Università degli Studi di Cagliari

DOTTORATO DI RICERCA

Dottorato di Ricerca in Ingegneria Elettronica e Informatica

Ciclo XