We present a mechanism which preserves the communication consistency during a hardware context switch in a reconfigurable architecture. The input/output communication data are managed together with the task context to ensure their integrity. The overall management of the hardware task context and communication data follows a dedicated protocol developed for heterogeneous reconfigurable architectures. This protocol thus allows a hardware context switch to take place while the task still has ongoing communication flows on Reconfigurable System-on-Chips (RSoCs). From the experiments, we discover that the overhead due to managing the communication data becomes negligible since our mechanism provides the necessary high responsiveness for preemptive scheduling, besides the consistency in communication. Finally, the applications of the proposed solution are presented in a task migration prototyping and in a hypervisor-based system.

RÉSUMÉ

La haute performance ainsi que la basse consommation d'énergie offertes par les Field-Programmable Gate Arrays (FPGAs) contribuent à leur popularité en tant qu'accélérateurs matériels. Cet argument a été confirmé par les intégrations récentes des FPGAs "dans les nuages" et les centre de données. Toutefois, le potentiel d'une architecture reconfigurable peut être encore optimisé en traitant les FPGAs comme une ressource virtualisée et en offrant un support multitâche. La solution pour interrompre une tâche sur FPGAs qui a pour objectif d'effectuer un changement de contexte matériel (hardware context switch) est un sujet de recherche depuis de nombreuses années. Les travaux précédents ont principalement proposé une stratégie pour extraire le contexte d'une tâche en cours d'exécution sur FPGA pour offrir la possibilité de sa reprise plus tard. Cependant, la gestion des données de communication tout au long du processus n'a pas reçu autant d'attention.

Dans cette thèse, nous étudions la gestion de communication d'une tâche matérielle durant son changement de contexte. Cette gestion de communication est nécessaire pour garantir la cohérence de la communication d'une tâche dans un système reconfigurable avec la capacité de changement de contexte. Autrement, un changement de contexte matériel est seulement autorisé sous des contraintes restrictives; quand toutes les données d'entrées/sorties ont été consommées et que le flux de communication est interrompu. De plus, certaines techniques demandent l'homogénéité au sein de la plate-forme pour qu'un changement de contexte matériel puisse se réaliser.

Nous présentons içi un mécanisme qui conserve la cohérence de communication durant un changement de contexte matériel dans une architecture reconfigurable. Les données de communication sont gérées avec le contexte de tâche pour assurer leur intégrité. La gestion du contexte et les données de communication suivent un protocole spécifique pour des architectures hétérogènes reconfigurables. Ce protocole permet donc un changement de contexte matériel pendant que la tâche a encore des flux de communication. À partir des expérimentations, nous validons que le surcoût de la gestion de communication devient négligeable et notre mécanisme assure une grande réactivité. Enfin, les applications possibles de la solution proposée sont présentées à travers la migration de tâches matérielles et dans un système utilisant un hyperviseur.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

API	Application Programming Interface.
ASIC	Application-Specific Integrated Circuit.
AXI	Advanced eXtensible Interface.
BRAM	Block Random-Access Memory.
CGRA	Coarse-Grain Reconfigurable Array.
COTS	Commercial Off-The-Shelf.
CPU	Central Processing Unit.
DPR	Dynamic Partial Reconfiguration.
DSP	Digital Signal Processor.
FF	Flip-Flop.
FIFO	First-In First-Out.
FPGA	Field-Programmable Gate Array.
FSM	Finite-State Machine.
FSMD	Finite State Machine with Datapath.
GPU	Graphics Processing Unit.
HLS	High-Level Synthesis.
HPC	High-Performance Computing.
I/O	Input/Output.
ICAP	Internal Configuration Access Port.
IDCT	Inverse Discrete Cosine Transform.
IP	Intellectual Property.
KPN	Kahn Process Network.

LUT	Look-Up Table.
NoC	Network-on-Chip.
OS	Operating System.
OS4RS	Operating System for Reconfigurable Systems.
PCAP	Processor Configuration Access Port.
PCIe	PCI Express.
PE	Processing Element.
PLB	Processor Local Bus.
RAM	Random-Access Memory.
Reconfigurable SoC	Reconfigurable System-on-Chip.
SoC	System-on-Chip.
SRAM	Static Random-Access Memory.
UltraRAM	Ultra Random-Access Memory.
VCGRA	Virtual Coarse-Grain Reconfigurable Array.
vFPGA	virtual FPGA.

1 INTRODUCTION

Contents

1.1 BACKGROUND

More than 50 years ago, Gordon Moore made a prediction which is widely known as Moore's Law, that the transistor capacity in an integrated circuit would double every two years [Moo06]. Moore's Law became a golden rule in the electronics industry, and it has been setting the pace of our modern digital revolution. One thing that has been undeniably affected by Moore's prediction is the development of microprocessors. Since the emergence of microprocessors in the late 1970s, its performance has increased nearly 10,000 times compared to the first general-purpose electronic computer [HP11]. Another thing that has not been left unnoticed is the integration rate of microprocessor-based systems in our life. The physical size of microprocessors has been shrinking dramatically which leads to their wide utilization in many devices. In the 21st century, microprocessors can be found not only in the mainstream devices such as in desktop computers and servers, but also in the household appliances, modern vehicles, and the Internet of Things (IoT) devices.

Microprocessors in a modern computing system work with other components in the system to run a specified application. They are most definitely connected to memory elements and coprocessors which assist them in specific tasks. These coprocessors are likely to be low-energy microprocessors, Graphics Processing Units (GPUs), Application-Specific Integrated Circuit (ASIC) devices, or Field-Programmable Gate Arrays (FPGAs). The system usually takes advantage of the co-processors to run small repetitive tasks which may consume too much of microprocessor resources. Although, the co-processors can also run in parallel with microprocessors to complete critical tasks to increase the overall performance of the system.

In this thesis work, we are particularly interested in the FPGA exploitation for modern computing architectures. Since the invention of FPGAs in 1985 [Car+86], their architectures have been evolving to respond to various needs. Initially conceived for prototyping [AYS06; Li+10], FPGA development was focused on increasing the device capacity to verify and emulate digital circuits as much as possible. As the FPGA capacity and performance have been substantially increasing, there exists a growing interest in using FPGAs to speed up or to accelerate the application that runs in a computing system. As a result, FPGAs are now used while considering the other aspects such as flexibility, cost, energy consumption, etc.

The latest trend in high-performance servers for data centers present the integration of FPGAs in the cloud computing service or cloud-FPGA. Since the use of connected mobile devices has been growing rapidly, the cloud computing requires powerful computing infrastructure at the other end of the system to provide the intended service. Such demand often causes the soaring cost of developing and maintaining cloud infrastructures, and high energy consumption will be skyrocketing due to high-performance computing resources. FPGA integration in the cloud can be the solution for such problems as FPGAs have been known to offer a high performance while consuming much less energy compared to most traditional microprocessor-based systems.

In alignment with the use of FPGAs for computing acceleration in cloud infrastructure as well as to treat them as a virtualizable resource for users, FPGAs need to support multitasking [FVS15]. Consequently, the ability to interrupt a task on an FPGA whether for the purpose of migration or not, and to resume the interrupted execution, known as hardware context switch, is necessary. With the hardware context switch ability, we can share the FPGA resource with multiple users and increase the flexibility in the system management and scheduling. With this objective in our mind, we study the mechanism to enable a hardware context switch on FPGAs particularly related to the communication.

1.2 RESEARCH CONTRIBUTIONS

The purpose of this work is to enable hardware context switch operations on tasks with communication flows in Field-Programmable Gate Arrays (FPGAs). A hardware context switch support should naturally offer the ability of interrupting the flow of execution due to preemption requests and saving the current state of the preempted task. It is also worth mentioning that, as a part of the system, a hardware task performs input/output (I/O) communication with the other tasks. While the ability to save and reload the task state at preemption is essential, context switching the task must also consider the communication aspect to avoid any irreversible error in the process. We discover that the communication issues during a context switch have not been receiving much attention. For that reason, we aim to provide a communication solution which enables the hardware context switch operation. The work

presented in this thesis report contributes to the communication in the hardware context switch as follows:

- **Communication data integrity**

  When a task is context switched while it still has ongoing communication flows, the risk of losing or causing disorder in-transit communication data is present. The data integrity in communication channels needs to be maintained to enable a proper context switch.

- **Communication continuity**

  It is important to ensure the resumption of communication between tasks after a context switch occurs. It consists of disconnecting and reconnecting the communication links between tasks, including when a migration takes place.

- **System responsiveness to preemption requests**

  Managing the communication in a hardware context switch removes the constraints in the communication. Instead of delaying the process after a preemption request is received due to busy communication channels, a hardware context switch can take place immediately. This offers a positive impact to the latency in task preemption.

- **Implementation in heterogeneous reconfigurable devices**

  The solution that we propose in this work is developed to be as generic as possible to accommodate heterogeneous reconfigurable platforms. Different reconfigurable System-on-Chip (SoC) platforms are used in this work to demonstrate the genericity of our solution and to provide a prototype of task migration application.

1.3 THESIS OUTLINE

In the following chapters, we present the results of our study regarding the communication in a hardware context switch operation.

In Chapter 2, we present the problematic challenges which motivate us to carry out this work. We focus on hardware tasks that still have ongoing communication flows while being context switched. Some issues which threaten the task execution will appear if this inter-task communication is not well treated.

In Chapter 3, we present a thorough literature review on the hardware context switch topic. Methods to extract and restore task contexts from FPGAs, which are essential in providing the hardware context switch support are proposed in some notable works. Meanwhile, there exist some interesting works which discuss inter-task communication management in reconfigurable architectures. Their methods are

reviewed to see the advantage and compatibility with the hardware context switch support.

In Chapter 4, we describe the solution that we choose to overcome the challenges in communication in a hardware context switch operation. We explain the hardware context switch method which our solution is based on. Then, we describe the communication model and the necessary context switch protocol to preserve the consistency in communication. Finally, we present the implementation of our solution in reconfigurable architectures.

In Chapter 5, we present the experiments which were conducted to evaluate the performance and overhead of our solution. The platforms which are used in this work are explained in this chapter as well. We present the evaluation results on each heterogeneous platform and the applications developed using our solution.

Finally, the conclusion of this work will be drawn in Chapter 6, followed by possible extensions in future work.

2

MOTIVATION AND PROBLEM STATEMENT

THE OBJECTIVE OF THIS CHAPTER is to describe the motivation behind this work and the problems which need to be resolved. We begin by presenting the trend towards the use of computing architectures which integrate reconfigurable accelerators to increase their processing performance, known as reconfigurable computing architectures. These architectures integrate FPGA devices which have shown a potential in providing an excellent performance-to-energy ratio. To fully exploit the potential of reconfigurable accelerators, there is a need to virtualize FPGA resources by offering them a hardware context switch support, which will be described in the next part of this chapter. Although hardware context switch has been a research subject for over a decade, there still exist some challenges which need to be overcome, particularly regarding the communication of the preempted task. At the end of this chapter, we develop a problem statement in regards to the communication challenges in reconfigurable accelerators which can hinder a context switch operation.

Contents

2.1 RECONFIGURABLE COMPUTING ARCHITECTURES

2.1.1 *Hardware Acceleration*

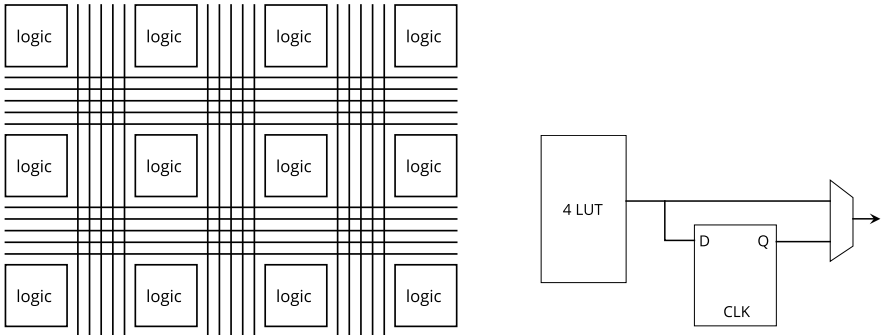
The endless demand for higher processing performance drives the evolution of modern computing architectures. As many believe, the key to obtaining faster results in computation are in parallelisms. With the energy and cost being the primary constraints, parallelizing applications in data-level and task-level is necessary [HP11]. Application parallelism is possible with the supports from adequate computer hardware peripherals in addition to existing Central Processing Units (CPUs). Provided that the scaling of supply voltage and frequency in CPUs cannot keep up with Moore's prediction [Moo06] anymore, the era of hardware acceleration has begun.

The term hardware acceleration has been used to describe a process of harnessing accelerators to speed up the computation of a CPUs in computing architectures. An accelerator is a specialized hardware unit that performs a set of tasks with higher performance or better energy efficiency than a traditional CPU [SB15]. It can be Digital Signal Processors (DSPs), Graphics Processing Units (GPUs), Application-Specific Integrated Circuits (ASICs), or reconfigurable devices such as FPGAs. In hardware acceleration, the accelerator which can be spatially exploited is dedicated to specific functions or algorithms, tasks which are repetitive and heavy when executed in CPUs. CPUs and hardware accelerators work in tandem to reduce the execution time of an application.

2.1.2 *FPGA as Reconfigurable Accelerator*

Among the types of hardware accelerators, reconfigurable devices have been attracting various communities due to its interesting characteristics. They offer performance and concurrency benefits over software, yet they can be reprogrammed cheaply and easily to provide a high flexibility in acceleration. With reconfigurable accelerators, the advantages of hardware and software can be merged into one. The idea of using reconfigurable devices to increase computing performance is not new; a structure of fixed and variable part in computing architecture was introduced in 1960 by Estrin [Est60]. Traditional CPUs and reconfigurable devices are put together in a system forming a reconfigurable computing architecture or a reconfigurable architecture for short. Through the invention of FPGAs by Xilinx in the mid-1980s [Car+86], reconfigurable accelerators have been widely utilized in many fields, e.g., image processing [Bat+02], High-Performance Computing (HPC) [VB13], neural networks [Zha+15], etc.

An FPGA is an integrated circuit which provides the possibility of digital circuit implementation after fabrication. In very general terms, FPGA resources consist of *logic* and *interconnect*. Logic is where the arithmetic or logical functions are carried out. It generally consists of Look-Up Table (LUTs) and memory elements such as



(a) Abstract overview of FPGA

(b) A simple logic block

Figure 1: The basic structure of an FPGA (Hauck and DeHon [HD10], p. 6-7)

Flip-Flops (FFs). Interconnect is the medium to forward data from one node of computation to another. The interconnects on an FPGA form a matrix structure which connects logic units through switch points. Figure 1 describes an abstract overview of FPGA architecture and a simple representation of logic block with 4-inputs LUT (4-LUT).

The logic and interconnect structures in commercial FPGAs are programmable (or configurable). Modern FPGAs use Static Random-Access Memory (SRAM) bits to hold user-defined configuration values or *bitstream* that program their logic and interconnect structures. By tiling logic blocks together and connecting them through a series of programmable interconnect, an FPGA can implement a task or function in the form of digital circuits or Intellectual Properties (IPs). The configuration of FPGAs in a reconfigurable architecture is normally performed by a CPU or a manager in another FPGA. For more detail information about architectures and programming model of FPGAs, the reader is referred to, among many others, the following literature [HD10].

In its basic configuration, a reconfigurable architecture consists of a CPU, an FPGA, and a shared memory. Such architecture is described in Figure 2. When running an application, the CPU can delegate to the FPGA heavy and repetitive parts of the application as hardware tasks or IPs. Meanwhile, the CPU can perform another execution in parallel while waiting for the execution in the FPGA finishes. The communication data and execution results can be stored in the memory to allow common access. The CPU in the architecture is normally also responsible for configuring the task to run on the FPGA.

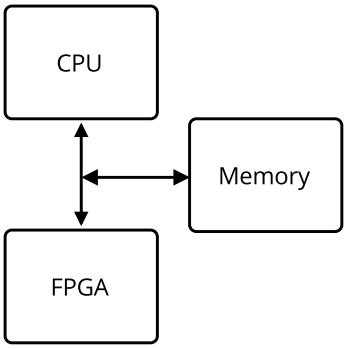


Figure 2: Basic configuration of reconfigurable architectures

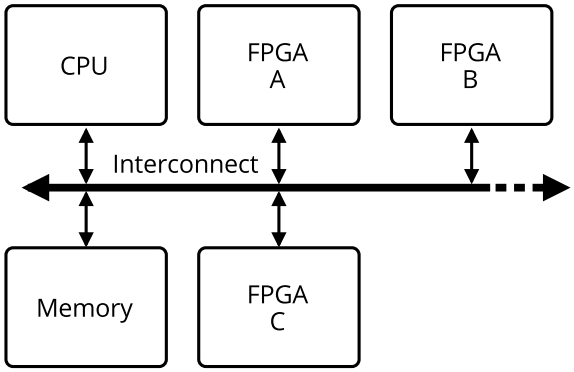


Figure 3: An illustration of multi-FPGA systems

2.1.3 *Multi-FPGA Systems*

As FPGA devices have become larger and more powerful, adding more FPGAs in reconfigurable architectures is desired to achieve higher overall application speedups and energy savings. Multi-FPGA systems are proposed to provide massive parallelism in the execution of the tasks, for instance in the HPC community. Figure 3 presents an illustration of a system with a host CPU and multiple FPGAs connected to the same interconnect. With more FPGAs in the system, more IPs can run concurrently to complete the application. It is worth mentioning that a multi-FPGA system requires a manager or administrator which manages the hardware task scheduling and allocation or reconfiguration for every FPGA in the system. It is also required to facilitate the communication between IPs, for instance to provide the arbitration. These administration purposes can be set in the host CPU of the system.

The interconnect between FPGA and CPU can be based on various standards. One of the standard that offers high-bandwidth interconnection, scalable performance, and ubiquitous presence is PCI Express (PCIe). RIFFA [Jac+15], CAPI [Stu+15], and Xillybus [Xil] are some of the solutions based on PCIe. Saldana and Chow [SC06] present an MPI-based communication protocol which is implemented

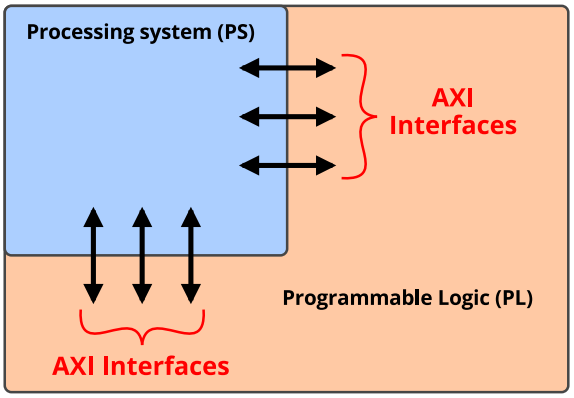


Figure 4: Abstract view of Xilinx reconfigurable SoC (Zynq)

using MultiGigabit Transceivers. Another solution which has slower communication bandwidth but requires a cheaper infrastructure is Ethernet. In [ABS10], UDP/IP packet transfers are used in communication between CPU and FPGAs.

The various communication standards shows that FPGAs are flexible enough to support the connection nearly to any device. However, the protocol standards are required to be implemented from the physical layer to support the interface to the FPGAs. The effort from designers to provide such standards is high, and sometimes it exceeds the effort required for implementing the original application. Furthermore, significant FPGA resources are often required to implement the CPU-FPGA communication. Hardware tasks allocation by reconfiguring FPGAs and communication data exchange also need to be performed by the CPU.

2.1.4 *Reconfigurable System-on-Chip*

To improve the performance as well as to reduce the overhead in the communication between CPUs and FPGAs in reconfigurable architectures, FPGA vendors provide Reconfigurable System-on-Chip (Reconfigurable SoC) in their line of products. Reconfigurable SoC integrates ASIC-based components in traditional System-on-Chip (SoC), i.e., CPUs and other peripherals, and an FPGA into one chip, e.g., Zynq FPGA from Xilinx [Cro+14] and Altera FPGA SoC family from Altera (now Intel FPGA) [Cor]. Figure 4 shows an abstract view of Reconfigurable SoC architecture from Xilinx: Zynq family. The Reconfigurable SoC architecture like Zynq consists of the fixed SoC part (CPU, memory, peripherals, etc.), which is shown in the figure as *Processing System (PS)*, and the reconfigurable part or *Programmable Logic (PL)*. The PL and PS parts are linked using industry standard Advanced eXtensible Interface (AXI) connections [Inc12].

Due to the integration of the CPU and FPGA, a system similar to the one in Figure 3 can be built with a much lower effort. Figure 5 presents CPU and hardware tasks (IPs) on FPGA built on Zynq architecture. The internal AXI interfaces

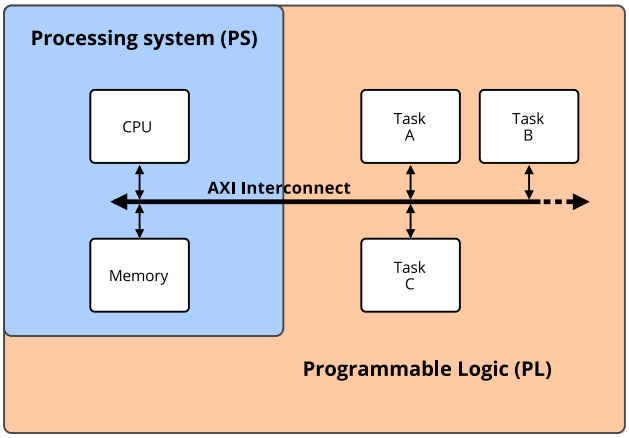


Figure 5: An illustration of hardware tasks built on Zynq architecture

require fewer resources compared to other communication standards, e.g., PCIe, and they offer higher performance than Ethernet. One or several IPs can be located on the FPGA of Reconfigurable SoC in a condition the FPGA has enough resources. The use of Reconfigurable SoC also offers a higher efficiency, particularly in the configuration and task allocation from the CPU.

2.1.5 *Communication Model and Requirements*

The traditional computing architectures follow *von Neumann* model, also known as stored-program computation model, to describe the communication between a CPU and a memory unit. In stored-program model, the memory stores both data and instructions (program), and the CPU fetches them to perform the operation. This architecture provides a very high flexibility for general-purpose use since moving from one instruction to the next can be done with ease by the CPU. With decades of progress in technology, stored-program model has evolved with some extensions in the architecture, e.g., memory-mapped I/O, cache, and scratch-pad memory.

While stored-program model remains used in today's computing architecture, it is more common to use dataflow representation to model the tasks in reconfigurable architectures. The dataflow model describes computation in terms of locally controlled events; each event corresponds to the "firing" of an actor [NLG99]. In the dataflow model, the execution of instructions is determined based on the availability of input data. As an alternative to the stored-program execution model, the dataflow model is suitable for describing a parallel execution of tasks in a system with hardware accelerators since a global synchronization between tasks are not required.

The concept of dataflow in computer architecture to exploit parallelism in programs was proposed in the early 1970s [Den74; Kah74]. In [Den74], the execution of an operation is enabled by the availability of the required input values. The completion of one operation or test makes the resulting value or decision available to

the elements of the program whose execution depends on them. Another style of the dataflow model where actors are replaced by sequential processes are proposed as Kahn Process Network (KPN) [Kah74]. In KPN, processes communicate by sending tokens through channels which consist of First-In First-Out (FIFO) queues. We can consider KPN as a generalization of Dennis dataflow model [LP95] which models the concurrent tasks in multithreaded architectures.

In the real-world implementation, the communication channel in dataflow consists of FIFO buffers and endpoints which connect software and hardware tasks. FIFOs are used to adapt to different protocols and bandwidth between tasks or to meet some constraints, e.g., asynchronous data read/write. They can be located on the private memory of CPUs, on FPGAs, or on a shared memory. There is a difference when storing the communication data in FIFOs between CPUs and FPGAs. On CPUs, FIFO data are stored in the memory (and cache). The size of FIFOs can be increased and reduced with ease, and they can be created and removed by managing pointers. On FPGAs, FIFOs consume FPGA's logic and internal memory resource. Their width and depth are fixed in the FPGA bitstream configuration.

2.2 HARDWARE CONTEXT SWITCH ON FPGAS

With the increase of FPGA speed and capacity, there are growing interests in exploiting them for general purpose computing [DeH96]. In general purpose use, computing resources are required to perform a wide range of tasks. For this reason, a concept of multi-users or multi-tasks must be supported in FPGA design. By default, an FPGA can provide a certain level of flexibility due to their reconfigurable characteristic. Unlike in ASICs, a task or an application running on an FPGA can be replaced through a bitstream reconfiguration. However, the progress of execution on the FPGA is wiped out once it is reconfigured with another bitstream, which makes it impossible to resume an interrupted task execution. To approach the flexibility of a CPU in executing a number of different tasks simultaneously, a context switch support is necessary on FPGAs. Note that a context switch on FPGAs can be performed either temporally or spatially. In this work, we are particularly interested in the temporal sharing of FPGA devices which requires discharging their resource from a currently running hardware task. Spatial sharing of FPGAs by storing multiple configurations inside the FPGA memory and multiplexing them according to the process executed, as presented in [Tri+97; SV98; LCH00; TV02], is beyond the discussion in this report.

2.2.1 *General Concept*

In Operating System (OS), a context switch is a well-known process of switching from a task or process to another task in order to temporarily share CPU resources. In context switch, the state of a task (task context) is saved before replacing it to

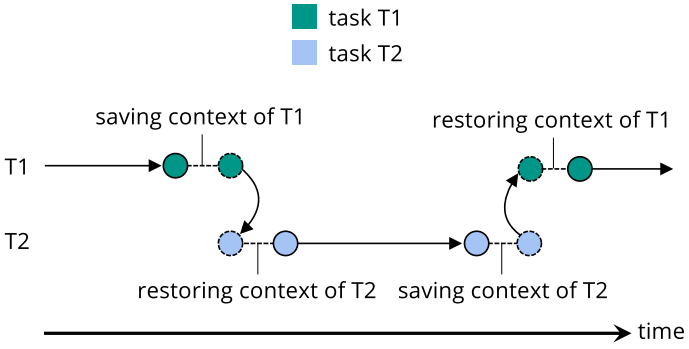
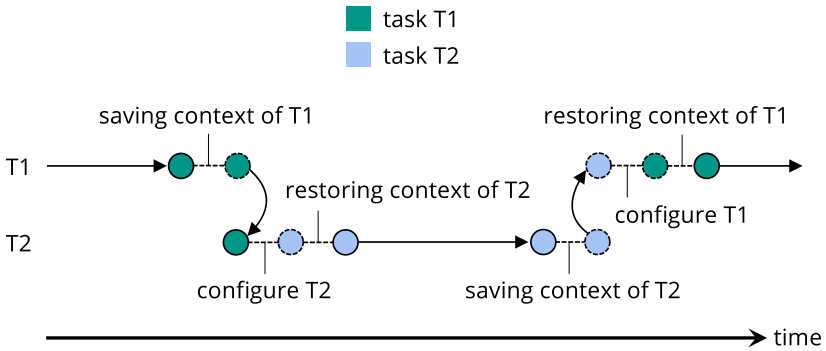


Figure 6: An illustration of a software-based context switch

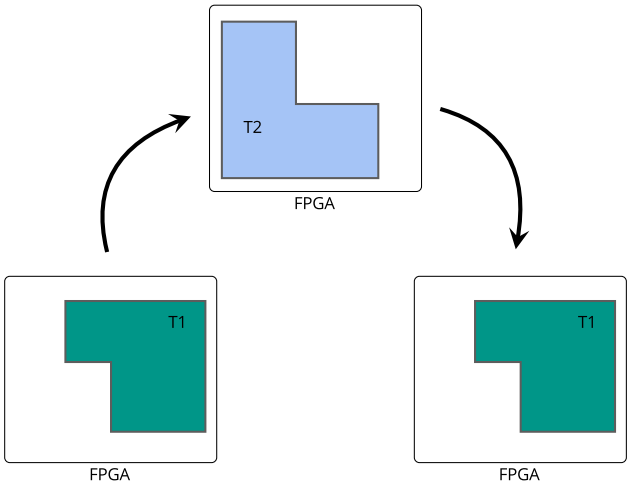
another task. Later, the task can be resumed by restoring the saved context either to the same CPU or another one which is compatible. A task context in a CPU is the contents of a CPU's registers and program counter at any time. Figure 6 describes a context switch between two different tasks: T1 and T2. Before the CPU execution is given to T2, the context of T1 must be saved. Afterward, the execution of T2 can start. To context switch back from T2 to T1, the context of T2 must be saved, followed by the restoration of the context of T1. Although the process of context switching a task on CPUs can be costly [LDS07] as it requires a certain amount of time for administrating – saving and restoring – context, it remains an effective solution to support software multitasking.

It turns out that a similar feature requires more efforts to be implemented on FPGAs. First of all, a context switch operation must be done at hardware level (physical layer) since the FPGA tasks are implemented as hardware circuits. A hardware context switch requires saving and restoring the contents of registers and memories of the circuits on FPGAs. Furthermore, a bitstream reconfiguration may be necessary when the task or IP on the FPGA is replaced. An illustration of hardware context switch operation is presented in Figure 7. Two different tasks, T1 and T2, are executed on the same FPGA. After the context saving of T1 is completed, the FPGA must be configured with T2 and vice versa. Second of all, an additional mechanism to save and reload the context from hardware circuits is necessary. A native support of the hardware context switch is available only in very limited commercial FPGAs at the time of writing of this thesis report. On top of that, a context switch due to migration purposes requires the task to continue to an identical FPGA. Unlike in CPUs, each hardware task on FPGAs may occupy different area in the FPGA which makes it difficult to support the hardware context switch. For these reasons, providing a context switch support on FPGAs requires efforts in the design, tool, and infrastructure.

Over the years, a number of research works have been proposing mechanisms to access the context from a task running on FPGAs [Joz+12]. In general, there exist two main classifications for hardware context switch support on FPGAs. The first classi-



(a) Temporal overview



(b) Spatial overview

Figure 7: An illustration of a hardware-based context switch

fication extracts the context through FPGA configuration port whereas the second through an additional specific interface of the respective hardware task. Some works based on intermediate fabrics to facilitate fast reconfiguration and context switch operation on FPGAs are also introduced [LLB14]. A more thorough discussion of the existing works for hardware context switching is presented in Chapter 3.

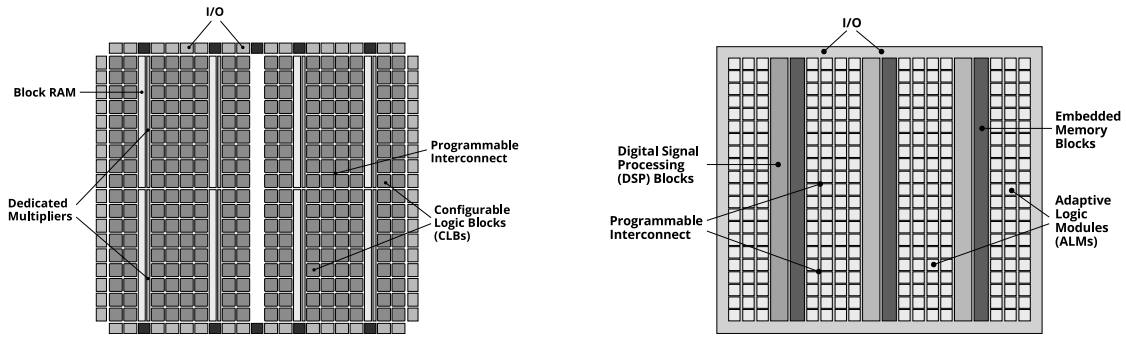
2.2.2 *Heterogeneity Requirements*

The FPGAs in today's market are thousands of times bigger and almost a hundred times faster than when the first FPGA was introduced. At the same time, they integrate more heterogeneous elements, e.g., DSPs, Block Random-Access Memory (BRAM), or even larger dedicated Ultra Random-Access Memory (UltraRAM) [Ahm+16]. Heterogeneous FPGAs offer a higher performance in custom computing since they integrate more and more specialized units for different purposes, such as high-performance DSPs and optimized Random-Access Memories (RAMs). Furthermore, FPGA vendors tend to provide a wide range of products (or families) with different cost, size, and performance in the market. On the one hand, FPGAs have become more capable of handling complex circuits, which is essential for their function as hardware accelerators in reconfigurable computing systems. On the other hand, the heterogeneity of FPGA architectures increases more than ever, which makes circuit design and configuration specific to the tools from FPGA vendor or manufacturer.

Besides that, there exists a heterogeneity which is found between FPGAs due to their difference in FPGA architecture, family, technology, etc. Figure 8 illustrates two examples of FPGA architectures from different manufacturers. A similar trait is also found among different FPGA families from the same manufacturer. A system that uses different FPGAs can be considered as a heterogeneous reconfigurable system. Given these points, the use of FPGAs as hardware accelerators must consider their heterogeneous architectures especially when we want to provide a multitasking support on them. For the rest of this report, the term heterogeneous architectures will be used to describe the characteristic in multi-FPGA systems.

2.2.3 *Scheduling Constraints*

Similar to the software context switch, a hardware context switch operation is performed due to a trigger or request from a scheduler. In reconfigurable architectures, the scheduler can be implemented in a CPU or in a dedicated FPGA specialized for hardware management. A scheduler follows a policy, static or dynamic, to take a decision upon a hardware context switch. A scheduling policy on FPGAs can fall into two categories according to the different approaches in reconfigurable systems [Gua+08]: *non-preemptive* and *preemptive*.



(a) Xilinx Virtex II architecture (b) Altera Cyclone V architecture

Figure 8: Heterogeneity in FPGA architectures

A non-preemptive scheduling, also known as co-operative scheduling, is a style of scheduling in which a task voluntarily yields the control of computing resources to another application without being initiated by another process. On FPGAs, a non-preemptive scheduling waits until the task runs to completion before a context switch can occur. Then, the current configuration of the FPGA can be switched to a new one [WP02]. A non-preemptive context switch can be performed regularly since the tasks occupying a common FPGA *cooperate* for the entire scheduling to work.

A preemptive scheduling is a style of scheduling in which a preemption is given to a running task without requiring its cooperation in order to exploit the occupied resources for another task. In this case, the execution flow of the preempted task is expected to be resumed at a later time. In preemptive scheduling, a context switch can be performed at arbitrary points in time according to the scheduler's request. Preemption requests can be given due to the arrival of urgent or higher priority tasks, insufficient execution window, or technical fault in the system. On the one hand, a preemptive context switch offers a high flexibility in managing multiple hardware tasks inside an FPGA. On the other hand, it requires higher efforts to handle the hardware task preemption. The preempted task may be in a computation phase which has ongoing communication flows. A preemptive context switch, therefore, should consider interrupting the flow of execution as well as communication in the FPGA.

2.3 PROBLEM STATEMENT

With the growing interest of using reconfigurable accelerators, FPGAs are put together with CPUs to offer higher processing performance and energy efficiency. The communication between FPGAs and CPUs in a reconfigurable architecture follows the dataflow model. Data transfers between tasks are done through FIFO channels. To answer the demand for multitasking on FPGAs, a hardware context switch sup-

port must be provided. Based on the scheduling policy, a hardware context switch can follow either a non-preemptive or preemptive scheduling. In non-preemptive scheduling, hardware tasks on FPGAs yield the control of the resource for a context switch due to the end of their execution flow. Although there still exist some issues which need to be solved [WP02], communication between tasks is not one of them as a non-preemptive context switch ensures the communication flow is finished before voluntarily yielding the control to the scheduler. Despite this, a non-preemptive solution lacks the reactivity to immediately release FPGA resources for high priority tasks and the capability to run multiple tasks simultaneously. For this reason, a preemptive context switch support on FPGAs is necessary to increase the flexibility in hardware task management.

In this thesis report, we limit the discussion only for the hardware context switch due to preemptive scheduling. For this reason, a hardware context switch on FPGAs may occur at arbitrary points in time, including when there are still ongoing communication flows. The preemption requests will interrupt the execution flow of the tasks on FPGAs as well as their communication flows. When the communication aspects are considered, there still exist some challenges that need to be overcome to support a hardware context switch on FPGAs. The following presents the questions which will be answered throughout this thesis report.

HOW DO WE MANAGE THE CONTEXT FROM A HARDWARE TASK? This is the first question that is raised when looking to enable the hardware context switch on FPGAs. It consists of providing the method to access the context of a task on an FPGA in order to save and restore the snapshot of the task at preemption. According to the method chosen, the communication of the task during a hardware context switch will be managed differently. Managing the task context also includes how it will be relocated and where it will be stored. The question of where the task will continue at a later time usually follows the context management since a context switch may also include migration.

HOW DO WE PRESERVE THE COMMUNICATION DATA INTEGRITY? One of the issues that may arise when a hardware task with ongoing communication flows is context switched is the inconsistency of communication data. As mentioned earlier, the scheduler can take control of the resources without any consent from the task which is currently running on the FPGA. Meanwhile, whether a task is still communicating with other tasks in the system depends on the current state of the task. In other words, a hardware context switch may compromise the communication data integrity or consistency which can lead to irreversible error in the execution [Xie+15].

To give an illustration of this risk, Figure 9a depicts some tasks in CPUs (SW) and in FPGAs (HW) running in parallel and communicating with each other through channels. Each task has one or multiple Input/Output (I/O) ports and each port

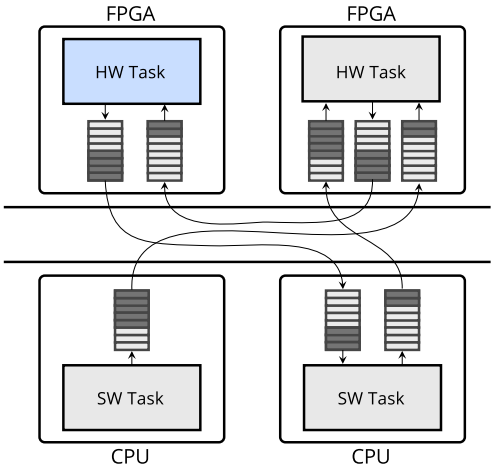
is connected to a FIFO that intermediately stores the communication tokens while they are being transferred. The SW tasks are placed in any available CPU in the system whereas the HW tasks are in any FPGA. The representation used in the figure is generic. It is independent of the communication topologies in the architecture. Popular communication topology, e.g., Bus, Switch, Network-on-Chip (NoC), etc. can be used in the architecture. In case a shared memory is used (distributed shared memory system), the FIFOs are used in communication channels in the same fashion.

I/O communication data (tokens) are sent to local FIFOs when a task on an FPGA is in communication or when the data are ready to be consumed by the destination task to increase the efficiency. When a hardware context switch occurs on an FPGA, the execution as well as the communication flows of the task will be interrupted. Figure 9b illustrates the situation where a hardware task is removed from an FPGA in the system while it still has ongoing communication flows. The communication tokens stored in FIFOs which are not consumed or forwarded yet will be left untreated since the associated hardware task is not there anymore. The explained situation will generate inconsistency issue during a hardware context switch due to several reasons. We may lose the communication data inside the FIFOs which are connected to the preempted task in the FPGA. Loss of data is unacceptable in hardware context switch as it will cause error, incomplete execution, etc. Maintaining the I/O data in the correct order is also important to maintain the consistency in communication data.

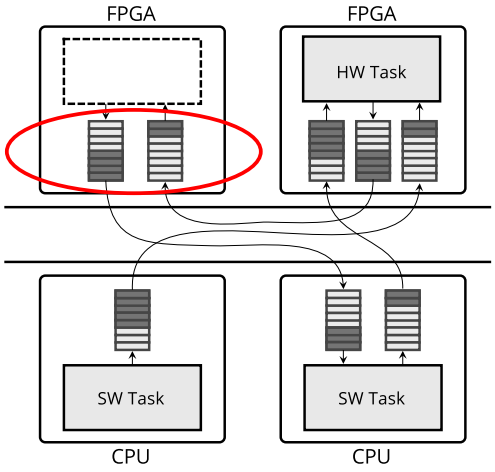
Another cause of inconsistency is when a new task arrives in the FPGA after a hardware context switch, as described in Figure 9c. Similar to the replaced task, this task requires FIFO buffers to store its I/O communication tokens. As the untreated tokens from the previously context switched task are still left in FIFOs, the current task will consume false I/O data. It should be somehow guaranteed that a task will not process the tokens which is not associated to it due to a hardware context switch.

Ensuring the consistency of communication data until the resumption of the preempted task is as important as extracting and restoring its context on the FPGA. Otherwise, providing a hardware context switch in reconfigurable architectures will not be easy, unless all the HW tasks run without any communication or isolated. In reality, isolating HW tasks hinders their role as accelerators and, therefore, is not preferred. As a consequence, a preservation of communication data integrity is necessary.

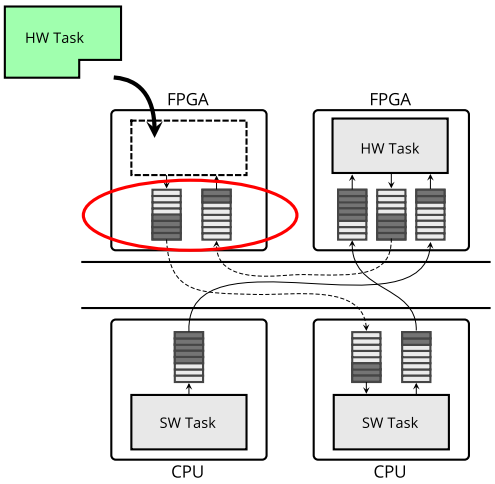
HOW DO WE SATISFY THE PERFORMANCE CONSTRAINT IN A HARDWARE CONTEXT SWITCH? In preemptive scheduling, a hardware context switch operation on an FPGA has to meet a certain deadline since the resources are required by a higher priority task. Failing to do so will result in a low performance of multitasking support. In some cases, it is more advantageous to wait a running



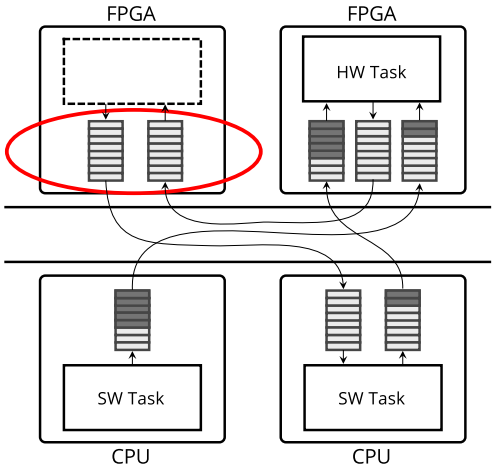
(a) Tasks in a reconfigurable architecture running in parallel and communicating with each other



(b) A hardware task which is context switched while it still has ongoing communication flows



(c) A new task occupies the FPGA resources and uses FIFOs for communication



(d) Hardware context switching a task on FPGA only when there is no communication significantly reduces the performance

Figure 9: Communication issues due to different situations in the hardware context switch

task to finish rather than going through a hardware context switch. And sometimes, the progress of an execution can be thrown away as it costs less than saving it. A study done by Rupnow, Fu, and Compton [RFC09] evaluates these possibilities to obtain the best performance of multitasking ability in reconfigurable architectures. Nonetheless, when a task preemption is inevitable, a faster hardware context switch is preferred.

The performance in a hardware context switch can be improved in several ways. Although further discussion is beyond this report, some popular techniques to reduce the size of bitstream configuration and context of a task can be used, such as using Dynamic Partial Reconfiguration (DPR) on FPGAs [LF09]. The objective is to minimize the size of data involved when context switching a task on an FPGA.

Unfortunately, the existence of communication flows when a hardware context switch occurs will reduce the performance. This issue, in fact, appears when we try to preserve the consistency of communication data without a proper method. Without any specific mechanism, the only solution is to avoid or to delay the hardware context switch if the task still has ongoing communication flows or I/O communication data in its FIFOs. Such approach inflicts a restrictive constraint to the preemptive context switch. As a result, the context switch will be delayed in an unpredictable manner in order to complete the communication flows of the task, as illustrated in Figure 9d. Satisfying the performance requirement in a hardware context switch is one of the issues which needs to be resolved to support multitasking on reconfigurable architectures.

HOW DO WE ENSURE THE CONTINUITY OF COMMUNICATION BETWEEN TASKS? To resume all of the previous questions, a hardware context switch needs to guarantee that a task execution can resume after it is reloaded to one of the FPGA in the system. As we have known, when a hardware task is context switched, its communication flows with the other tasks in the system is interrupted. Obviously, context switching a task which is a part of a communication network causes a certain impact in the entire communication flows. When the previously context switched task is recharged back to the system, the communication needs to be restored.

To ensure the continuity of communication between tasks, we have to consider the links between FIFOs. A connection and disconnection of the links associated with a preempted task must be managed in a hardware context switch. The links must be updated when a hardware task is replaced with another task and reloaded back, either to the same or different resource (migration).

2.4 CONCLUSION

Due to the performance and parallelism offered, FPGAs have been more and more used in accelerated computing. Although FPGAs offer a lower flexibility in execution compared to CPUs, their reconfigurability characteristic makes them exploitable for

general purpose use. An even higher performance can be obtained by supporting multitasking, for instance, using a hardware context switch on FPGAs. A hardware context switch operation consists of interrupting a running task on an FPGA, extracting the current task context, and inserting the context of the new task to the FPGA.

In this chapter, we have shown that, to provide a hardware context switch support on FPGAs, the ability to extract the context of a running task and later restore it on an FPGA to continue its execution is not sufficient. As a hardware task requires I/O communication and synchronization, a mechanism to manage the communication in a hardware preemptive context switch is undoubtedly necessary. In regards to these communication aspects, we have several issues which need to be treated during a hardware context switch operation in reconfigurable architectures. In this work, we present a solution that can answer the questions described here. Before we describe our solution in Chapter 4, the next chapter will present the existing works in a hardware preemptive context switch.

3

STATE OF THE ART

In this chapter, we present a review of the existing research works whose objective is to provide a hardware context switch in reconfigurable architectures. Before we discuss the methodology for managing the communication in a hardware context switch, it is important to know the solutions which have been proposed in the community.

The mechanism to manage the communication in reconfigurable architectures during a hardware context switch depends on the technique to perform the context extraction and restoration. The first part of this chapter describes the works which have been done to enable hardware task context extraction on FPGAs. The next part of this chapter presents the existing management of reconfigurable systems in regards to the hardware context switch support. These works include the communication and synchronization between SW and HW tasks in reconfigurable architectures. Finally, this chapter will be concluded with a vision of what has been done and what must be done next in order to manage the communication during a hardware context switch operation.

Contents

3.1 HARDWARE TASK CONTEXT EXTRACTION

As briefly presented in Section 2.2, some research works have proposed mechanisms to extract the context of a hardware task from an FPGA. A task context is defined as a minimum set of live variables and states which are required to resume the execution flow of a task after a context switch. These variables and states are stored in the FPGA resources (registers and memory) of the preempted task. They are extracted when a running task is interrupted due to a preemption request and restored back when the task resumes.

There exists currently few supports to extract the context of hardware tasks running in certain Commercial Off-The-Shelf (COTS) FPGAs. Additional mechanisms, therefore, are necessary to provide such a feature widely. This section presents the developments of the design methods and tools to enable the task context extraction (and restoration) on FPGAs. First, we describe the existing works which take advantage of the FPGA configuration port, known as the *configuration-based* technique. Then, we present other works that require a certain modification in the hardware task design, namely the *design-based* technique. Finally, we show another methodology to reconfigure and context switch tasks on FPGAs with overlay architectures.

3.1.1 *Configuration-based Technique*

The technique which reads back the task context including the configuration bitstream from the FPGA configuration port is classified as Configuration-based, also known as *Platform-specific*, *Configuration Port Access* or *Readback* technique. This technique is the first and considered the most popular solution that allows context switching hardware tasks on FPGAs as it is supported by a certain FPGA vendor.

Back in 2000, the authors of [Lev+00] and [SLM00] presented the ideas of FPGA multitasking and the proof-of-concept using readback technique on Xilinx XCV300, XCV400 and XCV1000. In their works, they specified the requirements of readback techniques as follows. Readback technique for FPGA multitasking requires the FPGA board to have configuration and readback ability. Therefore, this technique is only possible on specific FPGA families from certain vendors. A complete control of the clock on the board is necessary to halt the task at a known and restorable state. The bitstream extracted from the FPGA must contain the status of all internal registers and memory.

Although the use of readback technique in hardware context switching enables multitasking operation on FPGAs, it also presents some disadvantages. The useful information which is necessary to resume the execution of a task after the context switch is actually less than 10% of the total readback data [Lev+00; SLM00]. The rest of unnecessary data must be filtered in order to obtain the task state from a readback bitstream. With the technique proposed in [KP05], the data footprint in readback technique can be reduced by extracting only the part that includes state

information and belong to the suspended task. Another disadvantage in the readback technique is the reconstruction of the task when it resumes the execution, which is a merging process of the extracted task state and initial task configuration. To reduce the effort in task reconstruction, [LWH02] proposes a modification of the bitstream with custom tools such as JBits [JG00] to restore FPGA state. In [Lev+00; SLM00], a concern for invalid data read/write is discussed if a task is context switched while communicating with external RAM as the clock associated to the preempted task is frozen during a context switch. To avoid such issue in readback, a context switch is not allowed when there is communication.

Over the years, readback technique has been evolving to be more effective for FPGA multitasking especially on Xilinx products. An improved support in the COTS FPGAs has been highlighted in [Joz+10], for instance, the readback and configuration capability through Internal Configuration Access Port (ICAP) on Xilinx Virtex-4 device [Xil17]. As readback access the port which already exists on the FPGA, it does not require extra design efforts and additional hardware consumption. Reconfiguration throughput in readback technique can be increased using a solution such as Fast Reconfiguration Manager [DML11].

Nevertheless, readback technique remains exclusive for Xilinx FPGAs at the time of writing of this thesis report. For a hardware context switch that involves task relocation, the FPGAs used in reconfigurable systems must be identical. In other words, a homogeneous environment is strictly required to provide the hardware context switch support. A bitstream parser for each FPGA is necessary for separating the context from the bitstream and, therefore, for reconstructing the task that will be restored to an FPGA. These challenges including the overhead in reconfiguration time and memory size are to be addressed to provide the multitasking feature on FPGAs.

3.1.2 *Design-based Technique*

Another technique which allows task context extraction on FPGAs is design-based or *Task-Specific Access Structure (TSAS)* technique. Initially proposed for debugging purposes on FPGAs [Whe+01], the design-based technique integrates additional structures, e.g., scan-chain structures, into the task design to enable access of the task states and live variables stored in FPGA registers and memory. Design-based technique extracts only the state-related data without any redundancy, leading to high data efficiency in the task context extraction. In the design-based technique, the context is extracted and restored back through an additional interface of the task which does not require freezing the global clock on the FPGA [JTW07]. The other part of the FPGA, therefore, can still work, for instance, to run another hardware task.

Despite efficiency offered in the context extraction, the design-based technique has a drawback in the performance and FPGA resource consumption. Wheeler et al. ex-

plained the resource consumption in [Whe+01] due to the instrumentation of design primitives, e.g., FFs and RAMs, and design hierarchy on FPGAs. Likewise, a design instrumentation which uses register scan-path structures is presented in [Koc+04]. Such design instrumentations significantly increase the average FPGA resource utilization; furthermore, they require high efforts from the users. In hardware tasks that follow Finite State Machine with Datapath (FSMD) model, Jovanovic, Tanougast, and Weber [JTW07] also described the requirements of adding structures both in the controller and datapath for preemptive context switch support. Note that a bitstream reconfiguration is also required if the context switch operation involves different applications on the FPGA.

Nevertheless, the design-based technique can offer the advantages in data efficiency since the bitstream is not included in the context extraction. This technique is also independent of the FPGA technology. As a result, either any knowledge of the bitstream or customized bitstream parser is not required in providing a hardware context switch support. Koch, Haubelt, and Teich [KHT07] proposed using checkpoint concept in design-based technique to obtain higher efficiency during task preemption. A checkpoint state is a state where a context extraction from a hardware task on an FPGA is allowed. The checkpoint selection can benefit from the size of live variables that may vary from one state to another. By allowing the context extraction only at checkpoint states, a smaller size of task context may be obtained with a trade-off in the latency of preemption. In fact, all the recent works using design-based technique integrate the checkpoint concept on different levels of implementation [KHT07; Vu+16; BMR16]. In [KHT07], scan-chain structures are added to the netlist of hardware tasks. Vu et al. [Vu+16] proposed to modify the HDL code of the task to obtain a higher level of abstraction. Bourge, Muller, and Rousseau [BMR16] presented a mechanism which takes advantage of High-Level Synthesis (HLS) flow to reduce users' effort in adding scan-chain structures to hardware tasks while at the same time obtaining the efficiency offered by the checkpoint concept.

To summarize, the design-based technique produces an overhead in hardware resource consumption and task performance which remain a challenge. Furthermore, it may require an additional bitstream reconfiguration in addition to a context restoration. The advantages of using a design-based technique are that it offers higher data efficiency in context extraction and it is FPGA technology-independent. The integration of checkpoint concept reduces the overhead in performance and resource with a trade-off in preemption latency.

3.1.3 *Overlay Technique*

As an alternative to the explained techniques of hardware context switch, the overlay technique is proposed on FPGA architectures to answer the requirements for high performance in the context switch and task reconfiguration. The concept is to synthesize reconfigurable architectures on top of the commercial FPGA archi-

tectures. These synthesized architectures are homogeneous and independent to the FPGA technology used. Hence, the homogeneous architectures overlay the heterogeneous architectures of FPGAs to generate virtual FPGAs (vFPGAs) [Naj+17]. The overlay-based vFPGAs have some advantages, for instance the flexibility and compatibility between different FPGA architectures which are necessary in the hardware context switch support.

The overlay architectures can be developed on both fine-grained and coarse-grained architecture. In [Naj+17], the reconfigurable architectures are composed of fine-grain reconfigurable elements. Although the virtualization in a fine-grain architecture may produce a significant overhead, it makes sense for some applications that require portability and efficiency in the resource utilization. Nevertheless, coarse-grained overlay architectures or Virtual Coarse-Grain Reconfigurable Arrays (CGRAs) are more common. The concept is developed from classic Coarse-Grain Reconfigurable Array (CGRA) which consists of a mesh of Processing Elements (PEs) and interconnect network. The PEs in CGRAs are configurable at words-level granularity to map the parts of computation and to be connected to each other via the interconnect. For the overview of CGRA architectures, the reader can refer to RaPiD [ECF96], PipeRench [Gol+99] and ADRES [Mei+03]. A bitstream configuration is only required once to map a VCGRA architecture on an FPGA. After that, a task 'configuration' only involves the settings of the PEs and interconnect. As a result, the hardware context switch in VCGRAs can be done much faster than the fine-grained solutions.

With the fast reconfiguration and hardware context switch, VCGRAs are suitable for hardware multi-threading support on FPGAs. Despite this, VCGRA architectures have significant drawbacks in task performance and resource utilization. A 40× [BL12] and 100× [Lys+05] more hardware consumption than a regular FPGA implementation are obtained due to the coarse granularity of the architecture. A decrease in performance due to 10× longer critical path is also highlighted in [Lys+05]. For hardware acceleration purposes, the processing performance and resource utilization are, in fact, very important. Consequently, until the performance and resource challenges in VCGRAs can be solved, the applications of this technique will remain limited.

3.2 RECONFIGURABLE SYSTEM MANAGEMENT

The ability to preempt a running task on an FPGA and extract its context is essential in providing the hardware context switch support in reconfigurable systems. Managing the context (saving, storing, and restoring) as well as the communication in the process is equally important. In reconfigurable architectures, a CPU which runs a customized OS is generally required to manage the communication between tasks, both in CPUs and FPGAs.

In this section, we review some works which present an inter-task communication management in reconfigurable architectures [VPI05; LP08; Nar+11; Joz+13; Vu+16]. These works consider that one of the CPUs in the reconfigurable system manages the communication and synchronization between the software tasks on the CPU and the hardware tasks on FPGAs using OS modules. Although not all of them support hardware multitasking on FPGAs, their implementation on reconfigurable architectures may provide some insights to our work.

3.2.1 *Linux-based OS*

Vuletić, Pozzi, and Ienne [VPI05] proposed a virtualization layer which allows a hardware accelerator running on behalf of a user application to access the user space virtual memory. The virtualization in this work is provided by a Linux OS module which also assists the communication between hardware and software. With the utilization of virtual memory, FPGAs and CPUs in the system share the same address mechanism with the main memory. As a result, local memory inside the FPGA which stores the I/O communication data is not required. This solution offers some advantages, such as in duplicating or prefetching communication data with ease, and variable allocation of memory page size for each task since the communication in the system is performed through virtual memory.

The work presented in [VPI05] does not support the preemptive context switch on FPGAs. Nevertheless, a similar solution can be implemented in a reconfigurable system with a hardware context switch support. Their technique can be generally applied to a reconfigurable architecture with a CPU which runs Linux OS. When a HW task is context switched, the I/O data can be safely stored in the main memory. Storing the addresses and pointers of virtual memory is necessary to resume the communication at a later time. Although they present an interesting method to manage the communication, the solution is strictly limited to shared memory systems. The memory access latency must also be considered in the communication between hardware tasks. Since the hardware task is directly connected to the main memory in the system, there is a risk of inconsistency when context switching the task while it is communicating [SLM00].

3.2.2 *ReconOS*

Built on top of an existing (real-time) OS, ReconOS enable transparent thread-to-thread communication regardless of the HW/SW partitioning [LP08]. In ReconOS, HW tasks are modeled as independent executing threads whose communications are controlled by the OS modules. The HW-SW communication uses a method similar to message-passing whereas the HW-HW communication employs FIFO buffers. A specific OS synchronization state machine is added to the FPGA to facilitate the control of HW tasks. Thanks to the control from OS modules, flexible synchronization

and communication mechanisms for multithreaded HW/SW systems are provided on both the level of the programming model as well as the low-level implementation of the HW/SW interface. Moreover, these communications and synchronizations are transparent from the tasks.

Despite this, ReconOS lacks the scalability in HW-HW communication. It considers multithreading between HW and SW with a single HW in the system. Furthermore, preemptive scheduling is not available for HW task on the FPGA part of the system. A support for preemptive context switching in ReconOS was later proposed in [HTK15]. This work uses ReconOS to control the communication and task context extraction on the FPGA using the readback technique. Alas, Happe, Traber, and Keller [HTK15] did not take the consistency preservation into consideration when a context switch occurs. Although further development is still required, a mechanism to preserve the communication and synchronization in a preemptive context switch can be integrated to a stable OS which is optimized for a reconfigurable system like ReconOS.

3.2.3 *FOSFOR*

A more scalable and communication-intensive solution than ReconOS is proposed using FOSFOR platform [Nar+11]. In FOSFOR, a notion of HW OS that manages hardware tasks in FPGAs is introduced. These hardware tasks are configured in partial regions of a DPR system. The communication between tasks is managed by a middleware which monitors the channels requests and establishes the connection. This middleware can set the communication to blocking and non-blocking according to the existence of tasks. A NoC topology [Dev+10] is used to improve the flexibility and reduce the communication overhead.

A communication mechanism which adapts to blocking and non-blocking situation can be used in multitasking systems. FOSFOR provides this possibility using HW OS on the FPGA. In [Nar+11], this mechanism is combined with DPR for non-preemptive scheduling in hardware task management. Naturally, a further development is still necessary to provide a preemptive context switch support in FOSFOR platform. In their work, the impact of the solution to the performance in communication is also not precisely quantified.

3.2.4 *Rainbow*

Another work that manages inter-task communication is Rainbow which is based on Operating System for Reconfigurable Systems (OS4RS) [Joz+13]. Rainbow is developed to support preemptable hardware tasks on reconfigurable systems. It provides the Application Programming Interface (API) which allows the communication and synchronization from SW. On the HW side, it supports DPR and uses the readback technique to provide the preemptive context switch. In the commu-

nication aspect, hardware and software tasks in Rainbow are connected to a bus topology with the communication channels located between reconfigurable regions in the FPGA. The communication data is transferred between tasks using a point-to-point message-based communication. Such generic communication mechanism is possible by encapsulating the HW task inside each partial region with a wrapper.

As explained in Section 3.1.1, the use of readback technique in the preemptive context switch limits the implementation to a specific FPGA. However, the communication data inside the reconfigurable region can be managed as long as they are not included in the context extraction. In this case, communication FIFOs are considered as the part of the HW task in our hypothesis. In other words, users must have the knowledge of the HW task execution on the FPGA and third-party use is generally avoided. As the work in [Joz+13] is based on the readback technique, relocating tasks is possible only between identical partial regions. The HW task relocation has not been taken into consideration in their work. It also lacks the possibility of using multiple FPGAs in the system and communication channels using external memory, as presented in [SLM00].

3.2.5 *CPRtree*

Vu et al. [Vu+16] present an infrastructure which manages a context switch on FPGAs. Using the design-based technique, the context of a task can be accessed at checkpoints through a scan-chain interface. The global checkpoint/restart process on the FPGA is controlled using API functions on the CPU. Similar to Rainbow, HW tasks in CPRtree are connected on a wrapper inside the FPGA. However, this wrapper manages not only the communication but also the context extraction/restoration as their work uses the design-based technique.

Although the work in [Vu+16] provides a preemptive context switch support which can work in a heterogeneous environment, their system only extracts and restores the I/O communication data inside the hardware task. The communication FIFOs outside the task is not taken into consideration. Therefore, the only way to maintain the communication consistency when a preemption request is given to a task with ongoing communication flows is to stall the request until the task consumes all the I/O communication data. They propose to throttle the communication data and prevent new data from arriving at the buffers after a preemption request is received. Although they claim that this solution will speed up the latency when preempting a task, the impact of the communication solution is not evaluated in their work.

3.3 CONCLUSION

Providing a hardware context switch support on FPGAs has been a research topic for more than a decade. In order to context switch a hardware task on an FPGA, communication management between tasks is required. Before extracting the con-

text of a running hardware task, its execution and communication flows must be safely interrupted to preserve the communication consistency. Data integrity should be maintained to give a hardware task on an FPGA the possibility to resume its execution correctly after a context switch. Over the years, context extraction solutions have been proposed using the configuration-based, design-based, and overlay techniques. However, the communication management has not been receiving enough attention as it deserves.

Although none of the previous works particularly discuss the communication management in preemptive context switching, the readback technique may offer a certain advantage to preserve the consistency. In readback, the clock associated with a task on an FPGA is halted so that the context can be extracted as-is from the configuration port of the FPGA. The I/O communication data in the FIFOs can be extracted together with the task context as long as they are located on the same FPGA [Joz+13; HTK15]. However, readback techniques are technology-dependent and not applicable for heterogeneous FPGA environments. The context (and the I/O data) can only be restored to an identical region of the same FPGA. Besides the fact that the readback technique is tailored to specific FPGA families, the work in [Joz+13] also mentions a problem in BRAM extraction which strongly depends on the manufacturer.

A technology-independent approach in a hardware context switch can be offered using the design-based technique [KHT07; BMR16; Vu+16]. In contrast to readback, the context extraction and insertion using design-based technique must be done without stopping the associated clock of the task. On the one hand, this solution provides a higher efficiency since the rest of the system is minimally affected. On the other hand, it requires a much higher effort to maintain data integrity in the communication channels of the preempted task. Additional infrastructure is necessary to access the task context from a specific interface on the task. The FIFOs which contain the intermediately-stored data must also be managed.

In [KHT07] and [BMR16], the focus was reducing the overhead in the design-based technique by using a checkpoint architecture. In [BMR16], the design effort is also significantly reduced by using HLS flow to generate hardware tasks which have context switch capability. Despite this, the communication management while context switching hardware tasks on FPGAs is not discussed. They limit the strategy for isolated tasks and do not consider the environment around the tasks.

The first work which uses design-based technique and considers the challenges in communication is presented in [Vu+16]. Although they claim to offer a framework that supports a hardware context switch, the constraints in communication may reduce the multitasking performance to avoid inconsistency in the communication data. To summarize, the mechanism to preserve the consistency in communication data while maintaining the performance of multitasking is still required. The next chapter will describe our solution to solve the issues in regards to the communication during a preemptive context switch.

4

COMMUNICATION MANAGEMENT IN HARDWARE CONTEXT SWITCH

LOOKING AT THE EXISTING WORKS presented in the previous chapter, we conclude that a comprehensive communication solution in a hardware context switch has yet to be found. The previous works still lack a mechanism to preserve the consistency in communication data when a hardware task is context switched at arbitrary points in time (due to preemptive scheduling). Such a condition results in a restrictive constraint in a hardware context switch; the preempted task must not have any ongoing communication flow and all communication data in FIFOs has been consumed. Moreover, the existing works are limited to homogeneous FPGA architectures.

Within this chapter, the communication mechanism in the hardware context switch is described. The first part of this chapter presents the general hypothesis in this work. Next, we describe the state-of-the-art method in hardware context switch which motivates us in developing our communication solution. Afterward, we present the programming model related to the existing hardware context switch support. We introduce a dedicated protocol to manage the overall hardware context switch, including the communication aspect, in the following section. Preceded by showing the compatible systems for our solution, the implementation of our method in the physical and communication layer is presented next with a discussion of the limitation. Finally, the conclusion is presented at the end of this chapter.

Contents

4.1 GENERAL HYPOTHESIS

In order to perceive our solution in communication during a hardware context switch, we need to review the architectures that it is intended for in the first place. This section will help us to focus on the advantages of the given solution as well as its limitation.

As mentioned earlier, we aim to support multitasking in reconfigurable architectures by managing context switch operation between hardware tasks on FPGAs while considering the communication aspect in the process. We define a task as a part of an application that runs in a reconfigurable architecture. As a reconfigurable architecture consists of CPUs and FPGAs, we recognize two natures of tasks: software tasks and hardware tasks. We consider software tasks as programs which are fetched from the memory and run on CPUs. Hardware tasks run in the form of IPs or digital circuits on FPGAs. In many applications, the IPs on FPGAs are an operator which can be used repetitively in the application.

In this work, FPGA reconfiguration may not be necessarily performed when different hardware tasks use the same IP. We consider two tasks using the same IP but different datasets as two different hardware tasks. Figure 10 describes an illustration of two hardware tasks with the same IP. Re-using a third-party or legacy IP is common in reconfigurable architectures as most FPGAs are dedicated to a specific function. However, the dataset processed by the IP can be different each time. For this reason, a hardware context switch may involve the saving and restoring of the context without changing the reconfiguration of the task on the FPGA.

The architecture targeted in this work contains CPUs and FPGAs which can communicate with each other through interconnect, as illustrated in Figure 3. This architecture has a memory as well to store shared data, such as the FPGA bitstream and task context. Figure 11 depicts an example of tasks running in such an architecture. T1, T2, and T3 are software tasks whereas T4, T5, and T6 are hardware tasks. All tasks are connected to the main interconnect that allows them to communicate to each other. At every communication port of the tasks, FIFO buffers are used to intermediately store the I/O communication data due to the bandwidth difference between tasks and the main interconnect. Several tasks can be located in the same CPU or FPGA. However, each of them should have its own connection to the main interconnect. An instance is created in a CPU or FPGA to manage the

task allocation in the system (task configuration for FPGA), which is presented as Task Manager in Figure 11.

Let us assume that a context switch operation can be performed to any hardware task in the system. The decision to perform a context switch is taken by a scheduler (placed somewhere in the system, for instance in Task Manager) and forwarded to the tasks as preemption requests. For this reason, a mechanism to preempt a hardware task and take a snapshot of its current state is necessary. In Chapter 3, state-of-the-art works have shown that accessing the context of hardware tasks is possible with certain requirements, e.g., technology, hardware utilizations, etc. More importantly, the architecture must be able to manage a task context in the hardware context switch which includes the extraction of certain memory elements and the capability to reload it in to the system. In the architecture, we depend on a Task Manager which contains the information about the tasks related to the communication. It should also safely disconnect and reconnect the communication links between the tasks while preserving the consistency in the transferred communication data. Adjusting the communication link between the tasks which are affected by the task context switch is also necessary. This can also be done by the Task Manager as long as the technology used in the communication allows it, which is the case in most of the popular bus and NoC communication standards.

4.2 HARDWARE CONTEXT EXTRACTION FOR HETEROGENEOUS MULTI-FPGA SYSTEMS

Section 2.2.2 briefly mentions heterogeneous reconfigurable systems with different COTS FPGA architectures. Heterogeneous FPGAs are gaining popularity, especially in future cloud services and data center environments [Put+15]. A recent paper presents a comparison between several types of heterogeneous platforms in data centers [Con+16]. The performance and energy consumption of FPGAs are

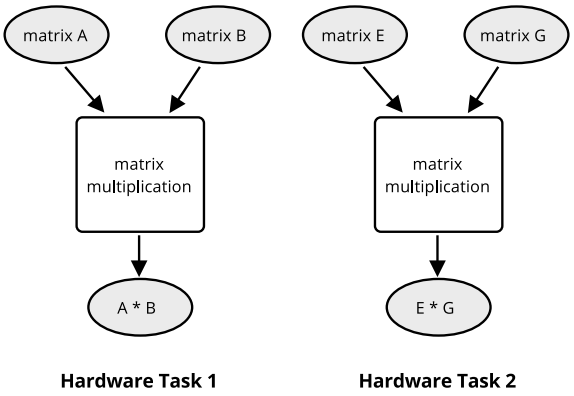


Figure 10: Re-using IP for different datasets is considered different hardware tasks in this work

evaluated using different implementations and programming models among other platforms, e.g., server-class CPUs, embedded CPUs, and GPUs. The comparison results show the benefit of combining various platforms with different traits to obtain the best trade-off in performance and efficiency. This has also been supported by the rise of Reconfigurable SoC platforms, e.g., Zynq-7000 and Zynq Ultrascale+ from Xilinx, and Cyclone V SoC and Arria V SoC from Altera, which integrate CPUs and FPGA resources in the chip.

For cost and compatibility reasons, multiple FPGAs with different sizes and architectures can be used in the same system. They can come from different FPGA families, or even from FPGA vendors. Each FPGA is normally dedicated to a specific task which may optimize the execution. These multiple FPGAs form a heterogeneous reconfigurable system when put together. In heterogeneous multi-FPGA systems, moving a task from an FPGA to another for migration purposes is complicated. That is to say, adding features such as multi-tasking in order to exploit FPGAs for general-purpose use is a conundrum.

4.2.1 *Design-based Technique*

To provide the capabilities of handling a wide range of applications, heterogeneous reconfigurable architectures are required. Such a requirement influences the way we provide hardware context switch support. In general, a context switch where the task continues on the same FPGA afterward is possible as a certain FPGA vendor, i.e., Xilinx supports configuration readback in some of their products. However, the readback technique extracts the configuration bitstream of the FPGA together with the context. This means that the readback technique needs to extract the context, store it in a normalized form and insert it to another configuration for a different FPGA used in the system which is an extremely complicated solution and requires conserving a huge amount of data for each FPGA technology. Consequently, the

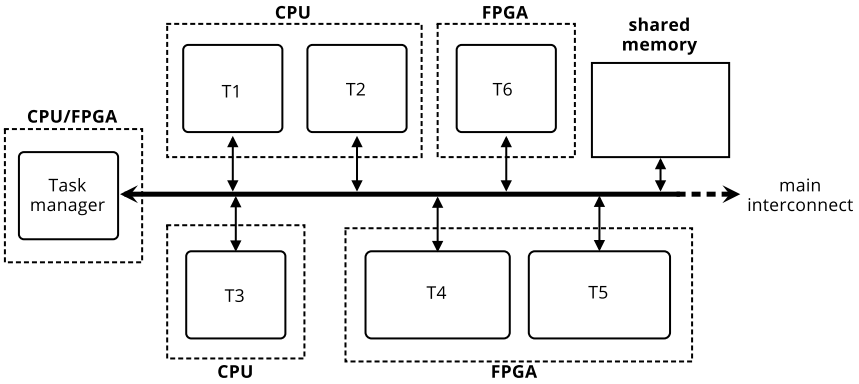


Figure 11: Example of tasks running on a reconfigurable architecture in this work

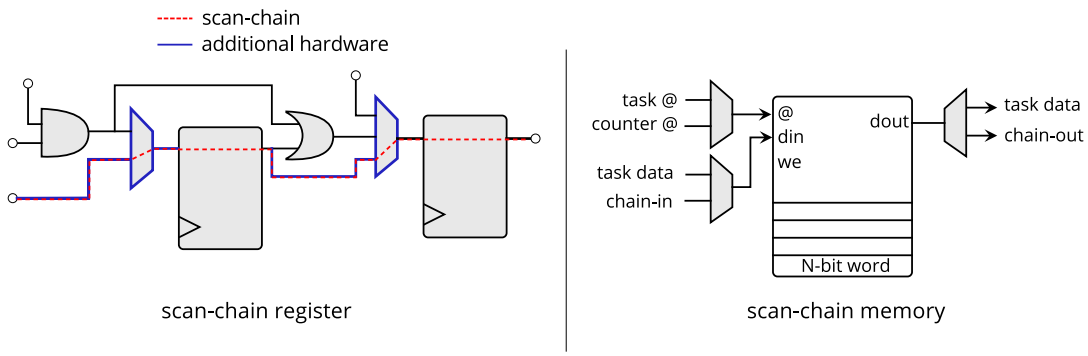


Figure 12: Illustration of a scan-chain insertion into datapath elements of a task (register and memory)

readback technique will hinder task relocation or migration when the FPGAs used in the system are heterogeneous.

As the alternative, another technique of context extraction must be used for the hardware context switch in heterogeneous architectures. The state-of-the-art works show that using either design-based or overlay techniques is possible to context switch hardware tasks between different FPGAs. However, both techniques require a certain modification in the regular task implementation. The design-based technique modifies the task design which is implemented on fine-grain FPGA architectures whereas the overlay technique runs hardware tasks on the virtual architecture. From the literature, we know that these techniques are technology-independent in contrast to the configuration-based technique.

We base our hardware context switch solution on the design-based technique. This technique instruments the hardware task with a structure which allows extracting its context from an FPGA and, eventually, restoring it to the same or different FPGA. In the FSMD model, a task consists of a state machine to control its execution flow and a datapath to execute the operation. The design-based technique inserts additional hardware both in the state machine and datapath of the hardware task. The hardware in the datapath is added to extract the context in a chain form (scan-chain) from memory elements of FPGAs, which are the register and memory contents associated to the task. Figure 12 shows an illustration of scan-chain structures connected to the register and memory elements. Since this technique adds hardware to the initial design, a higher FPGA resource consumption and a lower performance due to longer critical paths are expected, as already explained in Chapter 3.

In contrast to the readback technique, the IP in the design-based technique is not frozen while saving the current state of a task. Instead, the task execution enters into a specific flow in the state machine to perform the context extraction/restoration. Along with the scan-chain structures, some states in the Finite-State Machine (FSM) are selected as checkpoints which directs to additional states inserted in the state machine to handle the context switch. Figure 13 illustrates the modification

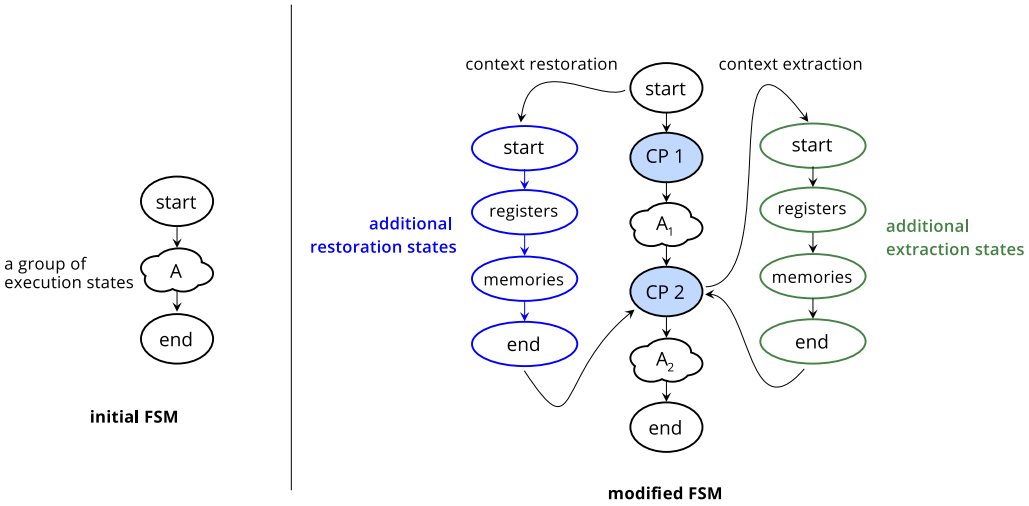


Figure 13: Modification in the finite state machine (FSM) of the task to handle a context switch operation

in the state machine to provide the hardware context switch support. In a normal condition, the execution flow of a hardware task follows the initial FSM with a group of execution states (A). The checkpoint states (CP1, CP2, etc.) are inserted in the middle of A as a gateway to the extraction flow. When a preemption request is received and the execution arrives at one of the checkpoints (CP2 in the figure), the extraction flow will be initiated. Likewise, when a hardware task is being restored, the restoration flow is carried out before going back to checkpoint. The extraction and restoration (context switch) flows in the state machine include accessing the register and memory contents. They are performed sequentially as they are forwarded in a chain form. Similar to the datapath, adding states in the task consumes some additional hardware and may reduce the performance.

Another aspect which must be considered in the design-based technique is the necessary effort to modify the task design from designers' perspective. As the realization of design-based technique is usually done at fine-grain architectures, adding the scan-chain structures which connect all registers and memories of the task is laborious. The scan-chain insertion in a design-based technique needs to be automated in order to reduce the effort of the designers. With this in mind, a high-level design flow is introduced to generate hardware tasks with scan-chain structures in the next subsection.

4.2.2 *High-Level Synthesis Flow*

While the task modification in a design-based technique can be performed by hardware designers as presented in [JTW07; KHT07; Vu+16], we develop our interest in the HLS flow which is easily reproducible and requires a lower effort than the

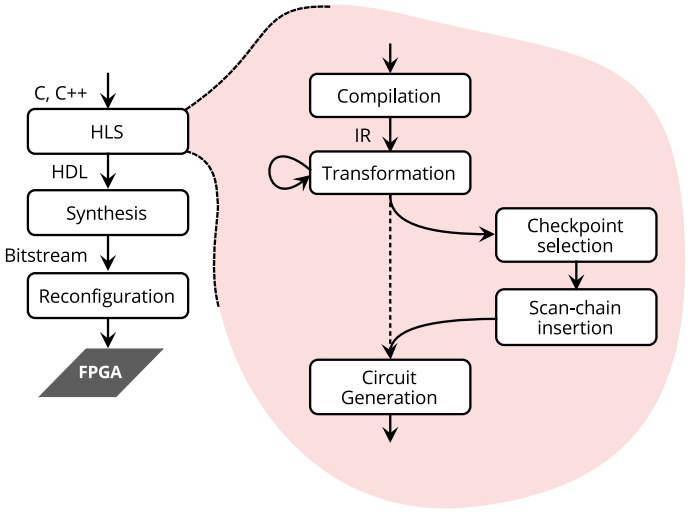


Figure 14: Autonomous checkpoint selection and scan-chain insertion using High-Level Synthesis Flow [BMR16]

former solutions. Bourge, Muller, and Rousseau [BMR16] in their work proposed an automated scan-chain insertion mechanism to hardware tasks which are generated by HLS. They integrate their mechanism as a plugin, namely CP3, in a free and open-source HLS tool AUGH [PMR14]. The flow of task generation and scan-chain insertion is described in Figure 14.

The mechanism proposed in [BMR16] is inserted at the end of the HLS flow, before the *Circuit Generation*. By doing this, the scan-chain structures are added to the hardware task design which has been optimized by AUGH. Their mechanism consists of two steps: checkpoint selection and scan-chain insertion. The checkpoint selection relies on static analysis and greedy heuristic algorithm to select checkpoint states. According to a given time constraint, all the execution states must be covered by the checkpoint states. Thanks to the HLS flow, the introduced mechanism can also search for the less costly selection of checkpoint states in terms of resource consumption. After the checkpoint states are selected, the scan-chain structures that chain the register and memory elements for in and out direction are added to the design. The reader who is interested with the detail of this solution is referred to [BMR16] and [Bou16].

As good as it gets when providing a hardware task context extraction on FPGAs using AUGH + CP3, the solution does not consider the task communication during a preemptive context switch. As a consequence, a hardware task cannot be context switched while having ongoing communication flows. To ensure the continuity of a task execution, we should also manage the I/O communication data associated to the task. The existing scan-chain insertion flow does not handle the memory elements located outside hardware tasks. In the next section, we present the communication model of hardware tasks in reconfigurable architectures.

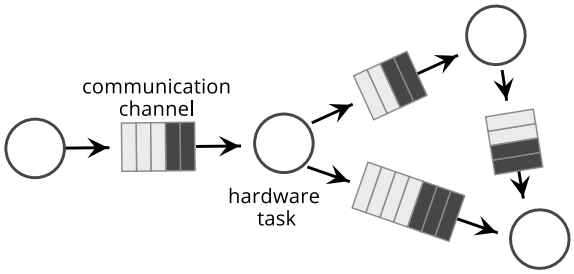


Figure 15: Example of task representation in a reconfigurable architecture using KPN

4.3 COMMUNICATION MODEL

4.3.1 *Kahn Process Network*

Kahn Process Network (KPN) is proposed by Kahn [Kah74] to model tasks and their communication channels. In KPN, a task is considered as an autonomous process that works concurrently with other processes in the network and communicate to each other through infinite FIFO buffers. These conditions results in a blocking operation for reading and non-blocking operation for writing. However, due to a finite amount of memory resources available on the FPGA, the reading and writing operations are actually blocking [GB03]. An example of KPN with several tasks connected to each other through channels is shown in Figure 15.

In this work, we use KPN to model an application in a reconfigurable system, including software tasks on CPUs, hardware tasks on FPGAs, and the communication between them. Due to the timing-insensitive model provided by KPN, the communication can be represented as generic as possible while still being able to describe the interaction between tasks. Each task can run concurrently and asynchronously with a global synchronization is subject to the existence of I/O communication data. The communication channels in KPN use FIFO buffers, or FIFOs, to intermediately store the I/O communication data or tokens while being transferred from one task to another. With this being said, this communication data needs to be managed when a hardware task in KPN is context switched due to preemptive scheduling.

4.3.2 *I/O Communication Scope*

Communication management during a hardware context switch requires a definition of the I/O communication scope of the task with context switch capability. The scope of I/O communication particularly includes the placement of communication channels in the architecture. Use of the HLS tool AUGH + CP3 allows designers to easily code hardware tasks from the desired functionality of acceleration and let the tool instrument them with context extraction support. KPN specification does not

Listing 1: A code snippet describing a simple hardware task in AUGH

```
int32_t process (int32_t input){
...
}

int main(){
   int32_t input_data, output_data; /* variable to store input and output data */
   do{
         input_data = read_input(); /* read one token from input port */

         output_data = process(input_data); /* execute */

         write_output(output_data); /* write one token to output port */
   } while(1);
}
```

restrain the location of communication FIFOs between tasks. It rather depends on the programming model of the tasks themselves.

Listing 1 shows a code snippet used to describe a simple hardware task in AUGH. In this example, we assume that the task has only one input and one output port. Without involving any *pragma*, the execution flow of a task can be generally described in three sequences: reading the input, executing the process, and writing the output. Input reading process is performed by *read_input()* subroutine whereas output writing process by *write_output()* in Listing 1. The task produces an output token from each input token consumed. Communication FIFOs are necessary in the architecture not only to satisfy KPN specifications but also to adjust to different bandwidths of communicating tasks and increase the transfer efficiency. These communication FIFOs are located between tasks in the architecture and not included in the task code. As a result, the CP3 plugin does not manage the extraction and restoration of I/O data inside FIFOs in the scan-chain.

In contrast to Listing 1, Listing 2 presents a code snippet of a hardware task with communication FIFOs coded in the task. These FIFOs are described in forms of array *input_data[32]* and *output_data[32]* which store all tokens during a process [1]. Such implementation allows the tasks to group I/O communication for efficiency. By coding the FIFOs inside the task, the extraction of communication data can be managed together with the task context thanks to scan-chain structures provided by the CP3 plugin. The different possible locations of communication FIFOs according to the programming model is illustrated in Figure 16.

One of the ways to preserve the communication consistency is to extract the I/O communication data which still remain in the FIFOs during a preemptive context switch. Provided the implementation as shown in Figure 16b is taken, I/O communication data and task context will be extracted and restored together during a context switch. However, such an approach requires designers to develop the task

1 32 was arbitrarily chosen as the size of the FIFOs

Listing 2: A code snippet describing a simple hardware task with FIFOs in AUGH

```
int main(){
   int32_t input_data[32], output_data[32]; /* array to store input and output data
        */
   int32_t i;
   do{
      for (i = 0; i < 32; i++)
         input_data[i] = read_input(); /* storing all the input token in an array
             */

      for (i = 0; i < 32; i++)
         output_data[i] = process(input_data[i]); /* execute */

      for (i = 0; i < 32; i++)
         write_output(output_data[i]); /* send all the output token in one go */
   } while(1);
}
```

(a) Communication FIFOs outside the task

(b) Communication FIFOs inside the task

Figure 16: Different location of communication FIFOs in an FPGA

and its communication channels themselves. In many cases, a third-party or legacy IP (task) is used and re-used, which makes the task a black box. To protect the IP, it is sometimes distributed in a netlist form, and therefore is hard to reverse-engineer and modify it. Without the knowledge of the communication scheme of the task with a context switch capability, FIFOs must be added as shown in Figure 16a. At the same time, communication FIFOs placed outside the task can be beneficial, for instance, to share the same channels among the tasks. Prefetching communication data is also possible when the FIFOs are separated from the task.

To summarize, the HLS tool AUGH and its CP3 plugin provide a hardware context switch support on heterogeneous reconfigurable architectures. In contrast to the other solutions, they allow automated flow to instrument the task design with scan-chain structures which significantly reduces designers' effort. However, the buffers outside hardware tasks are not managed in the tool. As a result, the consistency of communication data cannot be guaranteed if a task is context switched while still having ongoing communication flows. A solution at the protocol level is required to ensure the consistency in communication regardless of whether a task is communicating or not when it is context switched.

4.4 CONTEXT SWITCH PROTOCOL

4.4.1 *Existing Solution*

This section presents the protocol which is currently followed in a hardware context switch if the communication data is not managed. The objective is to preserve the consistency in communication data when a hardware task with ongoing communication flows is context switched in order that this task can continue without any error at a later time. We consider that the hardware task in the system has context switch capability using the state-of-the-art solution. It communicates with other tasks in CPUs and other FPGAs through the main interconnect, as illustrated in Figure 11. To support the asynchronous data transfer between tasks, communication FIFOs are used in the channels. A main memory is used to store the context of the tasks in the system.

Figure 17 presents the steps in the current context switch protocol. A hardware task T1 runs on one FPGA and communicates with other tasks in the system. The other tasks can be located in any CPU or FPGA as long as they are connected to the main interconnect. A preemption request is given to T1 when it is in the middle of its execution with I/O communication data stored in the FIFOs on its FPGA side. Despite the preemption request has been acknowledged, the context of T1 cannot be extracted yet as the FIFOs still contain I/O data. Hence, the output link must be maintained to empty the output FIFO and the input link can be disconnected as shown in Figure 17b. After the communication FIFOs located in the respective FPGA are empty (idle), the link can be completely disconnected

(Figure 17c). Figure 17d and Figure 17e show the process of switching the context of T1 to hardware task T2 whose context is stored previously in the memory. After the context switch operation is completed, the communication link of T2 is reconnected to the task which communicates with it.

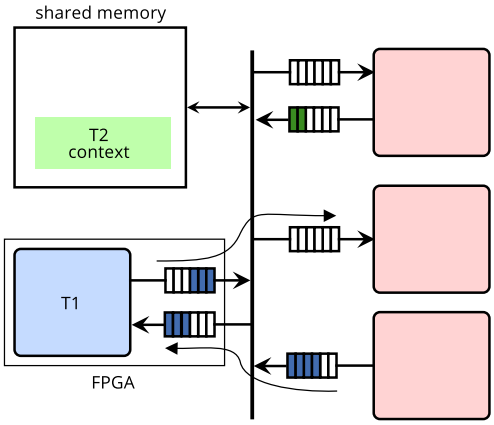
By following this protocol, the communication data integrity can be preserved since the context switch is prevented if there exist communication data inside the FIFOs associated to the preempted task. Among the steps which have been explained, step in Figure 17b requires much attention as it is unpredictable. The input process rate of a task at arbitrary points in time can hardly be known as it strongly depends on its real-time state. Preventing new input data to arrive to a preempted task in order to accelerate the discharge of input FIFO, as presented in [Vu+16], should reduce the waiting time in this solution. Although, the moment when the input FIFO is finally empty remains unpredictable. Furthermore, it is hard to know when the task may send all its output data to its successor. In this regard, the feature where designers can provide a fixed constraint in a hardware context switch [BMR16] is diminished.

4.4.2 *Proposed Solution with Communication Data Management*

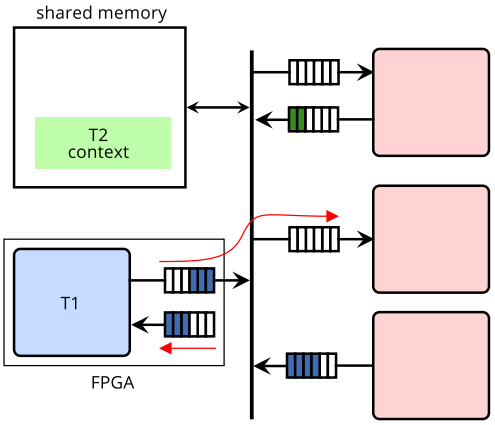
Our idea to preserve the consistency in communication data is simple and straightforward. We propose to perform an extraction to all the attributes of the hardware task in context switch. These attributes are the context and the I/O communication data stored in the FIFOs inside the FPGA. The extraction and restoration of the task context and I/O communication data of a preempted task follow a dedicated protocol that manages the overall context switch process.

Figure 18 describes the steps in the proposed context switch protocol. In contrast to the existing protocol (Section 4.4.1), the connection between the FIFOs of the preempted task and main interconnect can be immediately disconnected, as shown in Figure 18b. Afterward, the context of T1 and its I/O communication data which still remain in the FIFOs when the hardware context switch occurs are extracted (Figure 18c). Likewise, when another task (T2) is reloaded to the FPGA, its I/O communication data will be recharged to the FIFOs along with the restoration of its context. Finally, the connection between task T2 and its predecessor as well as successor can be re-established. By following these steps, the consistency in communication data can be preserved at all times.

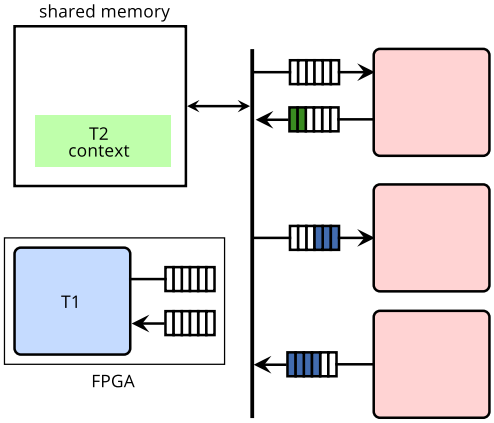
Associating communication data which are stored in the 'local' FIFOs during a hardware context switch with the task context is natural and straightforward. Although the idea is simple, it implies some challenges in exchange for its obvious benefits. First of all, we must have complete control of all communication data in-transit from the sender task to the receiver task in order to disconnect and later reconnect the communication link safely. Losing communication data during the process must be avoided at all cost. Second of all, the details of every one-on-one



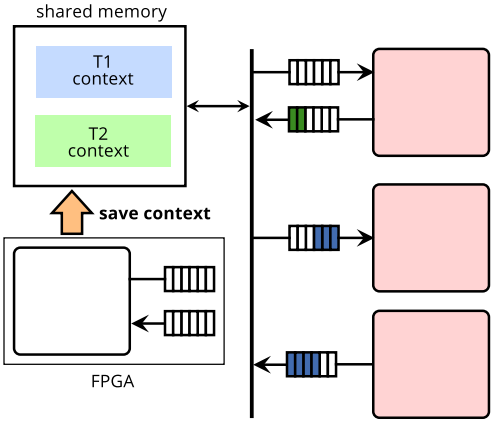
(a) T1 is executing and communicating



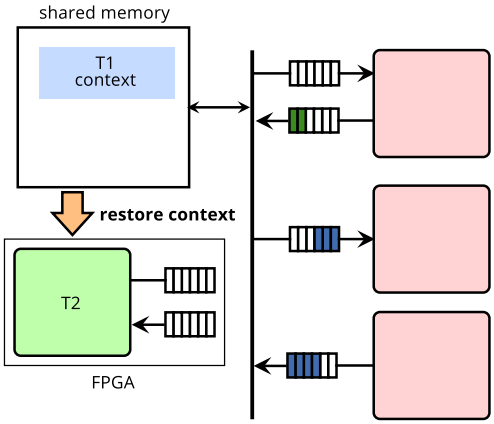
(b) Waiting the FIFO to discharge



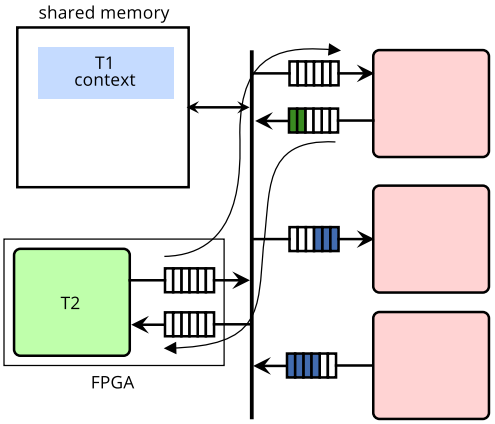
(c) Communication can be safely disconnected



(d) T1 context is extracted and saved

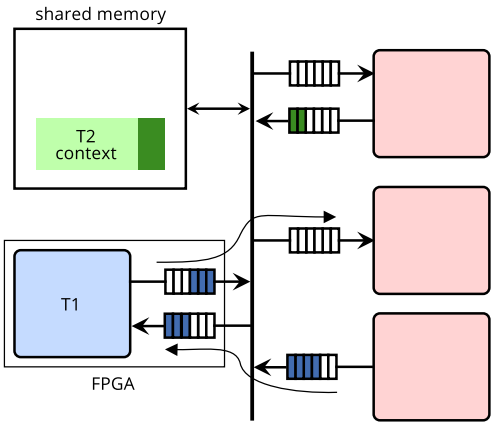


(e) T2 context is restored

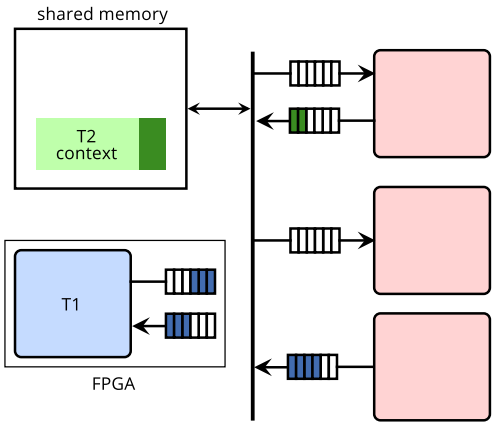


(f) T2 execution continues

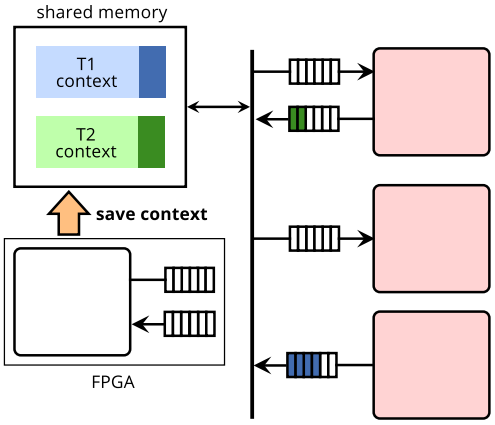
Figure 17: Context switch protocol in the existing solution



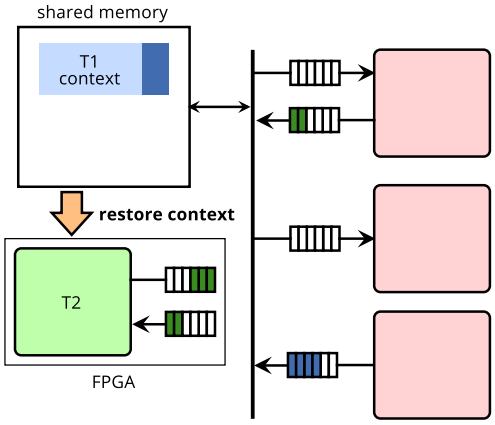
(a) T1 is executing and communicating



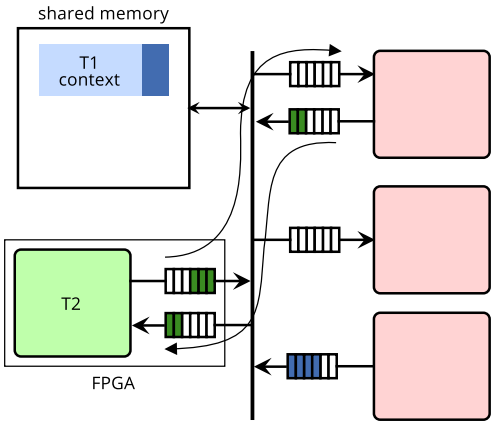
(b) Communication is immediately disconnected



(c) I/O data and context of T1 are extracted



(d) I/O data and context of T2 are restored



(e) T2 execution continues

Figure 18: Steps in the proposed context switch protocol

communication link must be memorized, including when the task is migrated. Since the IPs in FPGAs have FIFO-based ports, they do not include this information in their context. These challenges must be overcome in order to provide the communication management in the context switch protocol.

In comparison with the solution without communication data management, our solution increases data footprint in the task extraction and restoration of the hardware context switch. This results in longer extraction and restoration time in the process. However, it also offers predictability in the process. This predictability is related to the maximum time required to safely interrupt the communication. The method proposed in [BMR16] should guarantee the required latency in regards to the task interruption. As our solution offers control of the communication FIFOs outside the task, the time overhead due to the extraction/restoration of communication data can be estimated (at least the worst case). Furthermore, the size of FIFOs is something that can be defined by designers. The management of I/O communication data consecutively with the task context from the same FPGA only makes sense if the extraction and restoration of the context satisfy a defined time constraint. An evaluation regarding this argument will be presented in Chapter 5.

4.5 IMPLEMENTATION IN RECONFIGURABLE ARCHITECTURES

This section presents the implementation of the infrastructure that supports hardware context switch in reconfigurable architectures. The solution proposed in this work runs at protocol level and it can be implemented in any system which is able to fulfill the requirements presented in Section 4.1. Due to these requirements, not all platforms can adopt our solution, at least without modifying the nature of the protocol. First, we describe the systems which are suitable for our communication solution. Secondly, we present the developments necessary to implement the proposed protocol, particularly in the physical and communication layer.

4.5.1 *Compatible Systems*

Although the proposed solution is based on a generic protocol which can be applied in any reconfigurable architecture which supports hardware context switches, it requires the architecture to respect certain requirements. An instance is required to allocate hardware tasks in FPGAs and manage the communication, we refer to this instance as Task Manager. In many works [VPI05; LP08; Nar+11; Joz+13], this function is done by a module that runs in the CPU of reconfigurable systems. Our solution can be added to these existing software-based managements, although we do not rule out the possibility of placing the Task Manager in the FPGA. All the tasks in the system, both software and hardware, must be 'recognized' by this Task Manager. This can be supported with ease if all the tasks are connected to the same

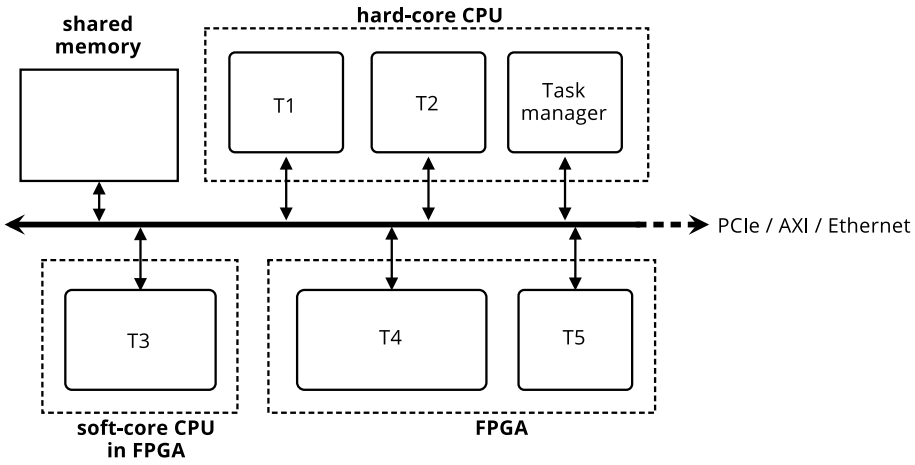


Figure 19: Illustration of a reconfigurable architecture targeted for the proposed context switch solution

interconnect. As a consequence, the same addressing method can be used by all tasks which simplifies the communication link update during a context switch.

Figure 19 depicts the illustration of a reconfigurable architecture which can be targeted by our context switch solution. In the past, soft-core processors (CPUs) were used to manage the hardware tasks in a reconfigurable system. The main interconnect was therefore based on the Processor Local Bus (PLB) standard [Liu+09; Joz+13]. However, such architecture consumes a considerable amount of FPGA resources and the communication speed is limited to FPGA technology. Thanks to PCIe-based communication controller that provides very high transfer bandwidth, e.g., RIFFA [Jac+15], JetStream [Ves+16] and CAPI [Stu+15], the bottleneck in the communication between FPGA and external hard-core processors can be reduced. PCIe-based communication between CPUs and FPGAs in reconfigurable systems is currently the most popular [Afo+13; Che+14; KGS17]. PCIe standard also offers the possibility of connecting multiple FPGAs to the same CPU. Another possibility of CPU-FPGA interconnect is by using Ethernet [ABS10]. With the recent Reconfigurable SoC platforms, the entire reconfigurable system can be placed in one platform that uses internal communication through AXI standards [SSS15].

As shown in Figure 19, software tasks are located in the CPU of the system whereas the hardware tasks are in the FPGA. We can also run software tasks in soft-core processors that use FPGA resources. The software tasks may also support a software-based context switch although it is not covered in this discussion.

We assume that the hardware tasks support the hardware context switch operation as they allow task context extraction. Due to the state-of-the-art design-based method, previously extracted task can also be reloaded to another FPGA in the system (in case of migration). Some of the existing works use Dynamic Partial Reconfiguration (DPR) solution to reduce run-time IP reconfiguration [DML12] and to support IP relocation [Fek+12]. The hardware task migration can therefore be

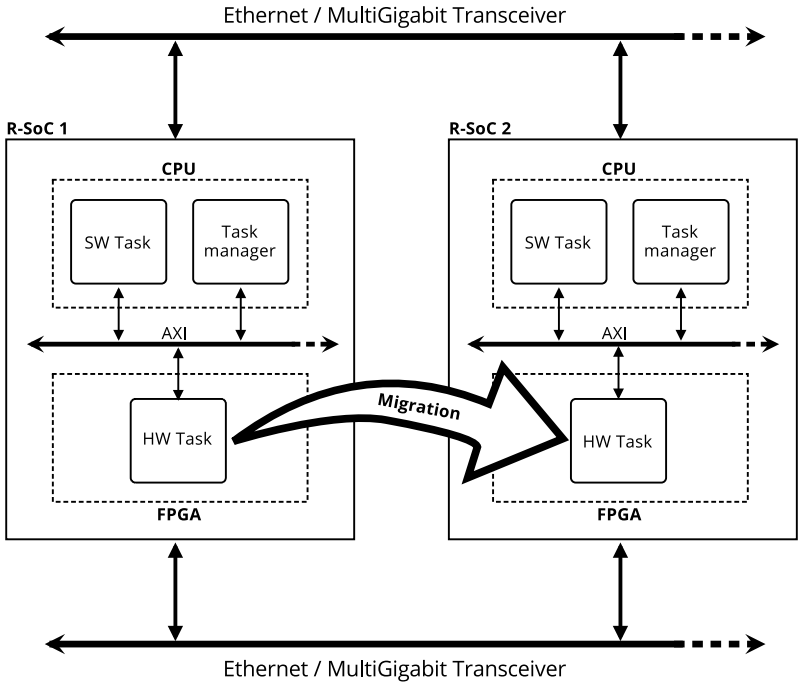


Figure 20: Reconfigurable system built with multiple Reconfigurable SoC platforms requires further development

done between hardware tasks located in different partial regions in the FPGA. Our solution can adapt to such solution with the communication channels placed in the static region.

The proposed context switch protocol, however, is not compatible with a system that consists of multiple Reconfigurable SoC platforms at the moment. As a result, hardware task migration between FPGAs in different Reconfigurable SoC while maintaining the communication, as presented in Figure 20, cannot be treated yet. This is due to the hierarchy in the communication interconnect between the tasks in the system. In addition to the AXI interconnect which connects the CPU and FPGA internally, another interconnect is required between Reconfigurable SoCs, for instance using Ethernet or MultiGigabit Transceiver. Such hierarchical interconnects increases the complexity in the communication between tasks in different Reconfigurable SoCs. Adjusting the communication by the Task Manager in a Reconfigurable SoC must consider the other Reconfigurable SoC. As a result, further development is necessary to enable an implementation of our solution in such an architecture, which is not covered in this thesis report.

4.5.2 *Development in Physical Layer*

The management of hardware task context and I/O communication data during a context switch requires a development in the physical layer. We develop a *com-*

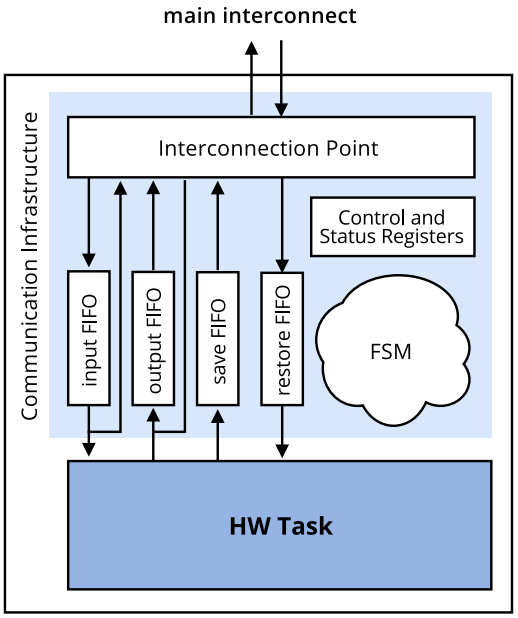


Figure 21: Communication infrastructure to support the preemptive context switch protocol with communication management on FPGAs

munication infrastructure which is essentially applied as a hardware task wrapper in the FPGA. This communication infrastructure is paired to each hardware task in the FPGA in order to provide an interface to the main interconnect. Figure 21 presents the illustration of the communication infrastructure in the FPGA. The communication infrastructure integrates I/O communication FIFOs (input and output), context FIFOs[2] (save and restore), an FSM, and an interconnection point. The FSM is responsible for controlling the communication flow and synchronization during normal execution as well as when a preemption occurs. As this infrastructure connects each hardware task to the rest of the system, the FSM is responsible for forwarding preemption requests to the hardware task. It also manages the data transfer flows among the FIFOs to follow the protocol presented in Figure 18. For FSM configuration and status, we add control and status registers in the communication infrastructure. The Interconnection Point is necessary to perform the protocol conversion between the FIFOs and the external communication architecture. The interface of this interconnection point will depend on the topology of the communication architecture and the technology used in the system.

The I/O communication FIFOs used in the proposed communication infrastructure are modified to allow data extraction and insertion without passing through their standard ports. Multiplexers are added to manipulate the data flow according to the condition of execution. For illustration purposes, we assume that the input FIFO receives input data from interconnection points and forwards them to the task

2 task context is intermediately stored in these FIFOs as the transfer bandwidths are different between inside and outside of FPGAs

in normal task execution. When the hardware task is being context switched, the input data will be transmitted back to the interconnection point to empty the input FIFO. These data will be considered as a part of the task context. The sequence and timing of these FIFO extractions are managed by the FSM. This implementation is simple and straightforward, yet it can guarantee the communication data integrity without waiting until the communication channels are idle.

Figure 22 details the FSM which controls the communication and context FIFOs. It consists of three parts which are connected to each other: FSM read, FSM write, and FSM context. The three FSMs control the input, output, save and restore FIFOs according to the flow. In the normal execution flow, FSM read manages the input data from the main interconnect. Every time there is an external request, it performs the input data transfer until there is no data or the input FIFO is full. When the context is restored, the FSM read restores the communication data and context to the input, output, and restore FIFOs. Likewise, FSM write manages the output data from the hardware task. It performs the output data transfer when the destination is ready to receive and the output FIFO is not empty. When the hardware task is preempted, FSM write saves the data from the input, output, and save FIFOs.

The FSM read and write are configured (controlled) by the control registers whilst they provide the information of their status in status registers. Through these control registers, the command to start, save, or restore a hardware task can be given. These commands control the interaction of FSM context with the hardware task to trigger the execution as well as context switch. The control registers also contain the configuration of connections of the hardware task with other tasks in the system. The status registers hold the information about the size of extracted/restored context and the states of the hardware task (execution or finish).

With the communication infrastructure wrapping each hardware task in the architecture, we have complete control of the in-transit communication data. The Interconnection Point used in the infrastructure only forwards the transmitted data without storing them. As a result, we ensure the preservation of communication data consistency during a hardware context switch.

4.5.3 *Development in Communication Layer*

The communication infrastructure developed in an FPGA is developed to ensure the consistency of communication data in a hardware context switch operation. It allows autonomous control and synchronization of the communication data and context associated to a preempted task inside the FPGA. Figure 23 describes how the I/O communication data and context are managed together. When a hardware task is context switched, its context and the I/O communication data which resides in the same FPGA are saved together in a shared memory. We refer to the set of the context of a task and its associated communication data as a *communication-aware context*. In addition, a communication-aware context includes the amount of data

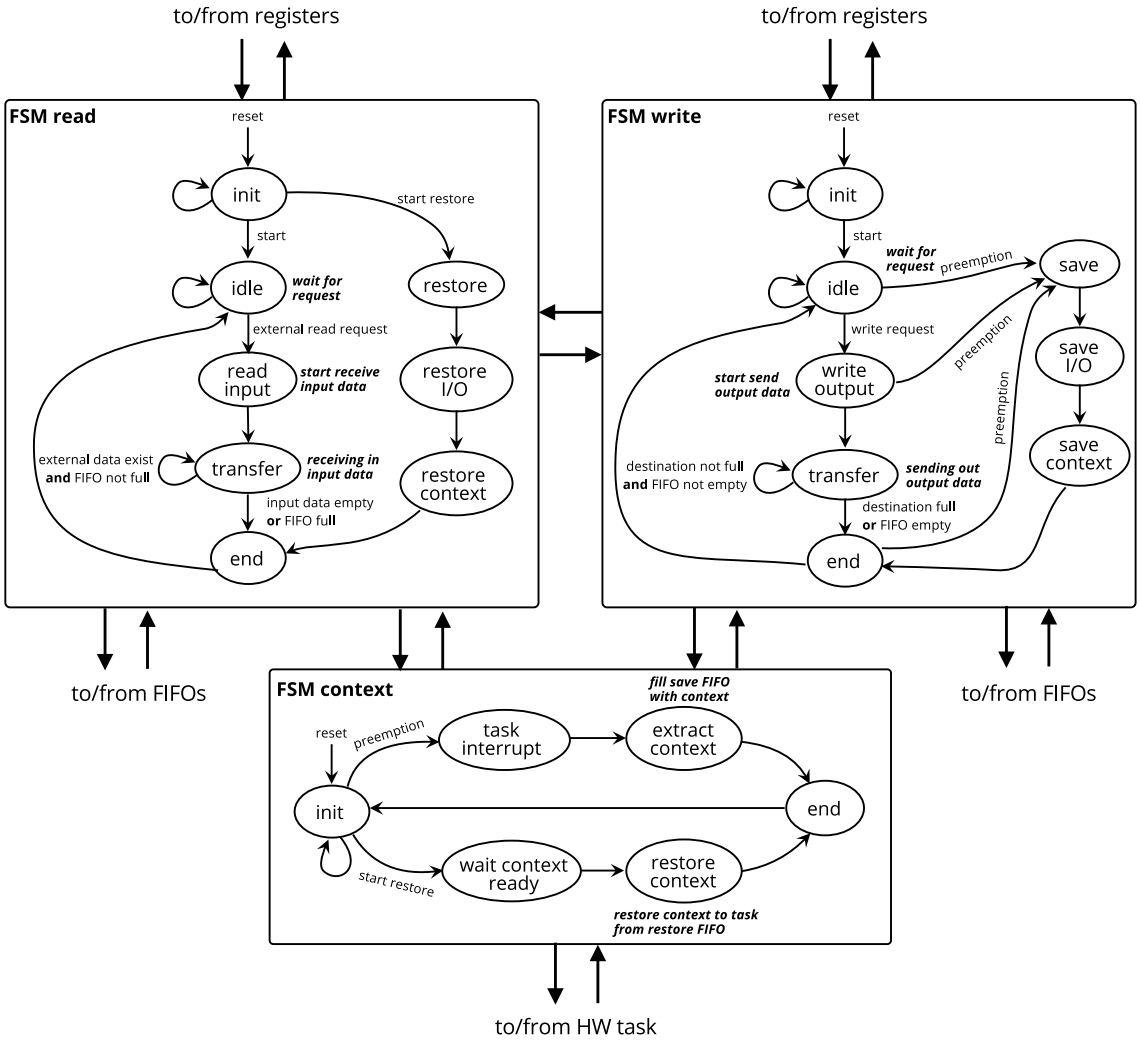


Figure 22: Details of FSM inside the communication infrastructure

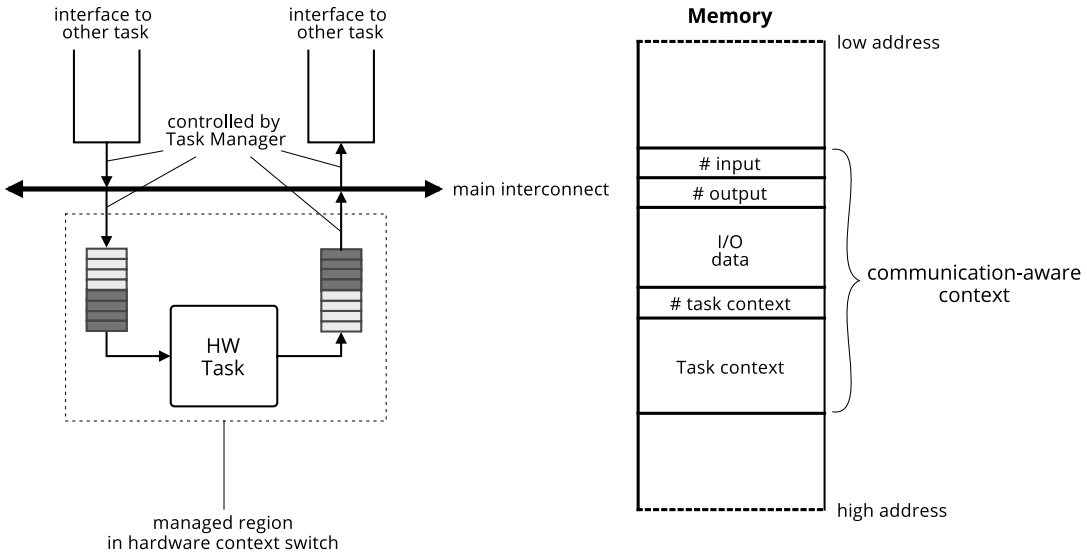


Figure 23: Management of the context and associated communication data of a hardware task being context switched

which acts as a header of the context. This information is particularly necessary to allow the communication infrastructure to perform the restoration of context and communication data autonomously.

As we can see in Figure 23, the communication-aware context stored in the memory does not contain the details of the ongoing communication flows when a hardware context switch occurs. Adjusting the communication link to disconnect FIFOs and later reconnect them when a task is restored either to the same or a different FPGA is done by the Task Manager. Practically, the Task Manager can be a program, driver or a module which runs on the CPU. It can be installed either as a standalone application or as a kernel module if a Linux-based OS is used. Figure 24 describes an abstract view of the Task Manager which controls the communication link of the tasks. It holds the information of the connections between tasks in a table and adjusts them according to the performed context switch in the system, for instance when the task is moved to another FPGA (migration). Besides managing the communication link, the driver also controls the FSM in the communication infrastructure to start the execution as well as to perform the extraction and restoration in a context switch. This control is done by writing values to the control registers and reading values from the status registers. Conversely, the communication infrastructure sends an interrupt (IRQ) to notify the driver of certain conditions, e.g., the end of execution, the end of context extraction, etc.

Thanks to the implementation in communication layer which includes the integration of task context and its associated communication data into a communication-aware context and the management of communication link by Task Manager, we can ensure the continuity of communication between tasks in a reconfigurable ar-

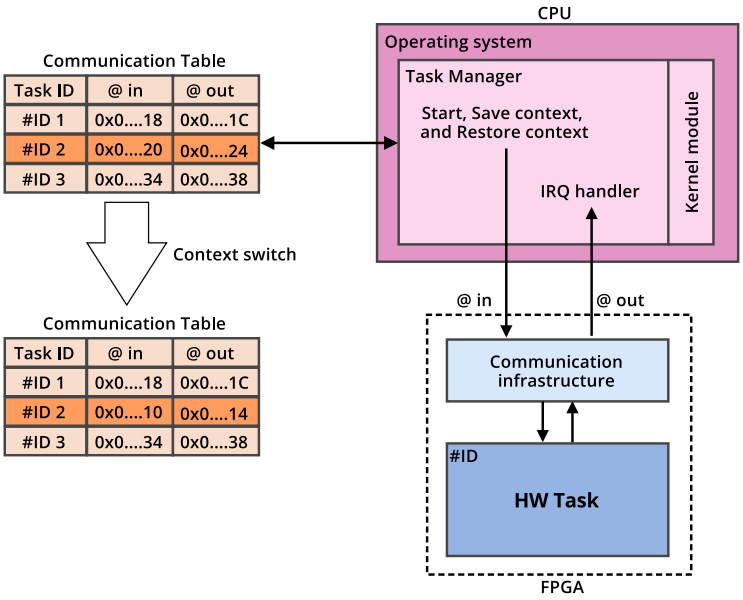


Figure 24: Abstract view of Task Manager which manages the communication link and controls the communication infrastructure on FPGAs. Communication address can be adjusted in case of migration by the Task Manager.

chitecture that supports hardware context switching. In case of a task migration between different FPGAs in the system is involved, e.g., in multi-FPGAs system, the communication link update can be applied as well. The proposed solution in this work is generic provided that the specified requirements are fulfilled.

4.6 CONCLUSION

In this work, we address the challenges which hinder hardware tasks on heterogeneous reconfigurable architectures to be context switched at arbitrary points in time due to preemption requests. The most efficient method to provide a hardware context switch support in heterogeneous reconfigurable architectures is the design-based one with checkpointing. As this method requires selecting checkpoint states and inserting additional scan-chain structures in the task design, higher efforts from the designers are necessary. The recent work proposed in [BMR16] takes advantage of HLS flow to generate hardware tasks or IPs which are already instrumented with the structures to perform context extraction (and restoration). However, it lacks the communication management which prevents a task to be preempted while having ongoing communication flows.

To preserve the communication data consistency in the hardware context switch, the communication between tasks must be managed. The communication data consistency must be preserved and the continuity in communication between tasks should be guaranteed. Throughout this chapter, we introduce a protocol which

manages the overall context switch process including the communication. During a hardware context switch, the associated attributes of the preempted task should be extracted and stored to offer a possibility of task resumption at a later time. These attributes are the context and communication data of the task which resides in the same FPGA. Meanwhile, the communication link should be safely disconnected when a task is preempted. The context and communication data are restored back to the same or a different FPGA (in case of migration) in the system, and the communication links of the task are reconnected when a hardware task resumes. A development in the physical and communication layer is necessary to implement the proposed mechanism in reconfigurable architectures. The physical layer development focuses on the communication infrastructure in FPGAs. The implementation in the communication layer is performed using a Task Manager, which is practically a driver/module which runs in the CPU.

In comparison with the existing solution, our solution ensures the communication data integrity with a predictable overhead, both in performance and resource. The introduced protocol is generic and straightforward and therefore is not limited to a certain communication topology in reconfigurable architectures, although it requires all the tasks to be connected to the same interconnect with a Task Manager which controls the communication. In the next chapter, a detailed evaluation of the performance and overhead in our communication management will be presented.

5

EXPERIMENTS AND RESULTS

THE PREVIOUS CHAPTER qualitatively presents the interests of managing the communication in a hardware context switch, particularly in preserving the consistency in communication, and their solution. This chapter presents the experiments performed to quantitatively measure the performance and overhead of the proposed context switch management.

We begin by presenting the overview and the different platforms used in the experiments. Next, we describe the generation of a FPGA configuration that integrates the IPs produced by AUGH [PMR14] and its CP3 plugin [BMR16] and the infrastructure introduced in our solution. We created scripts to autonomously launch the existing tools and generate the bitstream for the FPGAs in the experiments. Then, we measured the performance in hardware context switch operation on the platforms. For performance comparison purposes, we developed another framework which followed the context switch protocol without communication management in addition to the framework which followed our solution. Finally, we present a prototype of task migration in heterogeneous reconfigurable architectures using two Reconfigurable SoCs and an application of the solution to provide multitasking in the context of hypervisor.

Contents

5.1 OVERVIEW

While providing hardware context switch support in heterogeneous reconfigurable architectures will improve the flexibility in execution, it will also cause overheads to the system. In these experiments, we evaluated the overheads of hardware context switching in regards to the offered performance. Besides the overhead in the hardware utilizations, we particularly focus on the temporal data results when the hardware context switch occurs.

Figure 25 describes an overview of the system used to evaluate the performance and overhead of our solution. The system is built on a Reconfigurable SoC platform that integrates AXI interconnect for the communication between CPU and FPGA. We prepared a simple case where a hardware task (HW Task) is executed on an FPGA and context switched. A predecessor and a successor software tasks (SW Tasks 1 and 2) are placed in the CPU and communicate with the hardware task. The Task Manager that manages the connection between tasks are placed in the CPU as well. The objective is to interrupt the HW Task in the middle while it still has ongoing communication flows and remove it from the FPGA. Then, the HW task is recharged back again to the FPGA to resume the execution. While performing such operation, the performance and overhead are measured. Most importantly, the performed operation must not cause any error to the execution. The output of the HW task is verified with a reference to ensure that there is no error introduced due to the hardware context switch.

First of all, we present the resource utilizations in the hardware due to implementing our solution in FPGAs. Three architectures are built for comparison purposes. The first architecture does not have any support to perform a hardware context

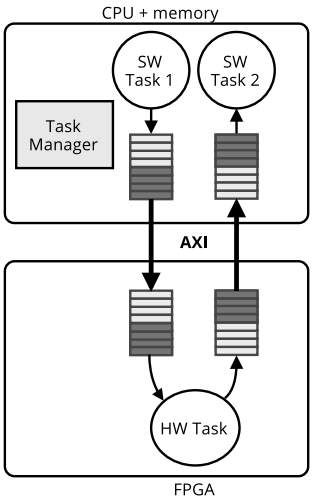


Figure 25: Experimental system overview

switch, which we refer to as *basic*. Naturally, the IPs in this architecture does not allow a context extraction from hardware tasks. The second architecture uses IPs which provides context extraction ability and follows the context switch protocol described in Figure 17 where context switching hardware tasks with ongoing communication flows must be prevented. We refer to this architecture as *CS*. The third architecture, which we refer to as *CSComm*, implements our communication solution in the hardware context switch, as presented in Figure 18.

Next, we present the temporal properties of the context switch operation and the impact of our solution. They are total execution time, context extraction and restoration time, and latency in context switch. These properties were measured in the CS and CSComm architectures using a scenario where a hardware task in a reconfigurable architecture is context switched. The results are compared in order to study the quantitative impacts of our solution besides its communication consistency preservation.

Before presenting the detailed evaluation results of our context switch solution, we introduce the experimental platforms in the next section.

5.2 EXPERIMENTAL PLATFORMS

The architecture presented in Figure 19 can be built using a Reconfigurable SoC platform. Among the available commercial Reconfigurable SoCs, we selected ZC706 Evaluation Board from Xilinx and Arria V SoC Development Kit from Intel FPGA for our experiments.

5.2.1 *Xilinx ZC706 Evaluation Board*

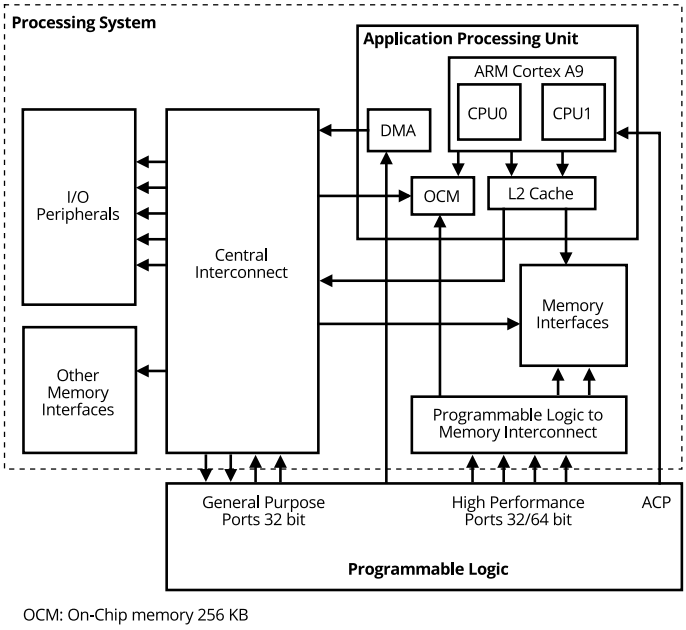
The ZC706 Evaluation Board is based on XC7Z045-2FFG900C chip from Zynq-7000 FPGA family developed by Xilinx. Much like the other Zynq boards, such as Zybo and Zedboard, ZC706 integrates 32-bit ARM cores and FPGA on its chip. Figure 26a depicts a block diagram of the Zynq-7000 architecture. The architecture shown in the figure consists of Programmable Logic, Application Processing Unit, and the connections to the peripheral devices which are all connected to Central Interconnect. In Zynq-7000 architecture, the FPGA part is shown as Programmable Logic (PL) whereas the rest of the system is known as a Processing System (PS). The communication backbone between PL and PS in the Zynq-7000 architecture is provided by AXI interconnect. There exist three types of ports for the AXI communication in this architecture: General Purpose Ports, High-Performance Ports, and Accelerator Coherency Port (ACP). As General Purpose Ports are connected to the Central Interconnect, they provide better efficiency in accessing the peripheral devices from the PL. The ACP facilitates a direct connection to the processor from the PL whereas High-Performance Ports provide high communication bandwidth to the memory. With this wide selection of communication ports, hardware designers are able to choose the ports which optimize their design.

Figure 26b illustrates the ZC706 platform with all the parts including the Zynq-7000 Reconfigurable SoC. With the processor in the SoC, the board can work as a standalone system. Furthermore, ZC706 has a PCIe connector which allows it to be a daughter card. A 1GB DDR3 memory is located in the board which can be accessed from the memory interfaces of the Zynq-7000 SoC. The presence of ARM processors in the ZC706 supports development of an OS-based system. A linux-based OS is particularly available in the community with popular distributions, e.g., Ubuntu and Debian. In addition, Xilinx offers Petalinux distribution [Inc] which is also based on Linux and optimized for their products. Due to the utilization of Linux OS, software development in ZC706 can be done with ease as many drivers are already available in the system.

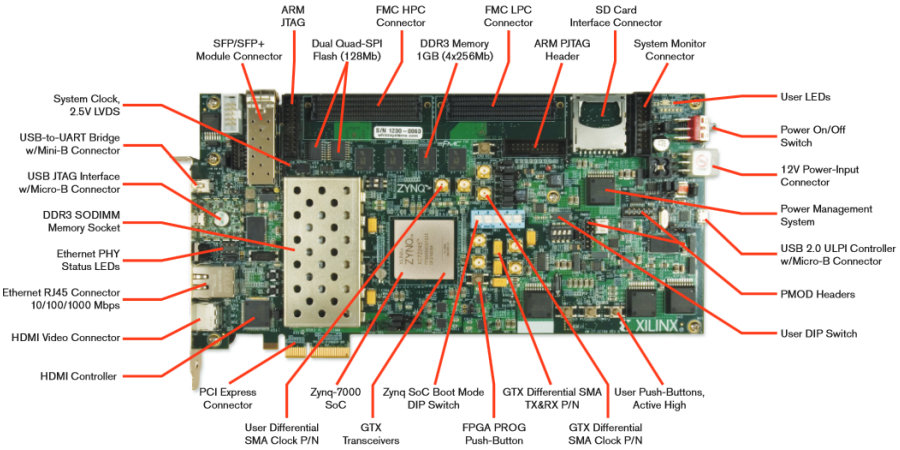
The setup of the ZC706 platform in this work is described in Table 1. Xilinx Vivado Design Suite 2015.3 was used to develop the design in ZC706. We chose 50 MHz as the working frequency of the FPGA to adapt to the benchmark applications which will be presented in Section 5.3.1. The size of a full FPGA configuration in XC7Z045-2FFG900C is 13.3 MB. In this work, we only use one ARM processor core which runs Linaro Ubuntu 15.04.

5.2.2 *Altera Arria V SoC Development Kit*

The Arria V SoC (A5SOC) Development Kit is based on 5ASTFD5K3F40I3N FPGA chip from Altera V SoC family developed by Altera (now Intel FPGA). It was chosen for the experiments because it provides comparable hardware resources as



(a) Zynq-7000 block diagram

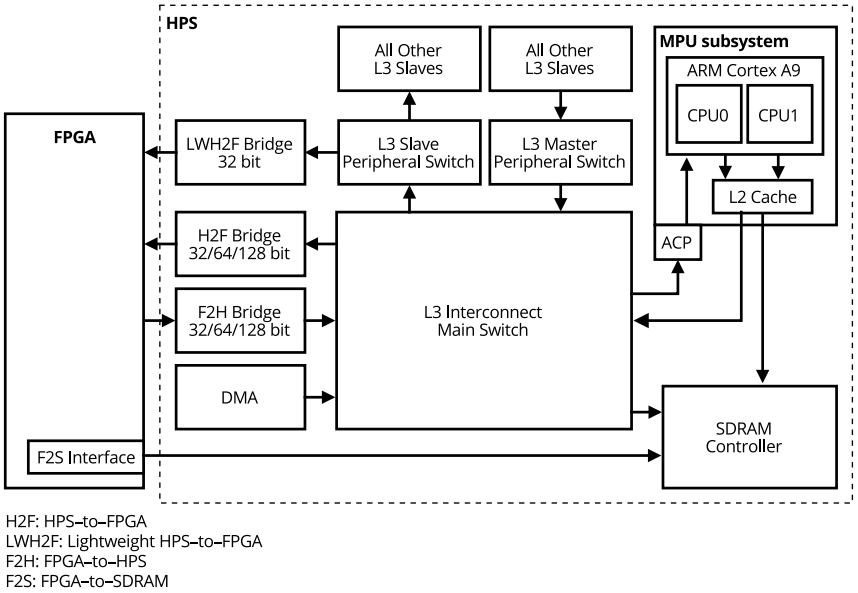


(b) Xilinx ZC706 Evaluation Board

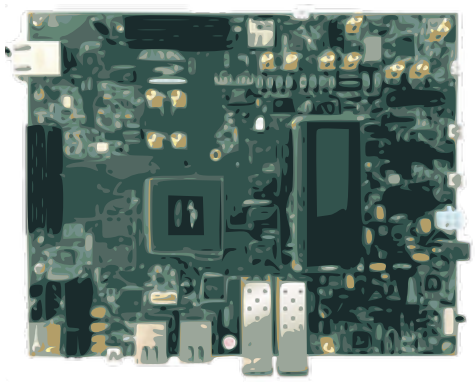
Figure 26: Xilinx reconfigurable SoC platform

Table 1: Experimental setup on ZC706

EDA Tool	Vivado Design Suite 2015.3
FPGA	Xilinx Zynq-7000 XC7Z045-2FFG900C
Clock Frequency	50 MHz
Configuration size	13.3 MB
Host CPU	ARM Cortex-A9
Operating System	Linaro Ubuntu 15.04



(a) Arria V SoC block diagram



(b) Altera Arria V SoC Development Kit

Figure 27: Altera reconfigurable SoC platform

the ZC706. Similar to the other Reconfigurable SoCs, Arria V SoC integrates 32-bit dual-core ARM processors and other peripheral devices in the chip. Figure 27a illustrates the Arria V SoC architecture in a block diagram. The architecture consists of FPGA, Microprocessor Unit (MPU) subsystem, and the switches to peripheral devices which are all connected to L3 Interconnect Main Switch. In Arria V SoC, the MPU subsystem and the connections to peripheral devices are grouped together as a Hard Processor System (HPS). The FPGA can communicate to the components in HPS through available bridges in the architecture: Lightweight HPS-to-FPGA (LWH2F), HPS-to-FPGA (H2F), and FPGA-to-HPS (F2H) bridges. In addition, the FPGA can access the SDRAM of the board via FPGA-to-SDRAM (F2S) interface. These bridges and interface use the AXI communication standard.

Table 2: Experimental setup on A5SOC

EDA Tool	Altera Quartus II 15.0
FPGA	Altera 5ASTFD5K3F40I3N
Clock Frequency	50 MHz
Configuration size	23.2 MB
Host CPU	ARM Cortex-A9
Operating System	Linaro socfpga_arria5

Figure 27b depicts an image of Altera Arria V SoC Development Kit. It offers various connectors, e.g., Ethernet, USB, UART, etc. On top of that, the platform contains 1GB SDRAM accessible from FPGA and 1GB SDRAM accessible from HPS. The support of Linux-based OS from Altera as well as from the community (Rocketboards) is also available in A5SOC. They provide Linaro socfpga and Angstrom distributions which simplify software developments due to provided drivers to access the hardware in the platform.

Table 2 presents the setup for our experiments in A5SOC. The hardware design for A5SOC in this work was developed using Altera Quartus II 15.0. To provide a similar test environment as ZC706, we chose 50 MHz as the working frequency in the FPGA. The size of a full FPGA configuration file of A5SOC is 23.2 MB. We use one of the ARM cores in the MPU subsystem that runs a Linaro socfpga_arria5 distribution provided by Altera.

5.2.3 *Platform Comparison*

The ZC706 and A5SOC are Reconfigurable SoCs that use different FPGAs. Not only do they have different architectures, but they also use technologies from different FPGA manufacturers. The different platforms are used in this work to show that our solution can be used in heterogeneous reconfigurable architectures. As we propose a generic protocol in the hardware context switch, it can work on any FPGA architecture used in the system. The decision to base our work on a design-based technique for task context extraction [BMR16] also allows us to migrate hardware tasks between different FPGAs. Unfortunately, our communication solution is limited to FPGAs with a common interconnect as already explained in Section 4.5.1. As a result, migrating a hardware task from a FPGA in a Reconfigurable SoC to a FPGA in another Reconfigurable SoC will involve the entire application which will be presented in Section 5.6.1.

Besides the difference of the architectures and FPGAs used in the ZC706 and A5SOC, there exist similarities between them as well. Both the ZC706 and A5SOC

are Reconfigurable SoCs that use the AXI communication standard for their internal connection, which is implemented in most commercial Reconfigurable SoCs. As a consequence, the AXI communication interface can be used for the Interconnection Point of the communication infrastructure in the FPGA (cf. Figure 21). In our design, the Interconnection Point is the only part of the communication that must be adapted to the main interconnect of the system. If the system uses NoC topology, for instance, to facilitate the communication between tasks, the Interconnection Point will be a NoC router/switch. With the AXI interconnect in both platforms, the same design can be used in our experiments, which simplifies our implementation.

As described in the previous sections, both Reconfigurable SoC platforms also present 32-bit dual-core ARM processors. A similar Linux-based OS, therefore, can run on the processors of both platforms. Clearly, some details such as the OS kernel version, available memory size on the board, and the peripheral addressing map will cause some differences in the drivers/modules code on both platforms. However, the same processor architecture will result in more or less a similarity in software development, especially for the Task Manager that must be implemented to manage the communication link between tasks in the system.

5.3 HARDWARE IMPLEMENTATION

5.3.1 *Benchmark Applications*

To evaluate the proposed communication management in the hardware context switch, third-party IPs were used. These IPs are instrumented with the structures that allow context extraction and restoration through a specific interface at checkpoint states. We used HLS tool AUGH with CP3 plugin to prepare these IPs. The IPs themselves are chosen from the benchmark applications which have been tested by the same tool. They are a subset of the CHStone benchmark suite [Har+08] for HLS tool and Inverse Discrete Cosine Transform (IDCT) application. Among the benchmark applications, we selected the ones that can work at frequency 50 MHz or higher and consume memory elements on the FPGA. They are the following benchmark applications:

1. ADPCM

   ADPCM (Adaptive Differential Pulse Code Modulation) implements the CCITT G.722 ADPCM algorithm for voice compression. It includes both encoding and decoding functions, which can be pipelined. The two functions can be also used as independent benchmark programs.

2. AES

   AES (Advanced Encryption Standard), also known as Rijndael, is a symmetric key cryptosystem. The AES program includes both encryption and decryption functions, which can also be used as two benchmark programs.

3. BLOWFISH

   BLOWFISH implements a symmetric block cipher. The BLOWFISH program contains only the encryption function.

4. GSM

   This is a program for LPC (Linear Predictive Coding) analysis of GSM (Global System for Mobile Communications), which is a communication protocol for mobile phones. This program implements only lossy sound compression of GSM.

5. IDCT

   IDCT (Inverse Discrete Cosine Transform) returns the inverse of the discrete cosine transform function. It is frequently used in the lossy compression of audio and images applications.

6. MOTION

   MOTION decodes a motion vector formatted according to the MPEG-2 standard, which is one of the decompression methods of video, audio and so on.

7. SHA

   SHA (Secure Hash Algorithm) is a cryptosystem consisting of a set of hash functions. This SHA program is written so as to conform with Netscapes SSL.

5.3.2 *IP Generation with AUGH*

In this work, we used AUGH to generate the IPs which were implemented in the FPGA. It is a free and open source HLS tool under Affero General Public License version 3 (AGPLv3.). AUGH receives the behavioral description of a task written in C for High-Level Synthesis and compiles it to an IP in Register Transfer Level (RTL). The plugin CP3 which instruments the IPs with context switch capability is called during the compilation, as presented in Figure 14.

At the time of the writing of this report, AUGH can only be launched via command line. Listing 3 shows a typical AUGH command line to generate an IP for FPGA. The tool is launched by a keyword **augh** and the associated parameters. In Listing 3, the IP is generated for **xilinx** platform, i.e., **xc7z045** chip. This chip has speed grade 2 (XC7Z045-**2**FFG900C). The working frequency intended for the IP is 50 MHz. Besides the command parameters, AUGH can accept more specific configurations by calling a script during the execution. More on the details of how to use AUGH are explained in the user guide found in [Lab].

The command to activate the CP3 plugin is located in the compilation script (shown as **ip.script** in Listing 3). Listing 4 presents the CP3 plugin part in the script. The width of the scan-chain structures is 32-bit. In the CP3 plugin, the maximum

Listing 3: Generating IP for FPGA via AUGH command line

```
# augh -v -p xilinx -chip xc7z045 -speed 2 -freq 50M -script ip.script
```

Listing 4: Script to call CP3 plugin in AUGH

```
plugin load cp3
plugin cmd cp3 init
plugin cmd cp3 set_cp_id_width 32
plugin cmd cp3 set_sc_width 32
plugin cmd cp3 set_log 0
plugin cmd cp3 set_costfun_type area
plugin cmd cp3 set_sc_type semifull
plugin cmd cp3 set_mem_sort_type cell
plugin cmd cp3 set_reg_stack_type fewhole
plugin cmd cp3 set_cp_mode escape
plugin cmd cp3 set_mem_analysis_type no
plugin cmd cp3 set_cs_latency 5000
plugin cmd cp3 cp
```

latency of context extraction/restoration can be defined by the designers (it is **5000** in Listing 4). This latency includes the total time to arrive at a checkpoint state and extract the task context in that state. The other CP3-related configurations in the script were used in this work by default, e.g., set_sc_type, set_cp_mode, etc. Unfortunately, the user guide of CP3 is not available at the time of writing of this report.

The results of IP generation using AUGH are written in the VHDL language. Listing 5 presents the I/O ports found in the top level file of the IP. The standard I/O interfaces in an AUGH-generated IP is handshake protocol. Both input and output interfaces have data, rdy, and ack ports. It has a signal to trigger the start of execution and an indicator can be added to notify the end of execution. In addition, the CP3 plugin enables context extraction and restoration through the CP3 interface which uses the handshake protocol as well. The infrastructure around the IP, therefore, must adapt to this interface.

5.3.3 *Communication Infrastructure*

The design-based context switch ability is added using the HLS tool AUGH whereas the communication infrastructure is implemented in the Verilog language. The implementation of the communication infrastructure followed the design illustrated in Figure 21. As the design is implemented in Reconfigurable SoC platforms, AXI interface is chosen as the Interconnection Point to support the communication between a hardware task to the rest of the system.

For comparison purposes, we built three versions of the communication infrastructure. The first version was built to give the idea of the communication infrastructure

Listing 5: Example of input–output port definition in top level file of the generated IP

```
entity top is
    port (
        clock : in std_logic;
        reset : in std_logic;
        start : in std_logic; -- start the execution
        finish : out std_logic; -- finish signal
        -- fifo_in
        stdin_data : in std_logic_vector(31 downto 0);
        stdin_rdy : out std_logic;
        stdin_ack : in std_logic;
        -- fifo_out
        stdout_data : out std_logic_vector(31 downto 0);
        stdout_rdy : out std_logic;
        stdout_ack : in std_logic;
        -- cp3 ports
        cp_en : in std_logic;
        cp_ok : out std_logic;
        cp_rest : in std_logic;
        cp_din : in std_logic_vector(31 downto 0);
        cp_dout : out std_logic_vector(31 downto 0)
    );
end top;
```

necessary for an IP which does not have hardware context switch ability. We refer to this version as Basic. The second version of communication infrastructure was built for IPs that can be context switched. However, it does not support I/O communication data extraction. We refer to this version as CS. The third version used our solution to manage the communication in a hardware context switch and is referred to as CSComm.

5.3.3.1 *Basic*

Figure 28 shows a schematic view of an AXI-based communication infrastructure for IPs without context switch ability. It consists of AXI interfaces, an FSM, and FIFOs. Two AXI ports were implemented in the design for the communication with the main interconnect; they are AXI-Lite and AXI (full). The AXI-Lite port was used to provide the control from the Task Manager via control and status registers while the AXI port was for the communication data transfer. The FSM triggers the read and write transfers from the FIFOs to the other tasks in the system.

5.3.3.2 *Without Communication Extraction (CS)*

The second version was the communication infrastructure that supported hardware context switch but did not manage the communication during the process. The schematic of this version of communication infrastructure is presented in Figure 29. Similar to the Basic version, CS integrates an FSM, FIFOs, and AXI interfaces. In addition, the context FIFO was implemented to the communication infrastructure

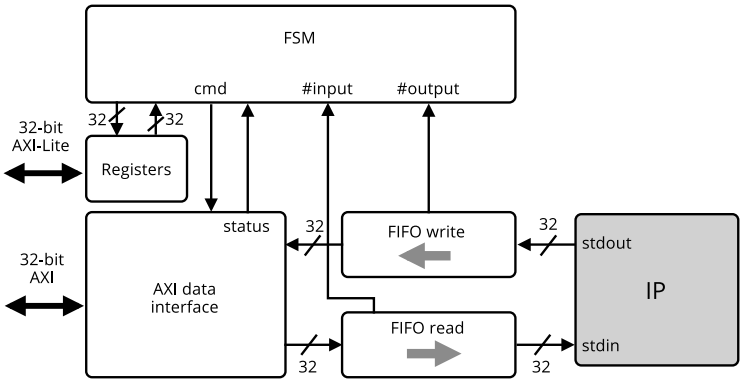


Figure 28: Schematic of AXI-based communication infrastructure for IPs without context switch ability

as the IP supported the context extraction/restoration through its CP3 interface. To share the AXI ports between the I/O communication and the context, multiplexers are added in the design.

The context switch operation using this communication infrastructure follows the protocol described in Figure 17. The FSM controls the flow of data transfer in the FIFOs. When a preemption request is received, the input data flow from external to FIFO read will be blocked. Meanwhile, the output data produced by the IP will be sent from FIFO write to external as soon as the destination is ready. Although the architecture has some differences, a similar solution was also presented in another work in the literature [Vu+16]. Unfortunately, they did not present the details of the hardware and its consumption. For this reason, this version of communication infrastructure was developed by ourselves.

5.3.3.3 *With Communication Extraction (CSComm)*

The third version of communication infrastructure that we implemented in this work is the one with communication data extraction in hardware context switch in order to follow the protocol presented in Figure 18. To provide the communication data management in the hardware, multiplexers were added and a modification was performed to the FSM. Figure 30 presents the schematic of the communication infrastructure that implements our communication solution.

5.3.3.4 *Hardware Resource Evaluation*

This section presents the evaluation of hardware consumption in the three versions of communication infrastructure which were explained previously. The designs of communication infrastructure are synthesized using the EDA tool for each FPGA vendor. The results for the ZC706 are presented in Table 3 whereas the ones for the A5SOC are in Table 4.

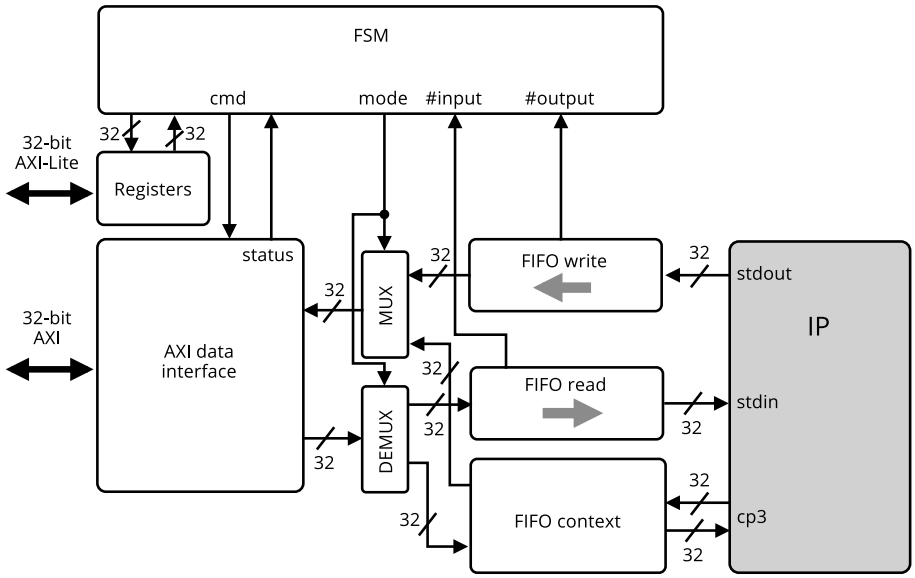


Figure 29: Schematic of AXI-based communication infrastructure without communication data extraction for IPs with context switch ability

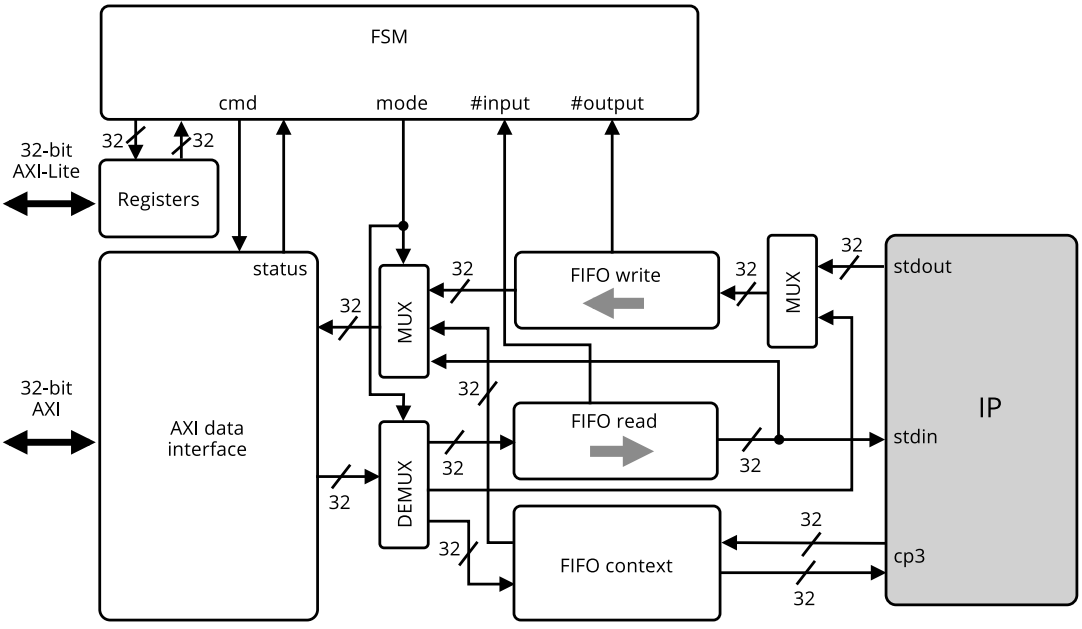


Figure 30: Schematic of AXI-based communication infrastructure with the proposed communication solution for IPs without context switch ability

Table 3: Resource utilization (post-placement) of the communication infrastructures synthesized for ZC706 in Vivado 2015.3

Version	Components	LUT	FF	BRAM
Basic [a]	AXI interfaces + registers	454	590	0
	FSM + mux	229	117	0
	I/O FIFO	0	0	1
	Total	**683**	**707**	**1**
CS [b]	AXI interfaces + registers	454	590	0
	FSM + mux	641	543	0
	I/O FIFO	0	0	1
	Context FIFO	0	0	6
	Total	**1095**	**1133**	**7**
CSComm [c]	AXI interfaces + registers	454	590	0
	FSM + mux	855	626	0
	I/O FIFO	0	0	1
	Context FIFO	0	0	6
	Total	**1309**	**1216**	**7**

[a]Without hardware context switch support
[b]Existing hardware context switch approach
[c]Proposed solution

Table 4: Resource utilization (post-placement) of the communication infrastructures synthesized for A5SOC in Quartus II 15.0

Version	Components	ALM	REG	M10K
Basic [a]	AXI interfaces + registers	411	581	0
	FSM + mux	65	179	0
	I/O FIFO	0	0	2
	Total	**476**	**760**	**2**
CS [b]	AXI interfaces + registers	411	581	0
	FSM + mux	499	801	0
	I/O FIFO	0	0	2
	Context FIFO	0	0	24
	Total	**910**	**1382**	**26**
CSComm [c]	AXI interfaces + registers	411	581	0
	FSM + mux	642	897	0
	I/O FIFO	0	0	2
	Context FIFO	0	0	24
	Total	**1053**	**1478**	**26**

[a]Without hardware context switch support
[b]Existing hardware context switch approach
[c]Proposed solution

As the FPGA chip in A5SOC comes from a different vendor than ZC706, and, therefore, they use different technology, the results are not quite comparable. The logic, register, and memory utilizations are presented in different units in both platforms. Nevertheless, we are able to observe the impact of adding features in the communication infrastructure for each platform. The three version used the same AXI interfaces where they consumed the same amount of hardware resources in the FPGA.

The difference between each version is particularly observed in the resource utilizations of FSM, multiplexers, and the FIFOs. Our measurement shows that CS consumed the logic resources in the FPGA almost three times as much as the Basic version due to adding the support of hardware context switch in ZC706. In A5SOC, the resource utilization increase was even higher. We showed that the complexity of the communication infrastructure increased due to enabling task context extraction and restoration. Indeed, it may be relative to the designers' coding ability when adding the support. However, we want to show that the complexity in the communication environment also increases due to adding a context switch ability to hardware tasks. Although, the environment has less effect to the performance of the tasks than the scan-chain structures which are directly integrated to hardware tasks. Likewise, CSComm consumed more FPGA resources than CS as the communication management was added in the context switch protocol. However, the complexity added to the infrastructure was not high as we only observed an increase of approximately 33% (214) for LUTs and 15% (83) for FFs in ZC706. In A5SOC, the observed increases were 31.2% (143) for LUTs and 12% (96) for FFs.

The other components in the communication infrastructures are the FIFOs for I/O communication and context. They consume the BRAMs (Xilinx) or M10Ks (Altera) in the FPGA. In our experiments, we fixed the data width to 32-bit words in our designs. For FIFO size of 1024 × 32-bit (4 Kbytes data), 1 BRAM 36K was consumed (or 4 M10Ks in Altera). As we considered KPN in the communication model which does not include timestamp in the communication, the size of the communication FIFOs was not important. In these experiments, we chose 256×32-bit for the I/O FIFO size which was sufficiently big for all the benchmark applications used. The necessary context FIFO size, however, depends on the context size of each IP. To give an idea of how many BRAMs (or M10Ks) are required, we present the context size of each benchmark application in Table 5. We fixed them at 3072×32-bit (12 KBytes) for Save FIFO and Restore FIFO to adjust to the maximum context size (motion). Both CS and CSComm consumed the same amount of BRAMs and M10Ks in the FPGA.

In summary, the proposed communication solution extends the feature in the existing hardware context switch support. It preserves the communication data integrity inside the FPGA at a cost of FPGA resources. The resource overhead is particularly due to the FSM and multiplexers which consumes more LUTs and FFs. The increase, however, is considered modest considering one-third of the total hard-

Table 5: Task context size extracted from IP

App	Context (bytes)
adpcm	1996
aes	2668
blowfish	4408
gsm	844
idct	628
motion	8400
sha	496

ware resources was already for AXI interfaces. Regarding the memory elements in the FPGAs (BRAM or M10K), our solution did not cause any impact to the consumption.

5.3.4 *Generation of FPGA Configuration File*

This section describes the flow in generating the FPGA configuration file or bitstream for each Reconfigurable SoC platform from the previously explained IPs and communication infrastructures. The aim was to obtain the bitstream that would be used to configure the FPGA for each benchmark application. Figure 31 and Figure 32 present an illustration of bitstream generation for both Xilinx and Altera platforms using each vendor toolsuite. We developed scripts to perform the flow autonomously, from a benchmark application which is described in C language until a bitstream file.

The applications in C were compiled using AUGH (with CP3 plugin) to generate the circuit descriptions in VHDL. At the same time, the information of port width was taken from the application and used to configure the port width of the communication infrastructure. After the IP was generated by AUGH, the information of the task context size was also used particularly to define the context FIFOs. The results were the communication infrastructure and the IP of the application written in HDL. Until this step, the execution flow for both platforms are identical.

For ZC706, Vivado Design Suite 2015.3 was used to generate the bitstream from the HDL codes. The tool packaged each IP and integrated them in a project. Afterward, the synthesis process was performed which was followed by the place-and-route step. Finally, the bitstream was generated and .bit file was received. The process inside Vivado was controlled by TCL scripts.

For A5SOC, Quartus II 15.0 was used to generate the bitstream. The IP packaging and project integration were done in Qsys, a system integration tool in Quartus.

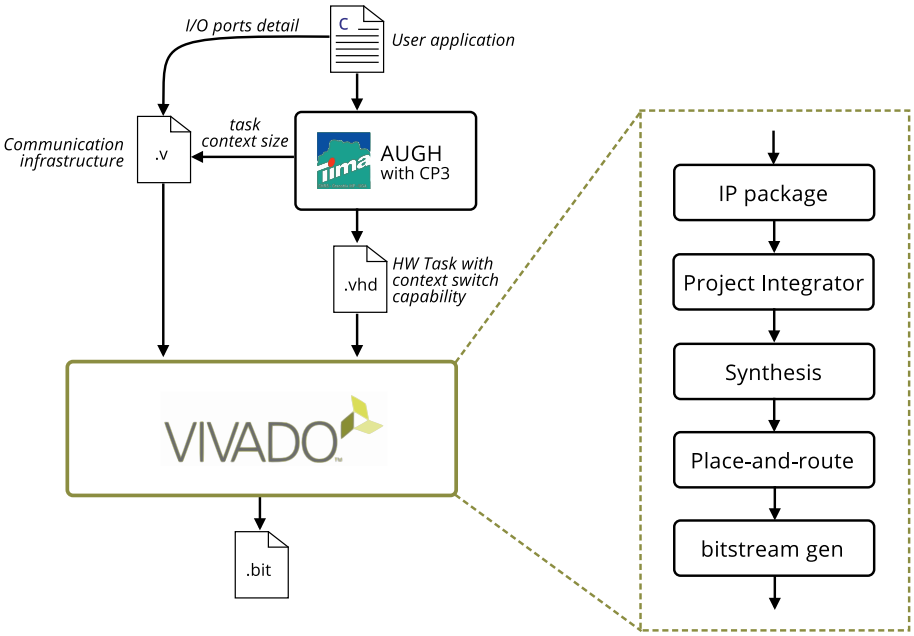


Figure 31: Autonomous flow of bitstream generation on Xilinx toolsuite

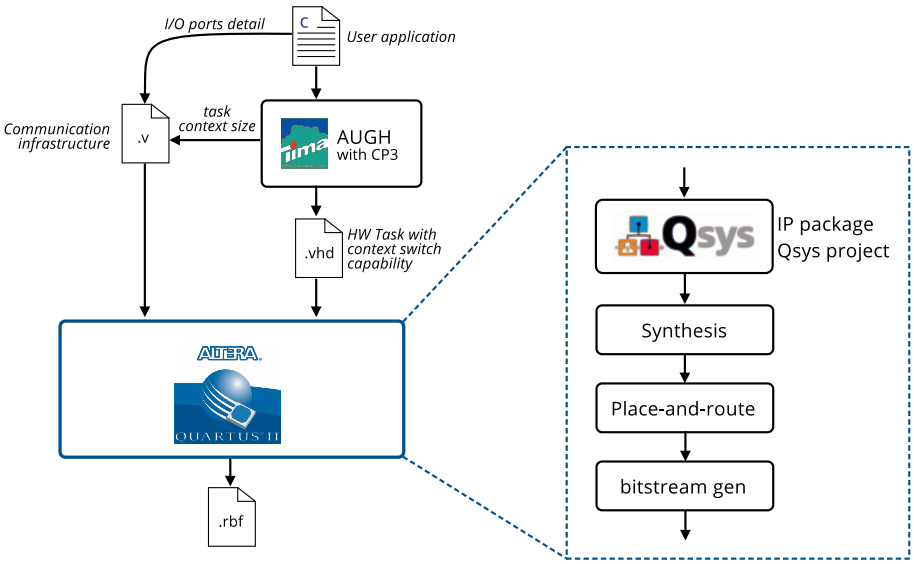
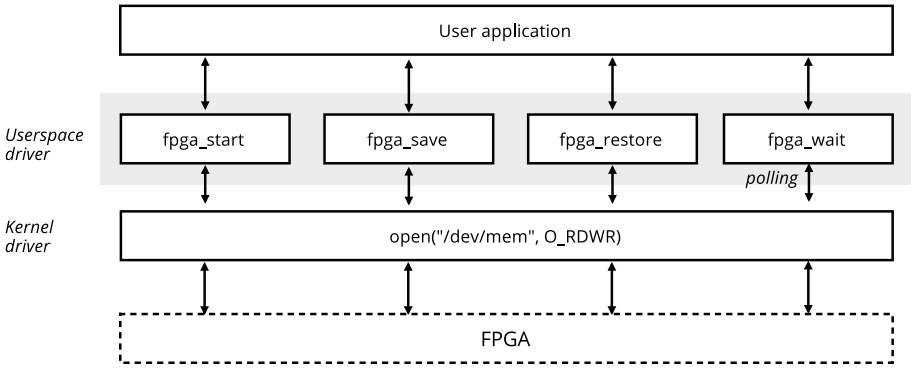
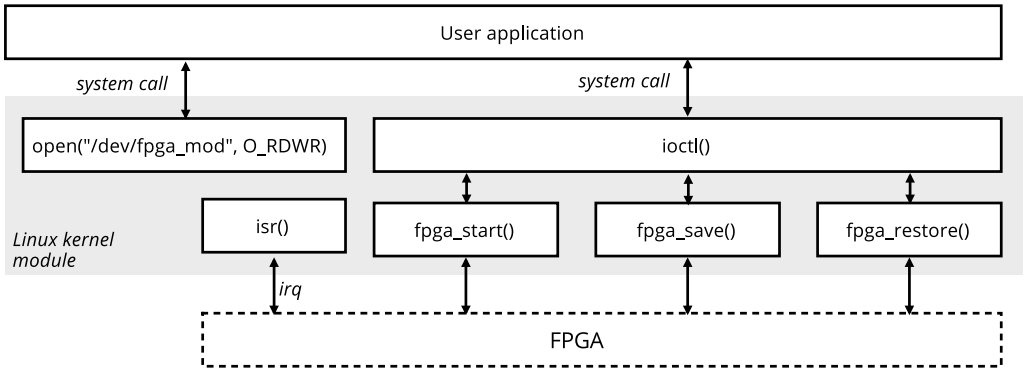


Figure 32: Autonomous flow of bitstream generation on Altera toolsuite



(a) Software implementation using userspace drivers



(b) Software implementation using kernel modules/drivers

Figure 33: Software architecture to access the FPGA in the experiments

Afterward, the steps were similar with Vivado, from synthesis to bitstream generation. The steps in Quartus were performed using a combination of TCL scripts for Qsys and bash scripts for synthesis, place-and-route and bitstream generation.

5.4 SOFTWARE IMPLEMENTATION

The software implementation was performed in Linux for both ZC706 and A5SOC platforms. Drivers are developed to control each hardware task in the FPGA and trigger commands. There exist two ways to implement these drivers. They can be implemented either as Userspace drivers or as Linux kernel modules. Figure 33 illustrates the possible software architectures for the experiments. We tried both methods and there was no significant difference in performance between them, except that kernel module implementation provided tidier access. In general, the drivers were needed to trigger start, save, and restore commands (fpga_start, fpga_save, and fpga_restore). When the drivers are implemented in the kernel space, they could treat the interrupt signal (irq) which were received from the communication infrastructure, for instance to notify the end of execution. In the userspace drivers, the application had to read the status registers by polling to verify that the execution was terminated.

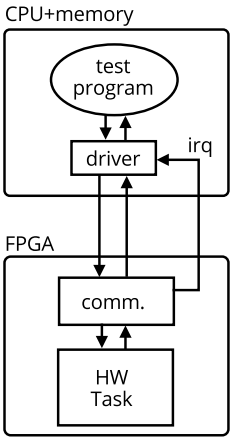


Figure 34: A program to evaluate the performance of the communication solution in a hardware context switch

For the experiments, a real scheduler was not necessary since the objective was to measure the performance of the solution. Instead, we developed a program written in C language that could send preemption requests at a specified time. The Task Manager and software tasks in the processors that communicated with the hardware task in the FPGA were also integrated into the program that sent preemption requests. Figure 34 illustrates this program. It accessed the driver that controls the task in the FPGA through a file descriptor, e.g., /dev/mem or /dev/fpga_mod. As this implementation was done in software, it did not add any hardware resource consumption.

5.5 PERFORMANCE EVALUATION

This section presents the results of the performance evaluation in the experiments. The evaluation mainly compares the context switch performance between communication infrastructures with CS and CSComm versions to see the quantitative benefits of our solution. The performance evaluation was done on ZC706 and A5SOC platforms for every benchmark application, except for SHA which did not run properly in A5SOC due to a low performance issue. To maintain the FPGA working frequency at 50 MHz, we excluded SHA application on A5SOC. As the FPGA clock rate is fixed for every benchmark application, the time measurements do not reflect the real performance of the tasks and communication infrastructure. For this reason, the measured time is presented in FPGA cycles.

Firstly, we describe the evaluation scenario which includes a hardware context switch operation. Then, we present the measurement results of total execution time when a context switch is performed. Afterward, the extraction and restoration time of context and associated I/O communication data are presented. Finally, we inspect the effect of our solution to the context switch latency.

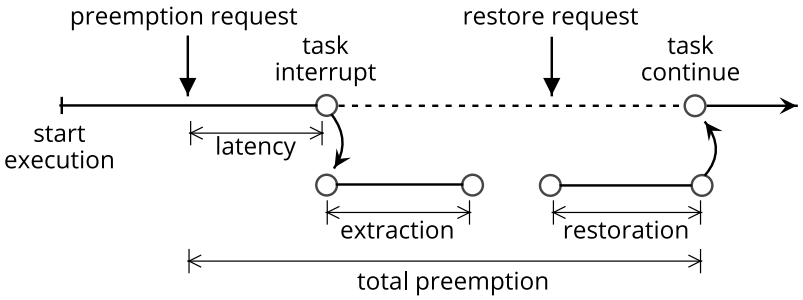


Figure 35: Hardware context switch scenario used in the experiments

5.5.1 *Evaluation Scenario*

Figure 35 presents the scenario that we followed in our experiments. During every task execution, a hardware context switch was performed to a hardware task. This scenario was repeated 1000 times for every benchmark application to obtain results with statistical confidence. The scenario began by starting the execution of a hardware task on the FPGA. At an arbitrary point in time, a preemption request was given to the task. We defined the time from the arrival of the preemption request to the beginning of context extraction as the latency in hardware context switch (preemption latency). The context and the I/O data inside the communication channels were extracted and stored in the main memory of Reconfigurable SoC platform through AXI interconnection. After the task context extraction was finished, it was restored back to the FPGA from the main memory to complete the execution. The storing time between the extraction and restoration of contexts was not considered as it was not relevant in this evaluation.

The explained scenario was used to evaluate the communication infrastructure used with every benchmark application. The FPGA configuration was done once before the scenario was executed on each application. The average time required to perform an FPGA full configuration from Linux on the processor using Processor Configuration Access Port (PCAP) driver was 0.546 seconds in ZC706 and 1.23 seconds in A5SOC due to the different configuration file size. To obtain a fair evaluation of our communication method, we did not use any specific scheduler to generate preemption requests. Instead, we developed a simple script in bash to generate preemption requests at a random time from the ARM processor. These requests were sent through AXI interconnection to the FSM in communication infrastructure.

5.5.2 *Total Execution Time*

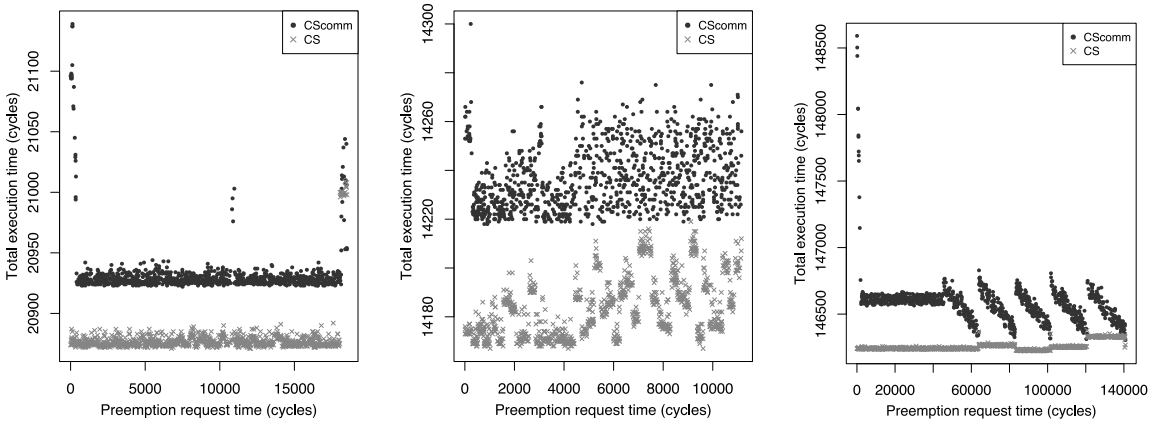
Due to communication data management during a hardware context switch in addition to the conventional task context extraction and restoration, the data footprint associated to the process was increased. First of all, we verified the impact of the con-

text switch to the total execution time. Using the prepared scenario, context switch was performed to the system which followed two protocols, CS and CSComm, and the total execution time was measured. Figure 36 presents the plots of the total execution time for each benchmark application when a hardware context switch is performed due to a given preemption request. We present only the results of the measure in ZC706 since the experiments in A5SOC provide similar results. The x-axis of the plots represents the moment when a preemption request is received by the communication infrastructure where it will be relayed to the hardware task. The maximum range in x-axis thus is the total execution time if there is no hardware context switch during task execution. The y-axis of the plots represents the total execution time including the normal execution and hardware context switch.

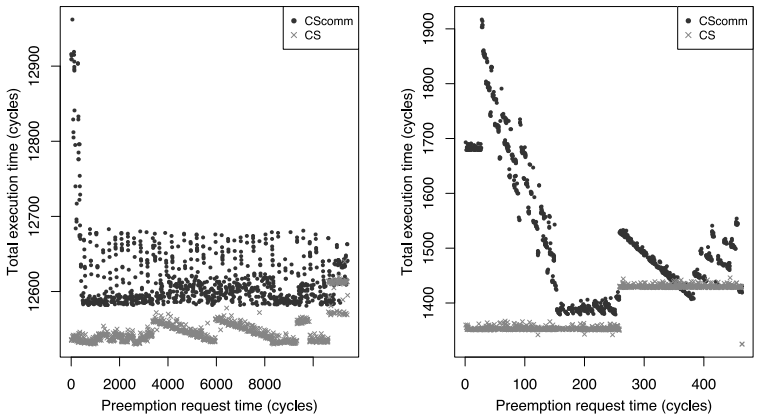
By sending preemption requests at arbitrary points in time, we were able to observe the impact of our solution in the entire spectrum. The total execution time in CS shows a small variation due to the different time required to arrive at checkpoint states. In contrast, different patterns in CSComm for each benchmark application can be observed in Figure 36. These patterns appeared due to the nature of the execution and the condition of communication channels at the moment when preemption requests were given. In the beginning of the execution, the input FIFO is usually filled with the input data. Since the CSComm manages the communication data, it increases the total execution time when the hardware context switch occurs. A longer time is required to extract and eventually restore the context and the communication data.

We can see that the amount of data inside the communication FIFOs evolves over time by observing the patterns in the plots for CS and CSComm. A shorter hardware context switch was obtained when the hardware task had consumed input data while it had not produced any output data yet. The patterns in CSComm can be clearly seen in Figure 36c, Figure 36f, and Figure 36g. In BLOWFISH, for instance, the context switch time (therefore the total execution time) with CSComm solution was lower when a preemption request was given at cycle 60000 than when it was given at cycle 40000. The total execution time of CSComm approached CS when the communication channel was empty. Less obvious patterns are seen for the system that uses CS solution. For instance, for MOTION (Figure 36f), the data in output FIFO caused the total execution time increases when preemption requests are given after cycle 1800.

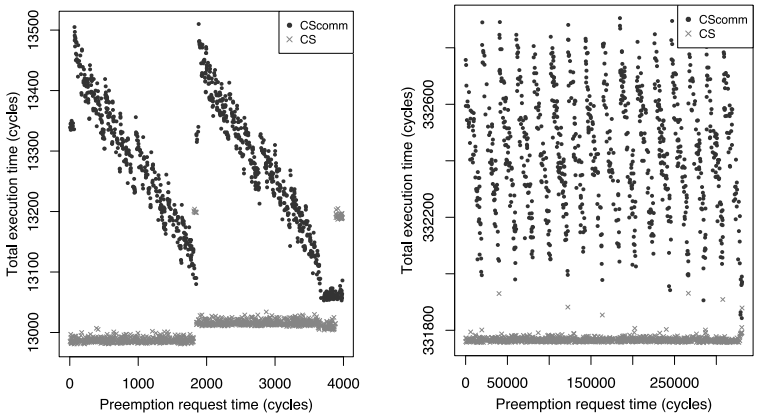
From the plots presented in Figure 36, we can also see that the total execution time in CSComm is constantly higher for most cases than in CS. The average values of total execution time with a hardware context switch are presented in Table 6 (for ZC706 platform). The results shown in the table are expected. As the CS solution does not manage the communication data in the context switch, it must delay the preemption requests if there exist ongoing communication flows. The latency shown in Figure 35 will therefore increase unpredictably, which will be explained in Section 5.5.4. However, the execution flow of the task in CS will advance while waiting



(a) ADPCM (b) AES (c) BLOWFISH



(d) GSM (e) IDCT



(f) MOTION (g) SHA

Figure 36: Total execution time with preemption requests at arbitrary points in time (ZC706)

Table 6: Comparison of average total execution time (FPGA cycles) with a hardware context switch between CS and CSComm in ZC706

App	CS	CSComm	Overhead
adpcm	20879	20933	0.26 %
aes	14184	14237	0.37 %
blowfish	146255	146583	0.22 %
gsm	12548	12619	0.57 %
idct	1388	1516	9.19 %
motion	13009	13268	1.99 %
sha	331767	332435	0.2 %

the communication flows to finish. As a result, CS will leave a shorter part to be resumed after the task is restored compared to CSComm. For this reason, the larger data footprint due to managed data in communication FIFOs has a bigger influence on the total execution time, which explains the overhead of CSComm compared to CS.

Nevertheless, the overhead in the total execution time due to our communication solution is negligible in most of the benchmark applications. The exception is particularly observed in IDCT which already has a short execution time in the first place. Performing the context extraction and restoration in CS increases more than 100% of the initial execution time (cf. Figure 36e). A higher overhead is obtained due to our communication solution in CSComm which includes I/O communication data in the process. The overall results in A5SOC, which are described in Table 7, generally agree with the results of the experiments in ZC706.

5.5.3 *Context Switch Time*

This section details the overhead of our communication solution (CSComm) to the context switch time, which is the extraction and restoration time of the hardware task context and I/O communication data which we refer to as communication-aware context (cf. Section 4.5.3). In contrast, the context in CS did not include the data in communication FIFOs. Table 8 describes the comparison of extraction and restoration of task context between the system which implemented CS solution and the one with CSComm solution. As expected, CSComm required more time in extraction and restoration for all benchmark applications. However, the table shows that the increases in the extraction time were not significant in most of the applications for CSComm. In these experiments, we took advantage of a character-

Table 7: Comparison of average total execution time (FPGA cycles) with a hardware context switch between CS and CSComm in A5SOC

App	CS	CSComm	Overhead
adpcm	21293	21341	0.23 %
aes	14768	14815	0.31 %
blowfish	147241	147625	0.26 %
gsm	12695	12757	0.48 %
idct	1480	1594	7.68 %
motion	14808	15099	1.96 %

Table 8: Comparison of average extraction/restoration time (FPGA cycles) in hardware context switch between CS and CSComm in ZC706

App	Extraction Time			Restoration Time			Total Overhead
	CS	CSComm	Overhead	CS	CSComm	Overhead	
adpcm	1151	1152	0.07 %	1174	1229	4.7 %	2.41 %
aes	1485	1486	0.06 %	1510	1562	3.45 %	1.77 %
blowfish	2400	2401	0.04 %	2429	2720	12 %	6.05 %
gsm	527	547	3.74 %	551	608	10.39 %	7.14 %
idct	417	501	20.18 %	442	539	21.94 %	21.08 %
motion	4494	4495	0.02 %	4519	4776	5.69 %	2.86 %
sha	396	714	80.33 %	422	745	76.28 %	78.24 %

istic of the scan-chain structures provided by the CP3 plugin. This characteristic is that there do not exist communication data reading and writing between checkpoint states. For this reason, we put the extraction of I/O communication data first when a preemption request is received. By the time the task arrived at a checkpoint, and the extraction of task context can be started, a certain amount of I/O data has already been extracted. For several applications, this led to an extraction overhead less than 0.1 %.

The benefit of starting the I/O data in advance can be overshadowed if a task has intensive communication and a small context size, such as for SHA. SHA had the smallest context size among the bench applications (cf.Table 5) and showed the highest overhead in context extraction. The second highest overhead is shown by IDCT with less intensity in communication. Although GSM also has a context size

Table 9: Comparison of average extraction/restoration time (FPGA cycles) in hardware context switch between CS and CSComm in A5SOC

App	Extraction Time			Restoration Time			Total Overhead
	CS	CSComm	Overhead	CS	CSComm	Overhead	
adpcm	1351	1352	0.06 %	1390	1438	3.44 %	1.78 %
aes	1770	1772	0.1 %	1809	1854	2.5 %	1.31 %
blowfish	2891	2893	0.05 %	2934	3286	12 %	6.07 %
gsm	599	613	2.4 %	632	681	7.83 %	5.19 %
idct	462	540	16.79 %	496	582	17.55 %	17.18 %
motion	5450	5450	0 %	5503	5795	5.3 %	2.66 %

lower than 1 KByte, its total overhead was only 7.14% due to its relatively low I/O data. BLOWFISH, on the other hand, has a relatively large context size and intensive communication.

In restoration, the checkpoint architecture does not bring any advantage. The I/O data and context must be restored to the channels and the task sequentially, which explains why the overhead of the restoration time is higher in CSComm for most of the applications. The restoration mechanism built in the communication infrastructures also caused longer restoration time for both CS and CSComm. Since the amount of I/O communication data and context size are stored in main memory, the FSM had to retrieve them first before restoring the data through AXI interface. This mechanism added approximately 25 FPGA cycles in restoration time for all applications. As we can see in Table 8, the restoration time of CS is longer even though the extracted and restored data size is identical. Similar results were found in A5SOC as presented in Table 9, besides the SHA application which was not implemented.

To summarize, the overhead in extraction and restoration due to the proposed solution is almost inevitable as the communication management increases data footprint. On top of that, the communication management during hardware context switch will generate a noticeable impact on a task that has small context size. Although, the worst case additional time in the extraction/restoration can be predicted from the size of the FIFOs which need to be managed. Minimizing the FIFO size for communication thus will reduce the maximum additional time in the extraction and restoration of communication-aware context.

5.5.4 *Preemption Latency*

The preemption latency, as described in Figure 35, is defined as the time between the arrival of preemption request and the beginning of context (and communication data) extraction. Whereas, the total preemption time is defined as the sum of latency and context switch time. There are two components in the preemption latency in our system. The first component is caused by the checkpoint architecture. In order to start the context extraction from a hardware task, the execution flow has to reach a checkpoint state. This latency depends on the implementation of scan-chain structures in the task. The second latency component is due to the ongoing communication flows which prevent a hardware task to be context switched. In CS, this latency is shown by the delay in context switch until there is no communication data intermediately stored in the channels. In contrast, CSComm removes this delay since it can handle the situation where the communication channels are still busy when a preemption request is received.

Table 10 describes the comparison of average preemption latency between CS and CSComm in ZC706. It is worth mentioning that our implementation of CS manipulated the input data flow to cut the delay due to the communication (cf. Figure 17). The input data flow is stopped when a preemption request is given so that no new data could arrive at the communication buffers, as proposed in [Vu+16] and presented in Section 5.3.3.2. At the output side, output data was immediately transferred to the FIFO of the software task every time they were produced until the context extraction began. This was possible due to the large size of FIFOs of the software task in the experiments. This condition is beneficial to the preemption latency of CS since, normally, the output data can only be transferred outside the FPGA if the receiver is ready (enough space in the FIFO of the successor task). Despite this, the latency of benchmark applications shown in the table are still significant, particularly for applications which are communication-intensive, e.g., BLOWFISH and SHA. For applications with less intensity in communication, the latencies were 147 cycles or more. This shows the cost of the penalty due to the communication.

With our solution in CSComm, the latency in preemption was significantly reduced to 56 cycles or less for all benchmark applications. This value reflects the initial latency caused by the checkpoint architecture since the preemption requests were not stalled anymore when the preempted task has ongoing communication flows. This result shows the evident benefit of our solution. While maintaining the communication data consistency in a preempted task, it also significantly reduces the latency in a hardware context switch operation.

With the latency and total preemption time in Table 10, we also present the ratio between them. When we compare the ratio of latency to total preemption time between CS and CSComm, we observe substantial differences between CS and CSComm. This ratio correlates to the system responsiveness to preemption requests.

Table 10: Comparison of average preemption latency (FPGA cycles) between CS and CSComm in ZC706

App	CS			CSComm		
	Latency	Total Preemption	Ratio	Latency	Total Preemption	Ratio
adpcm	147	2472	5.93 %	6	2387	0.23 %
aes	148	3143	4.71 %	17	3066	0.56 %
blowfish	20017	24847	80.56 %	56	5177	1.08 %
gsm	178	1256	14.16 %	14	1168	1.16 %
idct	180	1039	17.3 %	10	1050	0.93 %
motion	876	9889	8.86 %	7	9278	0.07 %
sha	10153	10971	92.54 %	7	1467	0.51 %

How fast hardware tasks in reconfigurable architectures react to preemption requests from the scheduler is frequently important in a system with a strict time constraint. When the latency ratio is high in the hardware context switch, the system has low responsiveness to preemption requests. CS spends more time waiting until the communication channels do not contain data anymore rather than performing the context extraction and restoration. In Table 10, the latency of CS took up to 92.54 % of the total preemption time whereas CSComm reduced the latency to 1.16 % and below for the benchmark applications. This result means that our communication management provides the high system responsiveness in a hardware context switch.

With the reduction of preemption latency in CSComm, we also obtained a shorter preemptive context switch time for most of our bench applications. This advantage was shown in most of the benchmark applications used in the experiments. However, more obvious differences between CS and CSComm regarding the total preemption time are shown in BLOWFISH and SHA. The overhead caused by extracting and restoring the I/O communication data became negligible thanks to the reduction in preemption latency. Nonetheless, the overhead in data footprint is still higher for IDCT due to its small context size and its less intensity in data exchange. Similar results when the evaluation was performed in A5SOC are presented in Table 11.

Finally, we would also like to highlight the predictability in the preemption offered by our solution. Figure 37 shows the preemption time presented in box-and-whisker plots. The box describes 50 % of the data and the whiskers represent the maximum and minimum values in our measurements. A wide range of values in the plots shows a high variation in the preemption time. Not only does our communication solution reduce the average latency in preemption, it also improves the predictability of the total preemption time for most of the benchmark applications. Again, these

Table 11: Comparison of average preemption latency (FPGA cycles) between CS and CSComm in A5SOC

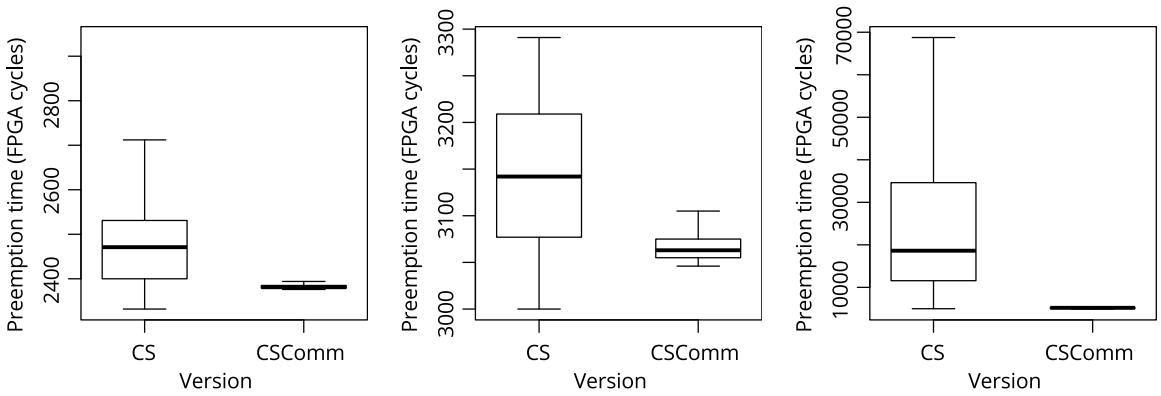
App	CS			CSComm		
	Latency	Total Preemption	Ratio	Latency	Total Preemption	Ratio
adpcm	145	2887	5.04 %	5	2795	0.19 %
aes	149	3728	4 %	18	3643	0.48 %
blowfish	19726	25551	77.2 %	54	6233	0.86 %
gsm	174	1405	12.4 %	15	1309	1.14 %
idct	175	1133	15.47 %	10	1132	0.84 %
motion	886	11838	7.48 %	7	11252	0.06 %

results are expected since our method bounds the latency due to the communication channels, at least in the worst case, before a context extraction can begin. The plots show that the systems which use CSComm have less variation in the total preemption time, except for IDCT. The negative impact in IDCT was due to the small context size of the task.

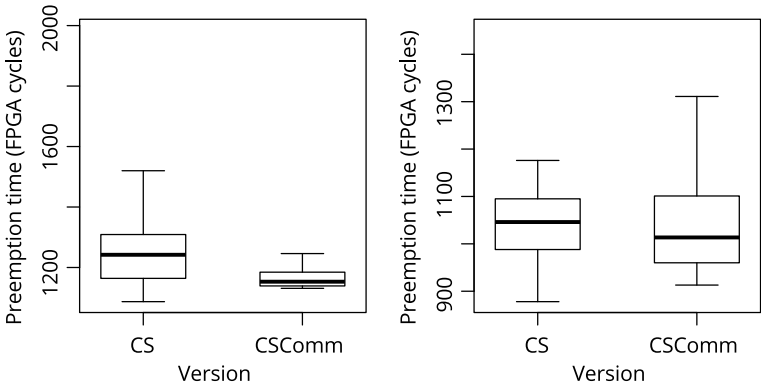
In conclusion, our solution provides the necessary high responsiveness in a hardware context switch besides preserving the communication data integrity during the process. It also improves the predictability in the context switch since the size of the FIFOs in the communication channels can be adjusted.

5.6 APPLICATION

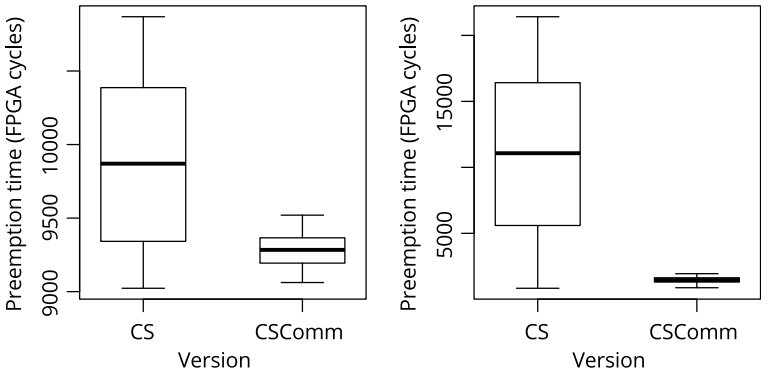
The hardware context switch ability on FPGAs which is offered by the state-of-the-art techniques and the I/O communication management which comes with it in our solution enable hardware multitasking in heterogeneous reconfigurable architectures. In addition to the performance evaluation, we developed two applications in the experiments. The first application was task migration in heterogeneous reconfigurable systems. We migrated a hardware task that ran in one of the Reconfigurable SoC platforms in the experiments, as shown in Figure 25, to another Reconfigurable SoC platform. As discussed in Section 4.5.1, communication between FPGAs in different Reconfigurable SoCs is not treated yet in this work. For this reason, we had to migrate the entire application from one Reconfigurable SoC platform to another. A task migration in this work is actually a migration of an entire application between Reconfigurable SoC platforms. Our communication solution ensured the communication data consistency at any time during the migration. The second application



(a) ADPCM (b) AES (c) BLOWFISH

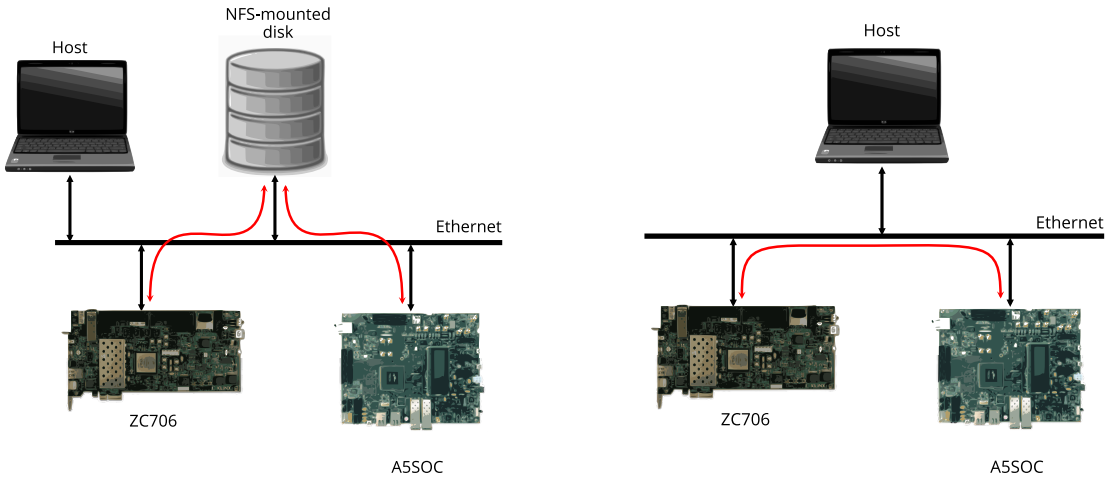


(d) GSM (e) IDCT



(f) MOTION (g) SHA

Figure 37: Hardware context switch time measured from the moment a preemption request is received until the task resumes (ZC706)



(a) Migration NFS (b) Migration SSH

Figure 38: Hardware task migration between heterogeneous reconfigurable SoCs

was built for FPGA virtualization where multiple users in independent operating systems share common hardware resources in a cloud-FPGA.

5.6.1 *Migration in Heterogeneous Reconfigurable Systems*

A hardware context switch for the purpose of migrating a task from an FPGA to another can occur due to many reasons, e.g., load distribution, fault tolerance, etc. As we base our work on design-based techniques to provide the hardware context switch ability, task migration between two different FPGAs should be absolutely feasible. We developed, using the two Reconfigurable SoC platforms, a system for prototyping task migration in heterogeneous reconfigurable systems.

Figure 38 describes the overview of two migration architectures that were built in the experiments. Both Reconfigurable SoC platforms in the architectures were connected through an Ethernet connection. The difference between the two architectures was the method to transfer the context and communication data between platforms. In Figure 38a, NFS protocol was used. To facilitate such a protocol, a disk was mounted in the network as a sharing media between the platforms. In Figure 38b, the context was transferred from a Reconfigurable SoC platform to another by using the SSH protocol.

The application was launched from a host PC. The request to migrate was given by the host PC as well. When a migration request is given to one of the platforms, a script that was created for the task (application) migration by accessing the drivers in each platform was called. Figure 39 presents the timeline in our task migration application. The migration script ran the execution flow in each Reconfigurable SoC platform as well as the transfer of context and communication data as files. Besides the drivers to control the communication infrastructure in the FPGA (cf. Section 5.4), we also used Linux drivers to read and write from the FIFOs of all

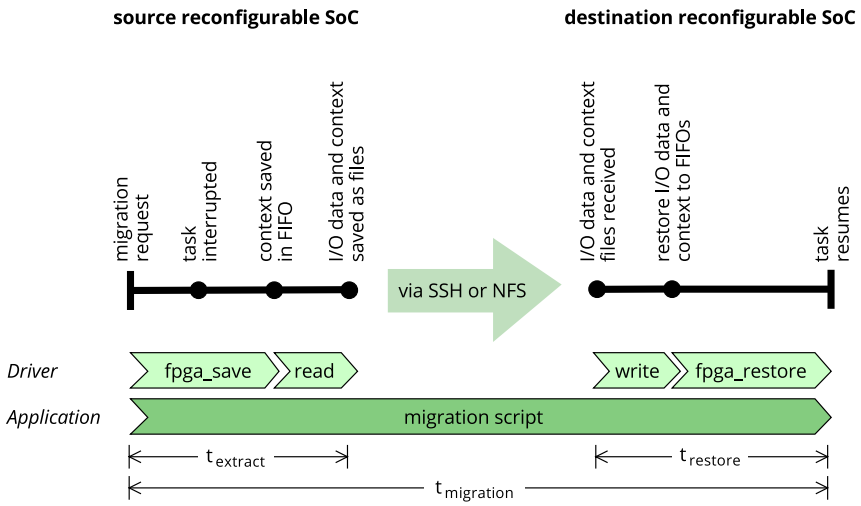


Figure 39: Task migration timeline

tasks on each platform. We consider the time from the arrival of migration request until the context is ready to be sent in the source Reconfigurable SoC as $t_{extract}$ whereas the time from the arrival of context until the task resumes in the destination Reconfigurable SoC as $t_{restore}$.

Table 12 presents the timing results in the task migration. The $t_{extract}$ took 0,03 seconds or less on average to perform the process of reading a task context and communication data as well as saving them as files. The $t_{restore}$ required roughly the same period to restore the tasks and resume the execution. Both the $t_{extract}$ and $t_{restore}$ include the preemption time in the FPGAs. The $t_{migration}$ shows the total required time as well as the cost of transferring the files from one platform to another. We can see in the table that the NFS transfer took a shorter amount of time thanks to asynchronous transfer through a NFS-mounted disk in both platforms. The SSH transfer, on the other hand, performed handshaking before doing a synchronous transfer from a sender to a receiver. The values shown in Table 12 are the average values for both migration direction. As a result, they do not include the FPGA reconfiguration time due to the difference in each platform. As a reminder, reconfiguring the FPGA in ZC706 takes 0.546 seconds while reconfiguring the FPGA in A5SOC takes 1.23 seconds.

Throughout this section, we showed a task migration prototyping in heterogeneous reconfigurable systems. Although further development is still necessary to improve the performance of the method, we have shown two advantages of our work. First, we have shown that the proposed mechanism is functional in heterogeneous reconfigurable systems and is capable to preserve the communication data integrity during the migration. Secondly, we have shown the genericity of our solution by implementing the same mechanism in different heterogeneous platforms. Although some differences due to memory mapping must be considered when implementing the software (cf. Section 5.4), the hardware architectures are identical.

Table 12: Task migration time between A5SOC and ZC706

	t_{extract} (s)	t_{restore} (s)	$t_{\text{migration}}$ (s)	
			NFS	SSH
ADPCM	0.026	0.021	0.577	0.759
AES	0.025	0.021	0.577	0.760
BLOWFISH	0.030	0.027	0.606	0.969
GSM	0.025	0.020	0.572	0.757
IDCT	0.024	0.020	0.574	0.762
MOTION	0.028	0.027	0.606	0.972

5.6.2 *Hypervisor-based System for FPGA Virtualization (Cloud-FPGA)*

For years, FPGAs have always been used as hardware accelerators. By moving the execution of the performance critical or heavy parts from a CPU to an FPGA, a typical 10×–100× speed-up can be obtained. Recently, there are growing interests in using FPGAs in cloud infrastructures (cloud-FPGA). FPGAs offer better security to data and computation compared to software solutions which is a fundamental problem in cloud computing [EV12]. The authors in [Che+14] explained the FPGA integration in cloud is non-trivial due to the capability of FPGAs in resource abstraction and sharing. Due to similar reasons, the work in [FVS15] proposed FPGA accelerators for cloud server. The latest implementations in datacenter application for Baidu [Ouy+14] and Microsoft Bing [Put+15] search services confirm the potential which can be offered by FPGAs.

In recent works, the virtualized environments have been used in popular heterogeneous CPU-FPGA architectures [KSH05]. Hypervisors, which is a form of paravirtualization in bare-metal level, were proposed both in X86 and ARM architectures. They offer several advantages in the cloud implementation, e.g., higher system security due to the user isolation, resource efficiency from the hardware virtualization, and thus power efficiency. Over the years, hypervisors that manage the CPU and FPGA resources have been proposed for various architectures. Xen hypervisor which runs on X86 processors was integrated with an FPGA [WBP13]. The processor and the FPGA communicate to each other via a PCIe connection. A similar architecture was also proposed in [KS15] where the FPGA resources were divided into partial regions. A solution using microkernel-based hypervisor for embedded platform were proposed in [Jai+14].

A hardware context switch support combined with a hypervisor-based system will provide an illusion of a higher capacity system. Most of the existing works consider the FPGA as a dedicated resource in fixed functions. Dividing FPGA resources

into partial regions enables run-time task allocation in the system. However, it also reduces the available resource for more complex applications. With support in hardware context switch, several users can share common FPGA resources for bigger applications. Such features can also be used for consolidating the cloud resources for higher energy efficiency.

We developed a hypervisor-based system on a heterogeneous CPU-FPGA architecture with the hardware context switch support. This architecture was built on the Xilinx Zynq Ultrascale+ MPSoC ZCU102 Evaluation Kit. This platform integrates a quad-core ARM Cortex A53 processors, dual-core Cortex-R5 real-time processors, a Mali-400 MP2 GPU, and programmable logic. The ZCU102 platform was chosen due to its compatibility with one of available hypervisors in the market, Xen.

Guest operating systems run as domains (Dom) in Xen hypervisor where it is responsible in their memory management and CPU scheduling of all virtual machines. One of the domains has the privilege to access the hardware directly (Dom0). The Dom0 is responsible to launch and manage the other domains (DomU) in the system. To access devices which are shared between domains, DomUs must communicate with Dom0. This is done by a two-part driver: frontend and backend. The drivers for interdomain communication in Xen hypervisor support Grant table and event channel, and ring buffer for data transfers between domains.

Figure 40 describes the overview of a Xen-based system built on Zynq Ultrascale+ (ZCU102). All the users in DomU may access the FPGA in the system through Dom0 without users knowing the detail. When a DomU is accessing the FPGA resource, the commands are forwarded from its frontend driver to the FPGA driver by backend driver in Dom0. We developed a scheduler in Dom0 that allocates the access from each DomU in the system and follows round-robin scheduling policy.

The illustration of task allocation for different DomU (users) in the FPGA is shown in Figure 41. Xen hypervisor performs the scheduling for each domain (operating system) in the system. In the figure, every domain can run for 100 μs in the CPU before being switched. Meanwhile, the task scheduling on the FPGA is managed by Dom0. Another width of execution window can be set to every task that runs on the FPGA, for instance 1 ms in the experiments. Due to the hardware task scheduling on the FPGA, Dom0 triggers the hardware context switch operation to switch between tasks from User 1 and User 2. The entire hardware task management is completely invisible to the users.

Using the hypervisor-based system, we show that our context switch solution can be used to virtualize the FPGA. The communication management that we propose preserves the communication data integrity in the process and provides a high responsiveness at the same time which is necessary in a system with tight time constraints. The developed system in this work only verified the functionality of the design. In future works, the system can be improved to implement real-time streaming applications and to obtain its quantitative characteristics, e.g., energy consumption, performance, etc.

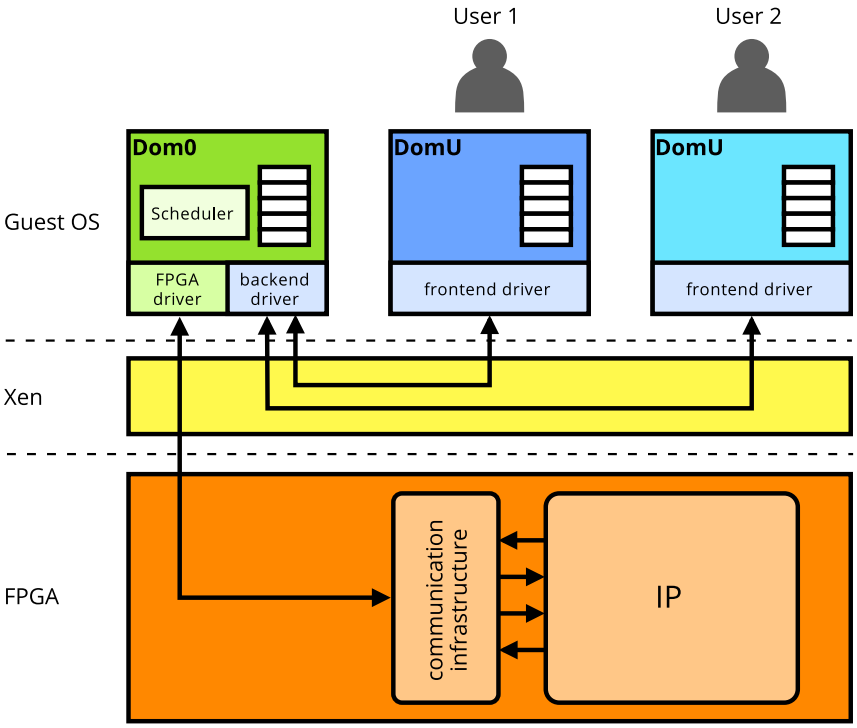


Figure 40: Overview of Xen-based system on Zynq Ultrascale+

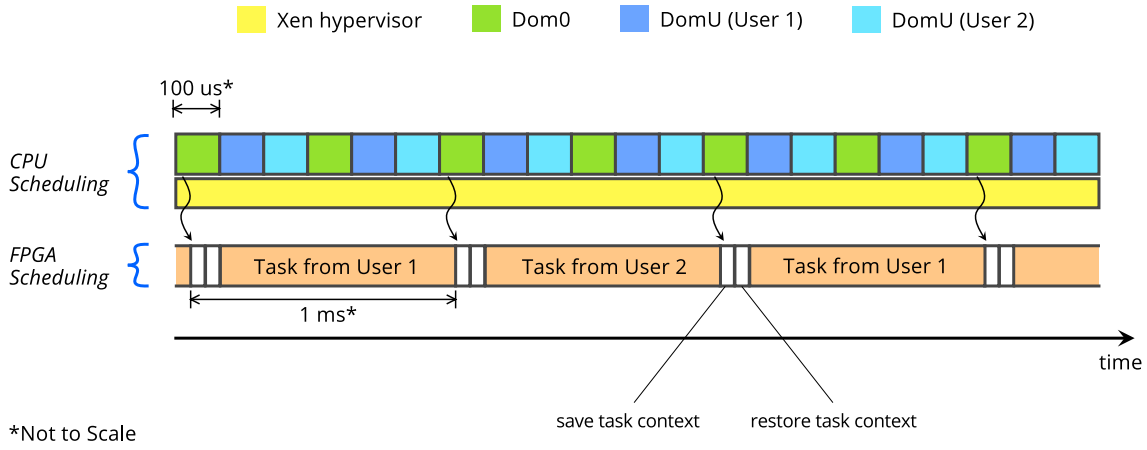


Figure 41: Task allocation (scheduling) on FPGA in the hypervisor-based environment

5.7 CONCLUSION

In this chapter we present the experiments to evaluate the performance and overhead of our communication solution as well as some applications in heterogeneous reconfigurable systems. Two popular commercial Reconfigurable SoCs from different vendors were used in the implementation: ZC706 and A5SOC. They are used in the experiments to demonstrate the genericity of our solution and the ability of our system to serve in heterogeneous reconfigurable systems. In the hardware part, we used an existing HLS tool AUGH (and its CP3 plugin) which allows the instrumentation of hardware tasks with a context extraction ability. A communication infrastructure which implements our solution is added to connect each hardware task to the rest of the system. We show, in this chapter, the integration of such infrastructure and hardware task, and the software development to manage the communication link between tasks.

The evaluations of the solution were performed on each Reconfigurable SoC platform. A scenario where a context switch was performed once to a running hardware task in the FPGA was devised. To the total execution time, our solution added a modest overhead in the benchmark applications. This was due to managing the I/O communication data stored in the FIFOs in addition to the context of a task in preemption. The overhead in the data footprint was detailed in the results of extraction and restoration time. However, the responsiveness of the system to preemption requests is significantly improved. Without any communication management, context switch must be prevented or delayed if the preempted task still has ongoing communication flows. Our solution guarantees the communication data integrity that enables a context switch operation to be started immediately after the preemption request is received. For the overall preemption process, the communication management that we propose also improved its predictability.

In this work, we present an application of hardware context switch to prototype dynamic task migration in heterogeneous reconfigurable systems. Hardware tasks can be migrated between two FPGAs in different Reconfigurable SoC platforms. Although, the prototype presented in this work required a migration of the entire application including the tasks in the CPU due to the limitation in our communication solution. Another application that we developed is to virtualize the FPGA resources in the cloud using hypervisor-based systems. In such systems, FPGA resources are shared among the users which have access to an independent guest OS. The hardware context switch support is therefore necessary to provide this multitasking feature. Our solution ensures the communication consistency when the hardware context switch occurs.

6

CONCLUSION AND FUTURE WORKS

THIS CHAPTER CONCLUDES this work which presents our contributions in the communication management and consistency preservation during a hardware context switch operation in heterogeneous reconfigurable systems. The possible future works that can extend the current development are also described in this chapter. This work is normally expandable either in the application or optimization/enhancement of the communication management in a hardware context switch.

The first proposition in the future works is to place hardware tasks that have the context switch ability in partial regions to optimize the FPGA resource utilization. We measured the performance of our solution in the experiments by using an FPGA with only one hardware task in the entire FPGA. However, the communication management presented in this work should be applicable for a system that has either multiple FPGAs or partial reconfigurable regions in an FPGA. The next proposition is to expand the compatibility of our communication solution to heterogeneous CPU-FPGA systems for large-scale distributed implementation. The scalability of the solution must be improved in order to adapt to such systems. The third proposition continues the application of hardware context switch in FPGA virtualization for hypervisor-based CPU-FPGA system, as presented in Section 5.6.2.

Contents

6.1 CONCLUSION

In recent years, FPGA devices have shown their capability to accelerate heavy computation and provide high performance to energy ratio. In order to fully exploit the potential of FPGA devices as hardware accelerators, multitasking support similar to the one in multiprocessors architectures is necessary. Executing, interrupting, migrating, or resuming hardware tasks on FPGAs from specific points in execution is possible if a hardware context switch support is provided. Enabling hardware context switch in reconfigurable devices has been an important research topic for decades, and many solutions have been proposed for this specific objective.

In this work, we focus on hardware context switch techniques that are appropriate for preemptive scheduling. The literature review has shown that design-based techniques which instrument hardware tasks on FPGAs with scan-chain structures for context extraction can offer a hardware context switch ability which is powerful and independent to the FPGA architecture used. The state-of-the-art hardware context switch solution presents a high-performance, reduced overhead, and low effort scan-chain instrumentation on hardware tasks that takes advantage of a high-level synthesis circuit generation. And yet, little attention has been paid to manage the consistency in communication when a hardware task is part of a communication network and needs to be context switched. Like in many other solutions in the literature, the questions such as data integrity, continuity, and performance in communication are left open.

This work deals with the communication data integrity that prevents a hardware task in an FPGA for being context switched while it has ongoing communication flows. Hence, the existing solutions so far proposed a hardware context switch that works under very strict constraints, i.e., absence of communication data in the channels or end of communication flows. These conditions cause a performance drop and unpredictability in the process, or the inability to context switch hardware tasks at the intended time.

To overcome the challenges in communication, we propose a context switch protocol that guarantees the communication consistency between tasks while eliminating the constraints in the existing solutions. In any case, the I/O communication data in the channels are to be managed during a context switch. We propose to extract the communication data in the FIFOs associated to the preempted task together with its context. Before the context and communication data of a preempted task are extracted, the communication link to the rest of the system has to be safely disconnected. Later, it must be reconnected to resume the interrupted execution and communication flows. Since the idea is simple and straightforward, the solution is generic and adaptable to various communication topologies as well as FPGA architectures.

A CPU-FPGA architecture was targeted in the implementation of our communication solution in the hardware context switch operation. The method is inte-

grated in a communication infrastructure that interfaces each hardware task with a context switch ability to the rest of the system. The communication flow and the extraction/restoration of task context as well as I/O communication data in the communication infrastructure are controlled by an FSM. The context and communication data are managed together as a communication-aware context in our solution. Meanwhile, the communication link is managed by a task manager, which can be a program or module, in one of the CPUs in the system. The task manager keeps the information of the communication between tasks in order to ensure the continuity in communication after a context switch.

Experiments were done to validate our method as well as to evaluate its performance and overhead in a real system. Two Reconfigurable SoC platforms with a FPGA from a different family and vendor were used. We implemented a system that followed our communication solution and a system which used the existing solution for comparison purposes. As expected from the additional communication data management in a context switch operation, our solution modestly increases the time required to extract and restore the context, thus the total execution time. According to the size of the communication FIFOs, the overhead in the context extraction and restoration may vary. Despite the overhead in the context extraction and restoration time, we had successfully shown that our communication significantly reduced the latency in the preemption. That means the responsiveness of the system toward preemption requests increases. Consequently, the total preemption time decreases and is more predictable in the system with our solution.

Nevertheless, the proposed solution has still some limitations which are not yet addressed in this work. Our communication management requires all the CPUs and FPGAs in the system to be connected in the same interconnect. To increase the scalability, a NoC-type communication topology is necessary. Otherwise, the number of CPUs and FPGAs that can be added to the system is limited. Without NoC topology, multiple interconnects need to be set in a hierarchy which requires further development. Some applications that use our solution are also possible in a system which contains partial reconfigurable regions or use hypervisor to manage the virtualization of FPGAs. The topics to improve our existing method and development will be discussed in Section 6.2.

6.2 FUTURE WORKS

6.2.1 *Hardware Context Switch in a System with Dynamic Partial Reconfiguration*

Due to the growing capacity of FPGAs, the resources in an FPGA can be divided into partial reconfigurable regions to optimize the resource utilization. Dynamic Partial Reconfiguration (DPR) has been gaining popularity recently as it provides not only dynamic placement of tasks [WP02] but also reduced FPGA configuration time [DML12]. Thanks to DPR technology, each task can be processed, interrupted,

or reconfigured separately without affecting the others. Due to this feature, some works claim to offer a non-preemptive context switch by partially reconfiguring FPGAs [Gua+08].

Integrating our solution in a system with DPR support will enable preemptive scheduling in the hardware context switch. The protocol which is proposed in this work is perfectly adapted to hardware tasks configured in partial reconfigurable regions. Each partial region in the FPGA contains a hardware task and is managed separately. Since we base our context switch solution on design-based technique, we do not require homogeneity in each partial regions, which reduces the constraints in the implementation. Each hardware task in the system is connected to a main interconnect through a communication infrastructure.

Nevertheless, there are other challenges that need to be resolved if we want to integrate a context switch method into a system with DPR support. Task relocation and fragmentation in an FPGA are some of the challenges. The potential to provide a faster communication mechanism between hardware tasks as they are located in the same FPGA must also be studied. Despite these challenges, our solution itself should be applicable in such systems.

6.2.2 *Dynamic Task Migration in Large-Scale Distributed CPU-FPGA Systems*

The communication solution in this work is developed for the system presented in Figure 19 where all the components share the same interconnect. However, it poses a limitation in the number of CPUs and FPGAs that can be added to the system. A NoC-type communication topology can be used to increase the scale of the system, which is adapted by our solution, but it is still limited to the network capacity. In other words, the current targeted system lacks scalability. Currently, there exist no interest in enabling a hardware context switch and task migration in large-scale heterogeneous architectures.

When the scale of a heterogeneous CPU-FPGA is increased for high complexity applications, further developments are necessary to enable hardware context switch and maintain the communication consistency in the system. In a large-scale CPU-FPGA system that handles complex applications, for instance the Catapult project proposed by Microsoft, many distributed servers are integrated into a common system as presented in Figure 42. Each server consists of a CPU and an FPGA device. These servers are connected by two links: PCIe and Gigabit Ethernet links. A similar architecture can also be built using Reconfigurable SoC plaforms.

The hardware context switch support in such architecture can be added if the tasks on FPGAs do not communicate with other tasks. That being so, a task migration is possible from a server to another. However, if the hardware tasks communicate with other tasks, migrating them will not be easy. The reason why our solution cannot be directly adapted to such a system is because it requires encapsulation in the communication. The connection inside each server and between servers have different

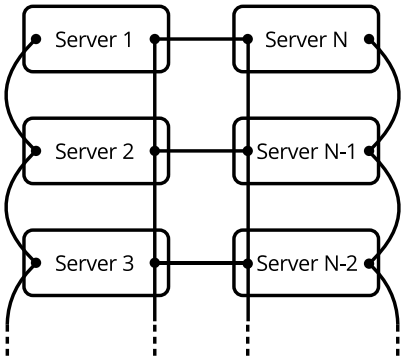


Figure 42: Abstract view of a large-scale CPU-FPGA architecture [Put+15]. Each server consists of a CPU and an FPGA.

levels. To provide the scalable solution that can satisfy a large-scale distributed system, the communication management in the context switch must consider the encapsulation for different levels of communication.

6.2.3 *Hardware Context Switch Support for Energy Efficient Cloud-FPGA*

FPGA devices are graining traction in cloud infrastructures due to the performance and efficiency offered. The objective is to virtualize FPGA resources and provide them to users in various cloud applications. In cloud architectures, accessing FPGA resources must be indirect to avoid malicious attack. For this reason, virtual machine managers, such as hypervisors are necessary to isolate the hardware from direct access as well as to manage the system transparently from the users. In order to obtain even higher performance and flexibility in the system, hardware context switch support is required to provide multitasking in FPGAs.

In this work, we presented an integration of the hardware context switch support in a Xen-based system. A common FPGA resource was shared between several users in guest OSes (DomUs). Since the users did not have the privilege to access the FPGA in the system directly, the scheduling and task allocation were performed by Dom0. A simple scheduler was implemented in Dom0 to give turns of FPGA use to each user.

Now that we have shown the possibility of hardware context switching tasks, further developments are required to move the work in a new direction. We target an energy-efficient cloud architecture which takes advantage of virtualized FPGA resources. Since the scale of cloud infrastructures has been significantly increasing due to the growing interests for remote applications, the cost and energy consumption in the cloud architecture is becoming an important concern. The direction of this work should focus on providing at least the same performance as the existing architectures that use microprocessors, but with a higher energy efficiency. The challenge lies in offering an accepted performance for as many users as possible. In order to

do that, a reliable scheduling and task allocation strategies are necessary which may include the decision to context switch based on priority, fairness, execution window, etc.

For the moment, the impact of the scheduling and task allocation strategies to the overall system performance is still unknown and needs to be evaluated. The relationship between the performance and the energy consumed needs to be studied in order to decide the maximum number of hardware tasks that can occupy the same FPGA in order to stay cost-effective. Regarding the context switch for task migration purposes, the management of hardware tasks from the hypervisor may offer certain advantages. We believe that these questions need to be resolved in the future works in cloud-FPGA architectures.

7

RÉSUMÉ

Ce chapitre en français synthétise le travail qui est décrit en anglais dans les chapitres de ce mémoire de thèse. Le chapitre résumé commence par une introduction qui explique le contexte du travail. Ensuite, les problèmes ciblés sont présentés. L'état de l'art décrit des travaux précédents qui concernent le même sujet. La solution proposée et son implémentation sont détaillées dans la partie méthodologie. Nous montrons par la suite les résultats des expérimentations pour évaluer notre solution. Nous finissons par la conclusion et les perspectives de ce travail de thèse.

Contents

7.1 INTRODUCTION

Au 21ème siècle, les microprocesseurs sont trouvés dans de nombreux dispositifs électroniques, de l'ordinateur de bureau aux serveurs hautes performances. Les microprocesseurs dans un système informatique fonctionnent en coopération avec les autres composants tels que des éléments mémoires et des coprocesseurs qui les aident à exécuter les calculs spécifiques. Ces coprocesseurs sont généralement des processeurs à faible consommation d'énergie, des processeurs graphiques (GPU), des circuits intégrés (ASIC), ou des processeurs reconfigurables de type FPGA.

L'utilisation des coprocesseurs pour accélérer les calculs dans les architectures des ordinateurs, connue sous le nom d'accélération matérielle, suscite de nos jours beaucoup d'intérêts. Un accélérateur matériel est capable d'exécuter des fonctions ou algorithmes complexes avec un meilleur rapport de performance/consommation énergétique qu'un microprocesseur (CPU) classique. Parmi les accélérateurs matériels existants, le FPGA offre un bon rapport entre la performance et la consommation énergétique avec une flexibilité pour changer d'application à la volée.

Au cours de la dernière décennie, l'utilisation de nuage informatique (cloud computing) pour fournir des services au marché industriel devient de plus en plus important. Les calculs sont exécutés en tant qu'instance sur des ressources de calcul virtualisées, dans ce cas des CPU et des FPGAs. Par conséquent, la capacité de gérer les instances dans cet environnement virtualisé est nécessaire. Elle est indispensable notamment pour distribuer les charges de calcul, gérer les ressources en panne et mettre à niveau l'infrastructure. La gestion des ressources virtuelles peut être offerte par le support de changement de contexte à la fois sur les CPU et sur les FPGAs.

En informatique, le changement de contexte est un phénomène bien connu et parfaitement maitrisé, il consiste à partager le temps d'utilisation du microprocesseur entre les applications en cours d'exécution. Le système d'exploitation gère l'arrêt d'une application, la mémorisation des informations nécessaires à la poursuite de son exécution avant l'exécution de l'application suivante. Le changement de contexte matériel sur FPGA suit le même principe; il permet l'interruption d'une fonctionnalité (ou d'une tâche) sur FPGA puis mémorise les informations nécessaires à la reprise de la fonctionnalité plus tard.

Contrairement aux tâches sur CPU qui sont stockées en tant que programme sur la mémoire du système, les tâches sur FPGA sont implémentées en matériel. Bien que la performance en exécution sur matériel soit meilleure, le changement de contexte matériel sur un FPGA doit considérer une sauvegarde des contenus de registres et mémoires sur FPGA ainsi qu'une gestion des communications de la tâche matérielle. Dans ce travail, nous avons étudié l'aspect communication durant un changement de contexte matériel. Une méthode de communication qui permet de maintenir la cohérence de communication lorsqu'une tâche sur un FPGA est commutée est nécessaire. Dans la section suivante, nous présentons les problèmes qui sont liés à la communication durant un changement de contexte matériel.

7.2 PROBLÉMATIQUE

7.2.1 *Le système hétérogène reconfigurable*

Le changement de contexte matériel est un sujet important dans la recherche au cours des dix années. Afin de fournir la solution qui permet d'interrompre une tâche en cours d'exécution, l'aspect hétérogénéité est souvent considéré dans des nombreux travaux précédents. La définition de l'hétérogénéité dans ces travaux est malheureusement différente. Cette section présente le type d'hétérogénéité considérée dans notre travail.

Un système reconfigurable est normalement constitué au moins d'un FPGA et d'un CPU. Le FPGA agit comme l'accélérateur des calculs lorsque le CPU exécute les applications logicielles. Cette intégration du CPU/FPGA va aboutir à un système qui est hétérogène. L'hétérogénéité se trouve aussi à l'intérieur de l'architecture de FPGA du système. Aujourd'hui, un FPGA intègre différents éléments de calcul et élements mémoires, par exemple des processeurs de signal numérique (DSP), des blocs mémoires (BRAM), etc. Ces éléments augmentent la performance de calcul fournie par un FPGA comme ils permettent d'implémenter les tâches plus grosses et plus complexes. Cependant, l'implémentation des tâches sur un FPGA devient de plus en plus spécifique aux outils fournis par les fabricants des FPGAs.

L'optimisation apporté par des fabricants pour viser les différentes gammes d'utilisations cause l'hétérogénéité entre les FPGAs de différentes familles, technologies, architectures, etc. En raison de l'évolution de l'infrastructure pour une modernisation ou pour remplacer les ressources en panne, les FPGAs utilisés dans le système peuvent être différents. Pour ces raisons, le support du changement de contexte matériel doit considérer cette hétérogénéité entre les FPGAs différents. Pour le reste de ce chapitre, le terme «hétérogénéité» sera utilisé pour décrire la différence d'architecture des FPGAs dans un système reconfigurable.

7.2.2 *Ordonnancement préemptif*

Similaire au changement de contexte logiciel, le changement de contexte matériel est effectué suite à un déclenchement ou une demande d'un ordonnanceur. Dans un système reconfigurable, l'ordonnanceur peut être implémenté sur un CPU ou un FPGA dédié pour la gestion de tâche matérielle. Il existe deux catégories d'ordonnancement selon la prise de décision à la gestion des tâches dans le système [Gua+08], non-préemptif et préemptif.

L'ordonnancement non-préemptif ou coopératif est un ordonnancement dans lequel une tâche volontairement donne le contrôle de ressources à une autre application. Au contraire, l'ordonnancement préemptif gère une demande d'interruption à une tâche sans coopération afin d'exploiter les ressources utilisées. D'un côté, l'ordonnancement préemptif offre une grande flexibilité à la gestion de tâche car une

interruption peut se faire à tout moment durant l'exécution des tâches. D'un autre côté, les efforts sont nécessaires pour gérer les flux d'exécution et de communication qui sont interrompus, ce qui doivent être considérés dans la méthode du changement de contexte.

7.2.3 *Synthèse de la problématique*

Dans ce travail, la discussion s'est bornée au changement de contexte matériel suite à une demande de préemption. Pour cette raison, un changement de contexte matériel peut se faire à des instants arbitraires, y compris lorsqu'il y a un flux de communication en cours. La demande de préemption va interrompre l'exécution en cours d'une tâche sur un FPGA et en même temps son flux de communication. Les questions de synthèse auxquelles nous allons tenter de répondre sont les suivantes :

- Comment le contexte d'une tâche matérielle doit être géré ?
- Comment doit-on préserver l'intégrité des données de communications ?
- Comment satisfaire les contraintes de performance du changement de contexte matériel ?
- Que faire pour assurer la continuité de communication entre les tâches dans un système reconfigurable ?

7.3 ÉTAT DE L'ART

7.3.1 *Changement de contexte matériel*

Au fil des années, des techniques d'extraction de contexte sur un FPGA ont été proposées. Elles sont principalement classées en deux grandes familles [Joz+10]. La première classe rassemble les techniques d'extraction de configuration de tâches et des contenus de registres et mémoires à travers le port de reconfiguration des FPGA. Le changement de contexte est donc limité à certaines familles et fournisseurs de FPGA. Cette méthode d'extraction de contexte est connue sous le nom de méthode de relecture. Les travaux précédents [Lev+00; SLM00] démontrent que cette technique permet de sauvegarder les contenus de tous les registres et les mémoires sur un FPGA à un moment précis. En conséquence, la gestion du contexte d'une tâche et des données de communication est fournie par la méthode de relecture. Cependant, cette méthode génère une taille de contexte importante en raison de l'extraction de la configuration en même temps que le sauvegarde du contexte. D'ailleurs, la méthode de relecture est moins adaptée pour interrompre une tâche sur un FPGA et la continuer sur un autre FPGA et inadaptée si les FPGA sont hétérogènes.

La deuxième méthode rassemble les techniques d'ajout d'interfaces aux fonctionnalités sur FPGA, qui permet un changement de contexte indépendamment de la

technologie du FPGA [Whe+01], connue sur le nom de la méthode embarquée. Une structure supplémentaire (scan-chain) qui connecte les registres et les mémoires sous forme chainée est ajoutée dans la conception de tâches sur FPGA. La méthode embarquée permet d'extraire seulement les données des tâches à l'état où l'exécution est interrompue, ce qui donne une meilleure efficacité au changement de contexte matériel et l'indépendance à la technologie FPGA utilisée. Cependant, la structure scan-chain introduit un surcoût en taille et performance de circuit qui peut être important à la conception de tâches matérielles. La notion de points de sauvegarde "checkpoints" est souvent intégrée à la méthode embarquée pour diminuer le surcoût matériel de la conception [KHT07; Vu+16; BMR16]. Le travail récemment mené par [BMR16] propose un outil de synthèse de haut niveau pour ajouter automatiquement la structure scan-chain aux tâches matérielle. Ceci diminue fortement l'effort nécessaire par les utilisateurs, qui rend la solution avec la méthode embarquée très intéressante et prometteuse pour offrir le support du changement de contexte matériel.

7.3.2 *Gestion de communication*

La capacité d'interrompre une tâche matérielle en cours de son exécution et d'extraire son contexte est primordiale afin de fournir un support au changement de contexte matériel. La gestion (sauvegarde, stockage, et restauration) du contexte et des données de communication est également importante. Dans des nombreux travaux existants [VPI05; LP08; Nar+11; Joz+13], la gestion de communication est effectuée par un système d'exploitation sur le CPU. Le CPU est alors responsable de la synchronisation entre tâches et de la communication entre logiciel et matériel. Cependant, ces travaux ne prennent pas en compte le changement de contexte matériel dans un système reconfigurable. Le travail dans [LP08] a été complété par [HTK15] avec l'intégration de méthode de relecture pour l'extraction de contexte du FPGA.

Pour satisfaire l'aspect hétérogénéité d'un support de changement de contexte, la méthode embarquée est plus prometteuse par rapport à la méthode de relecture. En revanche, la méthode embarquée ne gère pas les données de communication à l'extérieur de tâche matérielle. Un changement de contexte matériel est donc possible seulement quand il n'y a pas de flux de communication entre tâches. Ceci est montré par le travail de [Vu+16] qui propose une solution d'étranglement sur les canaux de communication afin de vider les tampons des données. Lorsqu'une demande de préemption est donnée à une tâche matérielle sur un FPGA, les flux entrants de communication sont bloqués et le changement de contexte matériel est retardé jusqu'à ce que la tâche ait consommé toutes les données dans les tampons de communication.

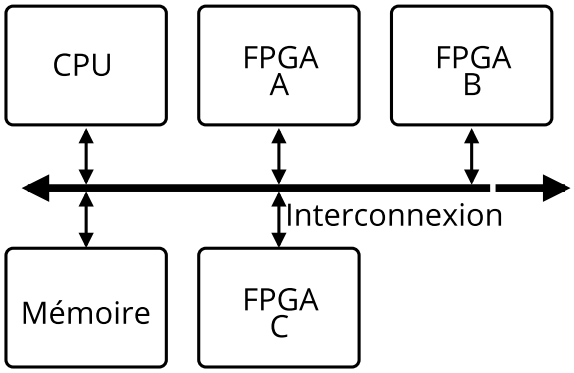


Figure 43: Une illustration d'un système reconfigurable avec multiple FPGAs

7.4 MÉTHODOLOGIE

Dans cette section, nous allons présenter brièvement l'hypothèse de travail incluant le système visé par notre solution. Ensuite, nous proposons la solution afin de résoudre les problèmes de communication durant un changement de contexte matériel sur un FPGA avec le modèle de réseau de communication de Kahn (KPN). Finalement, nous décrivons un protocole qui permet de faire un changement de contexte matériel en prenant en compte la consistence de communication.

7.4.1 *Hypothèse de travail*

Nous avons visé un système reconfigurable qui est constitué d'au moins un CPU, un FPGA et une mémoire partagée. Les tâches peuvent être exécutées à la fois sur le FPGA et le CPU, selon les besoins. La gestion des tâches est effectuée par le CPU. Les tâches matérielles sur le FPGA sont implémentées sous la forme d'IPs. Ces IPs peuvent provenir d'ailleurs, ce qui signifie que nous n'avons pas la maîtrise de leur conception. De multiples FPGAs sont possibles dans le système visé par notre travail et ils sont tous connectés au même réseau d'interconnexion avec le(s) CPU(s) comme illustré dans la Figure 43.

Nous basons notre travail sur la méthode embarquée pour les raisons détaillées dans la Section 7.3.1. Nous supposons que les IPs utilisées ont la capacité du changement de contexte matériel fournie par cette méthode. Le travail présenté dans [BMR16] propose un mécanisme automatisé d'insertion de structure scan-chain en conception des IPs dans une synthèse de haut niveau (HLS). Cette automatisation de la méthode embarquée est intégrée dans AUGH, un outil HLS libre proposé dans [PMR14]. Grâce au flux de synthèse de haut niveau, le mécanisme d'insertion de scan-chain et la sélection des checkpoints dans la méthode embarquée peuvent être optimisés.

Bien que la méthode embarquée offre les avantages sur la performance et l'hétérogénéité, elle ne gère pas les données de communication qui sont en transit

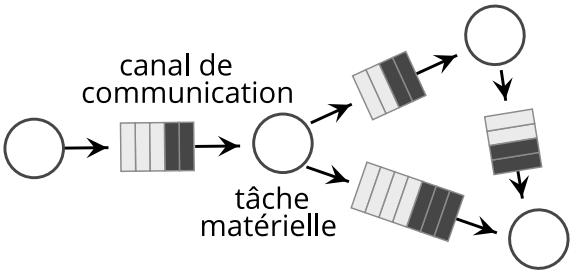


Figure 44: Exemple d'une représentation des tâches dans un système reconfigurable en KPN

entre deux tâches en communication. Par conséquent, un changement de contexte doit se faire lorsque les canaux de communication sont vides naturellement ou par un étranglement [Vu+16]. La section suivante présente notre solution pour résoudre les problèmes de communication durant un changement de contexte matériel.

7.4.2 *Solution proposée*

Nous avons modélisé la communication entre tâches en utilisant le réseau de communication de Kahn (KPN) [Kah74]. Dans KPN, une tâche est considérée comme un processus autonome qui travaille simultanément avec les autres processus dans le réseau et communique avec eux à travers des tampons FIFO de taille infinie. Cette condition aboutit à une lecture bloquante et une écriture non-bloquante en théorie. En pratique, les opérations de lecture et d'écriture sont bloquantes en raison de la quantité limitée de mémoire [GB03].

Le choix du modèle KPN est motivé par le fait qu'il est déterministe. La communication dépend de l'existence des données et l'ordonnancement n'est pas nécessaire tant que la sémantique de communication bloquante est respectée. De ce fait, la connexion entre tâches est interruptible et les autres tâches peuvent continuer leur exécution jusqu'à la saturation des canaux de communication (vide ou plein).

Nous proposons l'intégration des données de communication à l'extraction de contexte des tâches matérielles. Lorsqu'un changement de contexte matériel est effectué, le contexte et toutes les données de communication associées à la tâche matérielle sont extraits du FPGA vers la mémoire partagée et vice-versa dans le cas de la restauration. Avant que ces données soient extraites, un gestionnaire qui peut être à la fois une tâche matérielle et logicielle gère la déconnexion de communication de la tâche commutée. Cette solution permet de maintenir l'intégrité des données de communication. Grâce à cette approche, le changement de contexte matériel n'a pas besoin d'attendre que les canaux de communication soient vides. Ceci va améliorer la performance et satisfaire les contraintes de temps.

7.4.3 *Protocole de changement de contexte matériel*

La solution proposée est intégrée dans un protocole de changement de contexte matériel qui prend en compte la communication en cours d'une tâche sur le FPGA du système. La Figure 45 présente ce protocole. Premièrement, le flux de communication doit être interrompu lorsqu'une demande de préemption est envoyée à une tâche matérielle qui est en communication sur le FPGA. La connexion entre la tâche à commuter et les autres tâches dans le système doit être coupée. Selon notre solution, le contexte de la tâche et les données de communication peuvent être extraits du FPGA associé. Quand ces données sont sauvegardées, par exemple dans l'espace de mémoire partagée, le FPGA est disponible pour une autre tâche (cf. Figure 45c). Une nouvelle tâche peut démarrer ou continuer son exécution en restaurant son contexte et ses données de communication. Finalement, les tâches qui communiquent peuvent reprendre le flux de communication comme illustré dans la Figure 45e.

7.4.4 *Implémentation*

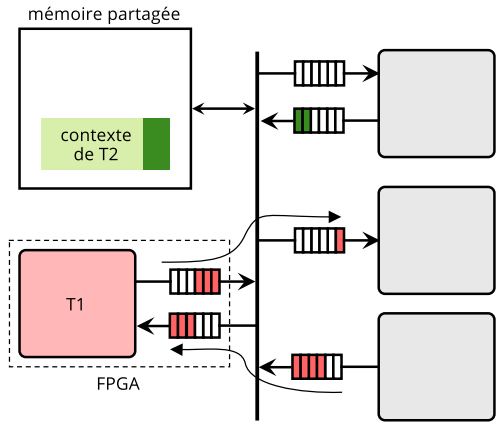
Nous avons implémenté la solution proposée dans un système reconfigurable qui contient un CPU, un FPGA et une mémoire partagée. Cette implémentation est constitué de la partie matérielle et la partie logicielle. L'implémentation matérielle est faite sur le FPGA alors que l'implémentation logicielle est faite en tant que module qui s'exécute sur un des CPUs dans le système.

Sur le FPGA, nous avons construit une infrastructure de communication comme illustré dans la Figure 46. Cette infrastructure de communication est associée à chaque tâche matérielle ou chaque IP qui est programmé sur les FPGAs dans le système reconfigurable. Elle intègre les FIFOs de communication entrée-sortie (E/S), les FIFOs de contexte[1], une machine d'état (FSM) et un point d'interconnexion. La FSM est responsable du contrôle du flux de communication et de synchronisation lors de l'exécution normale et du changement de contexte matériel. Elle est aussi responsable de transmettre la demande de préemption à l'IP. Le point d'interconnexion est nécessaire pour la conversion de protocole entre les FIFOs et l'interconnexion de communication externe.

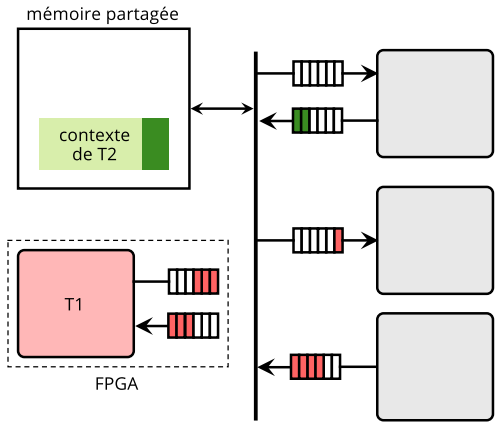
Les FIFOs E/S dans l'infrastructure de communication sont conçues pour respecter le protocole de changement de contexte matériel proposé dans la Figure 45. Elles permettent d'extraire et de restaurer les données de communication sans passer par ces ports standard. Le flux de communication entrant peut être redirigé à l'extérieur ou vice-versa durant un changement de contexte matériel. Les multiplexeurs sont ajoutés aux deux extrémités de ces FIFOs.

L'implémentation logicielle est faite sur un CPU dans le système reconfigurable. Nous avons développés les drivers qui permettent de contrôler l'infrastructure de

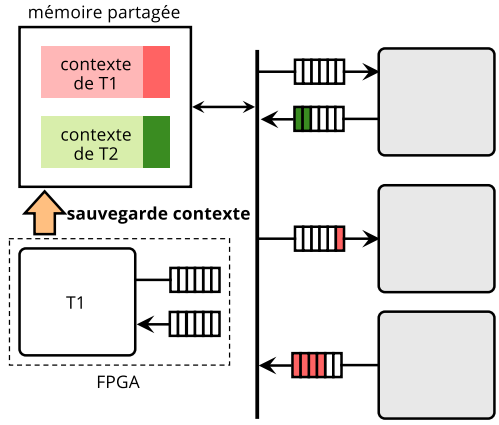
1 les contextes sont temporairement stockés dans ces FIFOs afin de s'adapter aux bandes passantes différentes entre l'intérieur et l'extérieur du FPGA



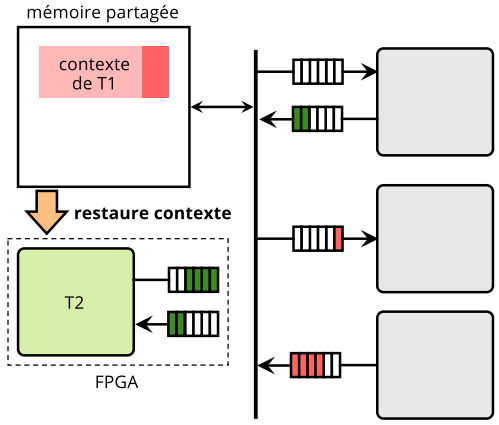
(a) T1 est en cours d'exécution et en communication



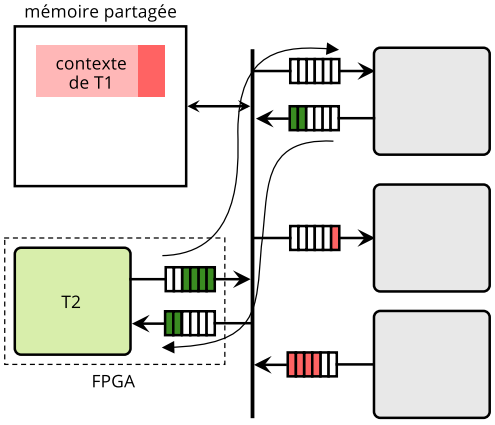
(b) La communication est déconnectée immédiatement



(c) Les données E/S et le contexte de T1 sont extraits



(d) Les données E/S et le contexte de T2 sont restaurés



(e) L'exécution de T2 est reprise

Figure 45: Le protocole de changement de contexte matériel

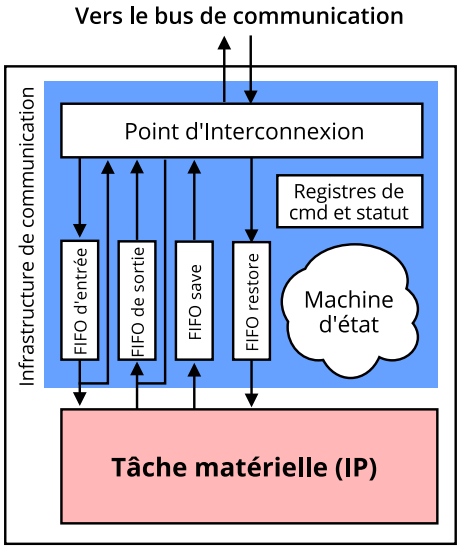


Figure 46: L'infrastructure de communication pour supporter le protocole de changement de contexte matériel avec la gestion de communication sur les FPGAs

communication sur le FPGA. Ces drivers prennent en compte les déclenchements qui démarrent l'exécution de tâche sur les FPGAs et interrompent l'exécution en cas de changement de contexte matériel. L'implémentation des drivers a été faite en tant que un module dans le noyau du système d'exploitation Linux. La Figure 47 présente le contrôle de connexion entre une tâche sur un FPGA et les autres tâches à l'extérieur.

La Figure 47 montre également comment le contexte et les données de communication sont stockés dans la mémoire partagée. Les tailles de données E/S ainsi que la taille du contexte sont inclus comme un en-tête. Ils sont tous intégrés comme un contexte d'une tâche matérielle qui prend en compte les données de communication durant le changement de contexte matériel.

7.5 EXPÉRIMENTATION ET RÉSULTATS

7.5.1 *Plate-forme d'expérimentation*

Les expérimentations ont été effectuée sur des plates-formes système-sûr-puce (SoC) reconfigurable. Deux SoCs ont été choisis pour montrer la généricité de notre solution, ZC706 de Xilinx et Arria V SoC de Intel Altera. Chaque SoC intègre un processeur embarqué du type ARM, une mémoire partagée (DDR) et un FPGA dans la même puce avec un bus de communication interne AXI. Le but d'utiliser ce type de plate-forme était de pouvoir lancer l'application en tant qu'une tâche matérielle sur le FPGA et de faire la gestion de tâche à partir du processeur ARM. Le processeur ARM est responsable de la configuration du FPGA, de la gestion

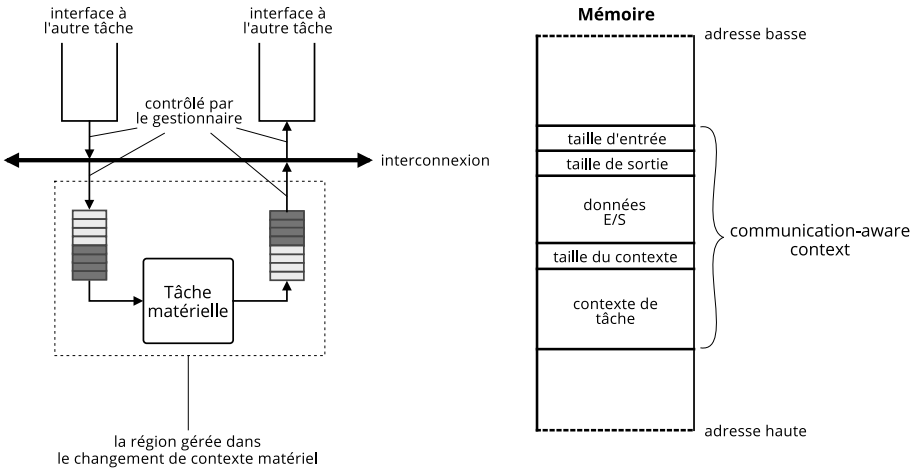


Figure 47: La gestion de contexte et des données de communication associées d'une tâche commutée

de communication entrées-sorties du FPGA et de la supervision du changement de contexte matériel.

Le même système a été implémenté dans les deux plates-formes SoCs. L'infrastructure de communication proposée est suffisamment générique et paramétrable pour que la modification supplémentaire en matérielle ne soit pas nécessaire. Du côté logiciel, une petite modification sur les drivers est nécessaire afin d'adapter l'interface de communication aux adresses disponibles en mémoire partagée.

7.5.2 *Caractérisation matérielle*

Cette section présente les résultats de caractérisation matérielle de notre solution. Ce sont principalement les ressources consommées dans le FPGA suite à l'implémentation de l'infrastructure de communication. Pour évaluer le surcoût matériel de notre solution, nous avons développé trois versions de l'infrastructure de communication. La première version, appelé *Basic*, est conçue sans la capacité du changement de contexte matériel. En version Basic, l'infrastructure de communication est utilisé uniquement pour la communication E/S. La deuxième version, appelé *CS*, a un support du changement de contexte matériel en plus de la communication E/S. La version CS contient les FIFOs de contexte mais elle implémente la solution classique presenté dans [Vu+16]. La troisième version de l'infrastructure de communication, appelée *CSComm*, a été développée avec notre protocole de changement de contexte matériel. Les Tableau 13 et Tableau 14 détaillent les ressources de FPGA consommées par l'infrastructure de communication dans les deux plates-formes utilisées.

Table 13: L'utilisation (post-placement) des ressources de ZC706 par l'infrastructure de communication

Version	Élements	LUT	FF	BRAM
Basic [a]	Interface AXI + registres	454	590	0
	FSM + mux	229	117	0
	FIFO E/S	0	0	1
	Total	**683**	**707**	**1**
CS [b]	Interface AXI + registres	454	590	0
	FSM + mux	641	543	0
	FIFO E/S	0	0	1
	FIFO Contexte	0	0	6
	Total	**1095**	**1133**	**7**
CSComm [c]	Interface AXI + registres	454	590	0
	FSM + mux	855	626	0
	FIFO E/S	0	0	1
	FIFO Contexte	0	0	6
	Total	**1309**	**1216**	**7**

[a]Sans support du changement de contexte
[b]Solution existante du changement de contexte
[c]Solution proposée

Table 14: L'utilisation (post-placement) des ressources de A5SOC par l'infrastructure de communication

Version	Élements	ALM	REG	M10K
Basic [a]	Interface AXI + registres	411	581	0
	FSM + mux	65	179	0
	FIFO E/S	0	0	2
	Total	**476**	**760**	**2**
CS [b]	Interface AXI + registres	411	581	0
	FSM + mux	499	801	0
	FIFO E/S	0	0	2
	FIFO Contexte	0	0	24
	Total	**910**	**1382**	**26**
CSComm [c]	Interface AXI + registres	411	581	0
	FSM + mux	642	897	0
	FIFO E/S	0	0	2
	FIFO Contexte	0	0	24
	Total	**1053**	**1478**	**26**

[a]Sans support du changement de contexte
[b]Solution existante du changement de contexte
[c]Solution proposée

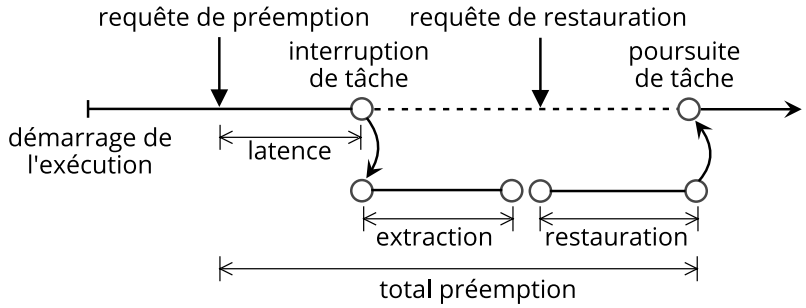


Figure 48: Le scénario de changement de contexte matériel utilisé dans l'expérimentation

7.5.3 *Caractérisation temporelle*

Afin d'évaluer la performance fournie par notre solution au système sur le changement de contexte matériel, nous avons fait la caractérisation temporelle. Nous avons notamment comparé la performance du changement de contexte matériel lorsque l'infrastructure de communication utilise la version CS et CSComm. Les expérimentations ont été effectuées avec une application IDCT et les applications de CHStone [Har+08]. Ces applications représentent les applications de traitement d'images et de chiffrement qui sont souvent accélérées sur FPGA.

Nous avons proposé un scénario qui inclut une exécution de tâche sur un FPGA et un changement de contexte matériel lorsque la tâche est en communication avec une autre tâche sur le CPU. La Figure 48 illustre ce scénario de test. Après chaque démarrage de l'exécution, une requête de préemption est envoyée aléatoirement par un ordonnanceur du système pour interrompre la tâche sur le FPGA. Ensuite, l'extraction de contexte (et des données de communication) est faite pour libérer le FPGA utilisé. La deuxième requête est envoyée plus tard pour restaurer la tâche sur le FPGA et reprendre l'exécution. Le temps entre l'arrivée de la requête de préemption et le démarrage de l'extraction est défini comme le temps de latence. Le temps total de préemption est donc la somme du temps de latence, du temps d'extraction et du temps de restauration.

Tout d'abord, nous avons mesuré le temps total d'exécution, y compris le changement de contexte matériel. Le Tableau 15 présente la comparaison de la moyenne du temps total d'exécution en cycles entre CS et CSComm sur ZC706. Notre solution ajoute un surcoût temporel en raison de l'extraction et de la restauration des données de communication. Ceci génère un surcoût temporel total relativement faible qui est présenté dans le tableau. D'après les expérimentations, notre solution a ajouté moins de 2% de surcoût temporel sauf pour l'IDCT, qui est une application rapide et ne devrait pas être préemptée. Pourtant, le surcoût temporel ajouté par notre solution sur l'IDCT est inférieur à 10%. Les mesures sur A5SOC donnent des resultats similaires.

Table 15: Comparaison des moyennes du temps total d'exécution (en cycle de FPGA) avec un changement de contexte matériel entre CS et CSComm sur ZC706

App	CS	CSComm	Surcoût
adpcm	20879	20933	0.26 %
aes	14184	14237	0.37 %
blowfish	146255	146583	0.22 %
gsm	12548	12619	0.57 %
idct	1388	1516	9.19 %
motion	13009	13268	1.99 %
sha	331767	332435	0.2 %

Ensuite, nous avons observé la latence de la préemption, ce qui correspond à la réactivité du système à la demande d'interruption qui arrive de l'ordonnanceur du système. Le Tableau 16 présente une comparaison des moyennes du temps de latence en préemption entre CS et CSComm sur ZC706. Contrairement à la solution classique (CS), la demande de préemption peut être traitée à tout moment par notre solution (CSComm). L'extraction de contexte peut être commencé presque immédiatement sans attendre que l'IP ait consommé toutes les données dans les canaux de communication. Ceci réduit énormement le temps de latence sur la plupart des applications dans nos expérimentations. Le temps total de préemption est diminué notamment pour les applications qui demandent beaucoup de transferts des données.

Avec nos expérimentations, nous avons réussi à montrer que la gestion de communication durant le changement de contexte matériel est nécessaire et qu'elle présente des avantages en terme de réactivité du système à la demande d'interruption. Notre solution génère un surcoût matériel raisonnable et un surcoût temporel faible. Dans la solution classique, l'estimation du temps total de préemption ne peut pas être fait à cause de données de communication qui ne sont pas gérées. Avec notre solution, nous sommes maintenant capables de mieux estimer le budget maximal de temps de préemption qui est la somme du temps d'extraction et de restauration du contexte et des données de communication.

7.5.4 *Application*

Grâce à notre mécanisme de gestion de communication, nous pouvons maintenir la cohérence de communication de tâches dans le support du changement de contexte matériel. Nous avons implémenté le support de changement de contexte matériel dans deux applications différentes. La première application est sur la migration des

Table 16: Comparaison des moyennes du temps de latence en préemption (en cycle de FPGA) entre CS et CSComm sur ZC706

App	CS			CSComm		
	Latence	Préemption totale	Ratio	Latence	Préemption totale	Ratio
adpcm	147	2472	5.93 %	6	2387	0.23 %
aes	148	3143	4.71 %	17	3066	0.56 %
blowfish	20017	24847	80.56 %	56	5177	1.08 %
gsm	178	1256	14.16 %	14	1168	1.16 %
idct	180	1039	17.3 %	10	1050	0.93 %
motion	876	9889	8.86 %	7	9278	0.07 %
sha	10153	10971	92.54 %	7	1467	0.51 %

tâches entre des FPGAs hétérogènes. La deuxième application est sur un système hypervisé.

L'application de migration hétérogène est développée pour démontrer la capacité de notre solution à réaliser un système dans les nuages. Les deux plates-formes utilisées dans les caractérisations de solution sont utilisées pour effectuer la migration de tâche. Elles sont connectées au même réseau Ethernet. Nous avons montré, avec cette application, qu'une tâche qui est en cours d'exécution sur un FPGA peut être migrée sur un autre FPGA qui n'est pas forcement de même nature.

Le support de changement de contexte matériel est fourni dans un système hypervisé pour virtualiser la ressource FPGA. Aujourd'hui, un hyperviseur est utilisé pour virtualiser les ressources de calcul afin d'augmenter la sécurité du système et la transparence d'accès. Dans un système CPU-FPGA, la ressource FPGA n'est pas encore bien virtualisée notamment pour partager cette ressource entre de multiples utilisateurs du système. Le changement de contexte matériel est nécessaire pour donner les accès au FPGA de façon équitable. Notre solution de changement de contexte matériel est capable de maintenir la cohérence des données de communication et de fournir la performance nécessaire à la virtualisation du FPGA.

7.6 CONCLUSION ET PERSPECTIVES

Dans ce travail, nous avons étudié l'aspect communication du changement de contexte matériel dans un système hétérogène reconfigurable. Nous avons proposé une solution pour garantir la cohérence de communication durant le changement de contexte matériel. La solution proposée est conceptuellement simple. Les données de communication qui sont temporairement stockées dans les FIFOs sur un FPGA

sont extraites avec le contexte d'une tâche matérielle quand la tâche est préemptée. De même manière, les données de communication et le contexte de la tâche sont restaurées à la reprise de son exécution.

Cette solution a été caractérisée dans notre expérimentation et nous avons montré que notre solution génère un faible surcoût temporel, fournit une grande réactivité à la demande de préemption au système, ainsi qu'une prédictabilité sur le temps de préemption. Nous avons démontré la généricité de la solution en réalisant le même environnement de test sur les FPGAs dans deux SoCs des fabricants différents : ZC706 (Xilinx) et Arria V SoC (Intel Altera). Deux applications ont été proposées avec notre solution. La première application est la réalisation d'un prototype de migration hétérogène sur les SoCs. La deuxième application est l'intégration de la capacité du changement de contexte matériel dans un système hypervisé.

Les perspectives de ce travail se déclinent en trois axes différentes. La première, axée sur le court terme, est l'implémentation de la méthode de changement de contexte matériel sur un FPGA multi-régions. La deuxième perspective propose l'intégration de notre solution aux systèmes distribués à une grande échelle. La dernière perspective concerne à la continuation de notre application sur un système hypervisé afin d'obtenir ses caractéristiques liées à la performance et à la consommation énergétique.

BIBLIOGRAPHY

[AYS06] Ben A Abderazek, Tsutomu Yoshinaga, and Masahiro Sowa. 'High-Level Modeling and FPGA Prototyping of Produced Order Parallel Queue Processor Core.' In: *The Journal of Supercomputing* 38.1 (2006), pp. 3–15 (Page 2).

[Afo+13] George Afonso, Zeineb Baklouti, David Duvivier, Rabie Ben Atitallah, Eli Billauer, and Stephan Stilkerich. 'Heterogeneous CPU/FPGA reconfigurable computing system for avionic test application.' In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE. 2013, pp. 260–267 (Page 46).

[Ahm+16] Sagheer Ahmad, Vamsi Boppana, Ilya Ganusov, Vinod Kathail, Vidya Rajagopalan, and Ralph Wittig. 'A 16-nm multiprocessing system-on-chip field-programmable gate array platform.' In: *IEEE Micro* 36.2 (2016), pp. 48–62 (Page 14).

[ABS10] Nikolaos Alachiotis, Simon A Berger, and Alexandros Stamatakis. 'Efficient PC-FPGA Communication over Gigabit Ethernet.' In: *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. IEEE. 2010, pp. 1727–1734 (Pages 9, 46).

[Bat+02] Joan Batlle, J Martı, Pere Ridao, and Josep Amat. 'A new FPGA/DSP-based Parallel Architecture for Real-Time Image Processing.' In: *Real-Time Imaging* 8.5 (2002), pp. 345–356 (Page 6).

[Bou16] Alban Bourge. 'Changement de contexte matériel sur FPGA, entre équipements reconfigurables et hétérogènes dans un environnement de calcul distribué.' PhD thesis. Grenoble Alpes, 2016 (Page 37).

[BMR16] Alban Bourge, Olivier Muller, and Frédéric Rousseau. 'Generating Efficient Context-Switch Capable Circuits through Autonomous Design Flow.' In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 10.1 (2016), p. 9 (Pages 24, 29, 37, 42, 45, 52, 55, 61, 101, 102).

[BL12] Alexander Brant and Guy GF Lemieux. 'ZUMA: An open FPGA overlay architecture.' In: *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE. 2012, pp. 93–96 (Page 25).

[Car+86] William S. Carter, Khue Duong, Ross H. Freeman, Hung-Cheng Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze. 'A User Programmable Reconfigurable Logic Array.' In: *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*. 1986, pp. 233–235 (Pages 2, 6).

[Che+14] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. 'Enabling FPGAs in the Cloud.' In: *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM. 2014, p. 3 (Pages 46, 87).

[Con+16] Jason Cong, Muhuan Huang, Di Wu, and Cody Hao Yu. 'Heterogeneous Datacenters: Options and Opportunities.' In: *Proceedings of the 53rd Annual Design Automation Conference*. ACM. 2016, p. 16 (Page 33).

[Cor] Intel Corporation. *Intel SoCs: When Architecture Matters*. Web. Last Accessed: June 9th 2018. URL: https://www.altera.com/products/soc/overview.html (Page 9).

[Cro+14] Louise H Crockett, Ross A Elliot, Martin A Enderwitz, and Robert W Stewart. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014 (Page 9).

[DeH96] Andre DeHon. 'Reconfigurable Architectures for General-Purpose Computing.' In: (1996) (Page 11).

[Den74] Jack B Dennis. 'First Version of a Data Flow Procedure Language.' In: *Programming Symposium*. Springer. 1974, pp. 362–376 (Page 10).

[Dev+10] Ludovic Devaux, Sana Ben Sassi, Sebastien Pillement, Daniel Chillet, and Didier Demigny. 'Flexible interconnection network for dynamically and partially reconfigurable architectures.' In: *International Journal of Reconfigurable Computing* 2010 (2010), p. 6 (Page 27).

[DML11] François Duhem, Fabrice Muller, and Philippe Lorenzini. 'FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA.' In: *International Symposium on Applied Reconfigurable Computing*. Springer. 2011, pp. 253–260 (Page 23).

[DML12] François Duhem, Fabrice Muller, and Philippe Lorenzini. 'Reconfiguration Time Overhead on Field Programmable Gate Arrays: Reduction and Cost Model.' In: *IET Computers & Digital Techniques* 6.2 (2012), pp. 105–113 (Pages 46, 93).

[ECF96] Carl Ebeling, Darren C Cronquist, and Paul Franklin. 'RaPiD—Reconfigurable Pipelined Datapath.' In: *International Workshop on Field Programmable Logic and Applications*. Springer. 1996, pp. 126–135 (Page 25).

[EV12] Ken Eguro and Ramarathnam Venkatesan. 'FPGAs for Trusted Cloud Computing.' In: *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE. 2012, pp. 63–70 (Page 87).

[Est60] Gerald Estrin. 'Organization of Computer Systems: The Fixed Plus Variable Structure Computer.' In: *Papers presented at the May 3-5, 1960, Western Joint IRE-AIEE-ACM Computer Conference*. ACM. 1960, pp. 33–40 (Page 6).

[FVS15] Suhaib A Fahmy, Kizheppatt Vipin, and Shanker Shreejith. 'Virtualized FPGA Accelerators for Efficient Cloud Computing.' In: *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*. IEEE. 2015, pp. 430–435 (Pages 2, 87).

[Fek+12] Sándor P Fekete, Tom Kamphans, Nils Schweer, Christopher Tessars, Jan C van der Veen, Josef Angermeier, Dirk Koch, and Jürgen Teich. 'Dynamic Defragmentation of Reconfigurable Devices.' In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 5.2 (2012), p. 8 (Page 46).

[GB03] Marc Geilen and Twan Basten. 'Requirements on the Execution of Kahn Process Networks.' In: *Programming languages and systems* (2003), pp. 319–334 (Pages 38, 103).

[Gol+99] Seth Copen Goldstein, Herman Schmit, Matthew Moe, Mihai Budiu, Srihari Cadambi, R Reed Taylor, and Ronald Laufer. 'PipeRench: A Coprocessor for Streaming Multimedia Acceleration.' In: *ACM SIGARCH Computer Architecture News* 27.2 (1999), pp. 28–39 (Page 25).

[Gua+08] Nan Guan, Qingxu Deng, Zonghua Gu, Wenyao Xu, and Ge Yu. 'Schedulability Analysis of Preemptive and Nonpreemptive EDF on Partial Runtime-Reconfigurable FPGAs.' In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 13.4 (2008), p. 56 (Pages 14, 94, 99).

[HTK15] Markus Happe, Andreas Traber, and Ariane Keller. 'Preemptive hardware multitasking in ReconOS.' In: *International Symposium on Applied Reconfigurable Computing*. Springer. 2015, pp. 79–90 (Pages 27, 29, 101).

[Har+08] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, Hiroaki Takada, and Katsuya Ishii. 'CHStone: a Benchmark Program Suite for Practical C-based High-Level Synthesis.' In: *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*. IEEE. 2008, pp. 1192–1195 (Pages 62, 110).

[HD10] Scott Hauck and Andre DeHon. *Reconfigurable Computing: the Theory and Practice of FPGA-based Computation*. Vol. 1. Morgan Kaufmann, 2010 (Page 7).

[HP11] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2011 (Pages 1, 6).

[Inc12] Xilinx Inc. *"AXI Reference Guide", UG761*. https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/v13_4/ug761_axi_reference_guide.pdf. [Online; accessed 7-June-2018]. Jan. 2012 (Page 9).

[Inc] Xilinx Inc. *Petalinux Tools*. Web. Last Accessed: June 21st 2018. URL: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html (Page 58).

[Jac+15] Matthew Jacobsen, Dustin Richmond, Matthew Hogains, and Ryan Kastner. 'RIFFA 2.1: A Reusable Integration Framework for FPGA Accelerators.' In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 8.4 (2015), p. 22 (Pages 8, 46).

[Jai+14] Abhishek Kumar Jain, Khoa Dang Pham, Jin Cui, Suhaib A Fahmy, and Douglas L Maskell. 'Virtualized Execution and Management of Hardware Tasks on a Hybrid ARM-FPGA Platform.' In: *Journal of Signal Processing Systems* 77.1-2 (2014), pp. 61–76 (Page 87).

[JG00] Philip James-Roxby and Steven A Guccione. 'Automated Extraction of Run-Time Parameterisable Cores from Programmable Device Configurations.' In: *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*. IEEE. 2000, pp. 153–161 (Page 23).

[JTW07] Slavisa Jovanovic, Camel Tanougast, and Serge Weber. 'A Hardware Preemptive Multitasking Mechanism Based on Scan-Path Register Structure for FPGA-based Reconfigurable Systems.' In: *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*. IEEE. 2007, pp. 358–364 (Pages 23, 24, 36).

[Joz+13] Krzysztof Jozwik, Shinya Honda, Masato Edahiro, Hiroyuki Tomiyama, and Hiroaki Takada. 'Rainbow: An Operating System for Software-Hardware Multitasking on Dynamically Partially Reconfigurable FPGAs.' In: *International Journal of Reconfigurable Computing* 2013 (2013), p. 5 (Pages 26–29, 45, 46, 101).

[Joz+12] Krzysztof Jozwik, Hiroyuki Tomiyama, Masato Edahiro, Shinya Honda, and Hiroaki Takada. 'Comparison of Preemption Schemes for Partially Reconfigurable FPGAs.' In: *IEEE Embedded Systems Letters* 4.2 (2012), pp. 45–48 (Page 12).

[Joz+10] Krzysztof Jozwik, Hiroyuki Tomiyama, Shinya Honda, and Hiroaki Takada. 'A Novel Mechanism for Effective Hardware Task Preemption in Dynamically Reconfigurable Systems.' In: *2010 International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2010, pp. 352–355 (Pages 23, 100).

[Kah74] Gilles Kahn. 'The Semantics of a Simple Language for Parallel Programming.' In: *In Information Processing* 74 (1974), pp. 471–475 (Pages 10, 11, 38, 103).

[KP05] Heiko Kalte and Mario Porrmann. 'Context Saving and Restoring for Multitasking in Reconfigurable Systems.' In: *International Conference on Field Programmable Logic and Applications (FPL), 2005.* IEEE. 2005, pp. 223–228 (Page 22).

[KSH05] Kenneth B Kent, Micaela Serra, and N Horspool. 'Hardware/Software Co-Design for Virtual Machines.' In: *IEE Proceedings-Computers and Digital Techniques* 152.5 (2005), pp. 537–548 (Page 87).

[KGS17] Oliver Knodel, Paul R Genssler, and Rainer G Spallek. 'Migration of Long-Running Tasks between Reconfigurable Resources using Virtualization.' In: *ACM SIGARCH Computer Architecture News* 44.4 (2017), pp. 56–61 (Page 46).

[KS15] Oliver Knodel and Rainer G Spallek. 'RC3E: Provision and Management of Reconfigurable Hardware Accelerators in a Cloud Environment.' In: *FPGAs for Software Programmers (FSP), 2015 2nd International Workshop on*. 2015 (Page 87).

[Koc+04] Dirk Koch, Ali Ahmadinia, Christophe Bobda, and Heiko Kalte. 'FPGA Architecture Extensions for Preemptive Multitasking and Hardware Defragmentation.' In: *Field-Programmable Technology (FPT), 2004. Proceedings. 2004 IEEE International Conference on*. IEEE. 2004, pp. 433–436 (Page 24).

[KHT07] Dirk Koch, Christian Haubelt, and Jürgen Teich. 'Efficient Hardware Checkpointing: Concepts, Overhead Analysis, and Implementation.' In: *Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays (FPGA)*. ACM. 2007, pp. 188–196 (Pages 24, 29, 36, 101).

[Lab] TIMA Lab. *AUGH: Autonomous and User Guided High-level synthesis.* Web. Last Accessed: June 23rd 2018. URL: http://tima.imag.fr/sls/research-projects/augh/ (Page 63).

[LLB14] Loïc Lagadec, Jean-Christophe Le Lann, and Théotime Bollengier. 'A Prototyping Platform for Virtual Reconfigurable Units.' In: *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on*. IEEE. 2014, pp. 1–7 (Page 14).

[LWH02] Wesley J Landaker, Michael J Wirthlin, and Brad L Hutchings. 'Multitasking Hardware on the SLAAC1-V Reconfigurable Computing System.' In: *International Conference on Field Programmable Logic and Applications (FPL)*. Springer. 2002, pp. 806–815 (Page 23).

[LP95] Edward A Lee and Thomas M Parks. 'Dataflow Process Networks.' In: *Proceedings of the IEEE* 83.5 (1995), pp. 773–801 (Page 11).

[Lev+00] L Levinson, Reinhard Männer, M Sessler, and Harald Simmler. 'Preemptive Multitasking on FPGAs.' In: *Field-Programmable Custom Computing Machines (FCCM), 2000 IEEE Symposium on*. 2000, pp. 301–302 (Pages 22, 23, 100).

[LDS07] Chuanpeng Li, Chen Ding, and Kai Shen. 'Quantifying the Cost of Context Switch.' In: *Proceedings of the 2007 Workshop on Experimental Computer Science*. ACM. 2007, p. 2 (Page 12).

[Li+10] Meng Li, Charbel Abdel Nour, Christophe Jego, and Catherine Douillard. 'Design and FPGA Prototyping of a Bit-Interleaved Coded Modulation Receiver for the DVB-T2 Standard.' In: *Signal Processing Systems (SIPS), 2010 IEEE Workshop on*. IEEE. 2010, pp. 162–167 (Page 2).

[LCH00] Zhiyuan Li, Katherine Compton, and Scott Hauck. 'Configuration Caching Management Techniques for Reconfigurable Computing.' In: *Field-Programmable Custom Computing Machines (FCCM), 2000 IEEE Symposium on*. IEEE. 2000, pp. 22–36 (Page 11).

[LF09] Wang Lie and Wu Feng-Yan. 'Dynamic Partial Reconfiguration in FPGAs.' In: *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on*. Vol. 2. IEEE. 2009, pp. 445–448 (Page 19).

[Liu+09] Ming Liu, Zhonghai Lu, Wolfgang Kuehn, Shuo Yang, and Axel Jantsch. 'A Reconfigurable Design Framework for FPGA Adaptive Computing.' In: *2009 International Conference on Reconfigurable Computing and FPGAs*. IEEE. 2009, pp. 439–444 (Page 46).

[LP08] Enno Lübbers and Marco Platzner. 'Communication and Synchronization in Multithreaded Reconfigurable Computing Systems.' In: *ERSA*. 2008, pp. 83–89 (Pages 26, 45, 101).

[Lys+05] Roman L Lysecky, Kris Miller, Frank Vahid, and Kees A Vissers. 'Firmcore virtual FPGA for just-in-time FPGA compilation.' In: *FPGA*. 2005, p. 271 (Page 25).

[Mei+03] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. 'ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix.' In: *International Conference on Field Programmable Logic and Applications*. Springer. 2003, pp. 61–70 (Page 25).

[Moo06] Gordon E Moore. 'Cramming More Components onto Integrated Circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff.' In: *IEEE Solid-State Circuits Newsletter* 3.20 (2006), pp. 33–35 (Pages 1, 6).

[Naj+17] Mohamad Najem, Théotime Bollengier, Jean-Christophe Le Lann, and Loïc Lagadec. 'Extended Overlay Architectures For Heterogeneous FPGA Cluster Management.' In: *Journal of Systems Architecture* (2017) (Page 25).

[NLG99] Walid A Najjar, Edward A Lee, and Guang R Gao. 'Advances in the Dataflow Computational Model.' In: *Parallel Computing* 25.13 (1999), pp. 1907–1929 (Page 10).

[Nar+11] Surya Narayanan, Daniel Chillet, Sebastien Pillement, and Ioannis Sourdis. 'Hardware OS Communication Service and Dynamic Memory Management for RSoCs.' In: *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*. IEEE. 2011, pp. 117–122 (Pages 26, 27, 45, 101).

[Ouy+14] Jian Ouyang, Shiding Lin, Wei Qi, Yong Wang, Bo Yu, and Song Jiang. 'SDA: Software-Defined Accelerator for Large-Scale DNN Systems.' In: *Hot Chips 26 Symposium (HCS), 2014 IEEE*. IEEE. 2014, pp. 1–23 (Page 87).

[PMR14] Adrien Prost-Boucle, Olivier Muller, and Frédéric Rousseau. 'Fast and Standalone Design Space Exploration for High-Level Synthesis under Resource Constraints.' In: *Journal of Systems Architecture* 60.1 (2014), pp. 79–93 (Pages 37, 55, 102).

[Put+15] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. 'A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services.' In: *IEEE Micro* 35.3 (2015), pp. 10–22 (Pages 33, 87, 95).

[RFC09] Kyle Rupnow, Wenyin Fu, and Katherine Compton. 'Block, Drop or Roll (back): Alternative Preemption Methods for RH Multi-Tasking.' In: *Field-Programmable Custom Computing Machines (FCCM), 2009. 17th International Symposium on*. 2009, pp. 63–70 (Page 19).

[SC06] Manuel Saldana and Paul Chow. 'TMD-MPI: An MPI Implementation for Multiple Processors across Multiple FPGAs.' In: *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*. IEEE. 2006, pp. 1–6 (Page 8).

[SV98] Stephen M Scalera and Jóse R Vázquez. 'The Design and Implementation of a Context Switching FPGA.' In: *Field-Programmable Custom Computing Machines (FCCM), 1998. Proceedings. IEEE Symposium on*. IEEE. 1998, pp. 78–85 (Page 11).

[SB15] Yakun Sophia Shao and David Brooks. *Research infrastructures for hardware accelerators*. Vol. 10. 4. Morgan & Claypool Publishers, 2015, pp. 1–99 (Page 6).

[SSS15] João Silva, Valery Sklyarov, and Iouliia Skliarova. 'Comparison of on-chip communications in zynq-7000 all programmable systems-on-chip.' In: *IEEE Embedded Systems Letters* 7.1 (2015), pp. 31–34 (Page 46).

[SLM00] Harald Simmler, L Levinson, and Reinhard Männer. 'Multitasking on FPGA Coprocessors.' In: *International Workshop on Field Programmable Logic and Applications (FPL)*. Springer. 2000, pp. 121–130 (Pages 22, 23, 26, 28, 100).

[Stu+15] J Stuecheli, Bart Blaner, CR Johns, and MS Siegel. 'CAPI: A Coherent Accelerator Processor Interface.' In: *IBM Journal of Research and Development* 59.1 (2015), pp. 7–1 (Pages 8, 46).

[TV02] Jim Tørresen and Knut Arne Vinger. 'High Performance Computing by Context Switching Reconfigurable Logic.' In: *ESM*. Vol. 2. 2002, pp. 207–210 (Page 11).

[Tri+97] Steven Trimberger, Dean Carberry, Anders Johnson, and Jennifer Wong. 'A Time-Multiplexed FPGA.' In: *Field-Programmable Custom Computing Machines (FCCM), 1997. Proceedings., The 5th Annual IEEE Symposium on*. IEEE. 1997, pp. 22–28 (Page 11).

[VB13] Wim Vanderbauwhede and Khaled Benkrid. *High-performance computing using FPGAs*. Springer, 2013 (Page 6).

[Ves+16] Malte Vesper, Dirk Koch, Kizheppatt Vipin, and Suhaib A Fahmy. 'JetStream: An Open-Source High-Performance PCI Express 3 Streaming Library for FPGA-to-Host and FPGA-to-FPGA Communication.' In: *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. EPFL. 2016, pp. 1–9 (Page 46).

[Vu+16] Hoang Gia Vu, Supasit Kajkamhaeng, Shinya Takamaeda-Yamazaki, and Yasuhiko Nakashima. 'CPRtree: A Tree-Based Checkpointing Architecture for Heterogeneous FPGA Computing.' In: *2016 Fourth International Symposium on Computing and Networking (CANDAR)*. IEEE. 2016, pp. 57–66 (Pages 24, 26, 28, 29, 36, 42, 66, 81, 101, 103, 107).

[VPI05] M Vuletić, Laura Pozzi, and Paolo Ienne. 'Seamless Hardware-Software Integration in Reconfigurable Computing Systems.' In: *IEEE Design & Test of Computers* 22.2 (2005), pp. 102–113 (Pages 26, 45, 101).

[WP02] Herbert Walder and Marco Platzner. 'Non-Preemptive Multitasking on FPGAs: Task Placement and Footprint Transform.' In: *Proceedings of the 2nd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA)*. 2002, pp. 24–30 (Pages 15, 16, 93).

[WBP13] Wei Wang, Miodrag Bolic, and Jonathan Parri. 'pvFPGA: Accessing an FPGA-based Hardware Accelerator in a Paravirtualized Environment.' In: *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on*. IEEE. 2013, pp. 1–9 (Page 87).

[Whe+01] Timothy Wheeler, Paul Graham, Brent Nelson, and Brad Hutchings. 'Using Design-Level Scan to Improve FPGA Design Observability and Controllability for Functional Verification.' In: *International Conference on Field Programmable Logic and Applications (FPL)*. Vol. 1. Springer. 2001, pp. 483–492 (Pages 23, 24, 101).

[Xie+15] Mimi Xie, Mengying Zhao, Chen Pan, Jingtong Hu, Yongpan Liu, and Chun Jason Xue. 'Fixing the Broken Time Machine: Consistency-Aware Checkpointing for Energy Harvesting Powered Non-Volatile Processor.' In: *Proceedings of the 52nd Annual Design Automation Conference*. ACM. 2015, p. 184 (Page 16).

[Xil17] Inc. Xilinx. *Virtex-4 FPGA Configuration User Guide*. June 2017 (Page 23).

[Xil] Xillybus. *An FPGA IP core for easy DMA over PCIe with Windows and Linux*. Web. Last Accessed: July 8th 2018. URL: http://xillybus.com/ (Page 8).

[Zha+15] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 'Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks.' In: *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM. 2015, pp. 161–170 (Page 6).

LIST OF PUBLICATIONS

[Wic+16a] Arief Wicaksana, Alban Bourge, Olivier Muller, and Frédéric Rousseau. 'Demonstration of a context-switch method for heterogeneous reconfigurable systems.' In: *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE. 2016, pp. 1–1.

[Wic+17] Arief Wicaksana, Alban Bourge, Olivier Muller, Arif Sasongko, and Frédéric Rousseau. 'Prototyping dynamic task migration on heterogeneous reconfigurable systems.' In: *Proceedings of the 28th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*. ACM. 2017, pp. 16–22.

[Wic+16b] Arief Wicaksana, Olivier Muller, Arif Sasongko, and Frédéric Rousseau. 'Validation automatique d'une methode de migration des tâches sur la plateforme Zynq.' In: *Journées Nationales du Réseau Doctoral en Micro-nanoélectronique (JNRDM), 2016 19ème édition des*. CNFM. 2016, pp. 1–6.

[Wic+16c] Arief Wicaksana, Adrien Prost-Boucle, Olivier Muller, Frédéric Rousseau, and Arif Sasongko. 'On-board non-regression test of HLS tools targeting FPGA.' In: *Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*. ACM. 2016, pp. 41–47.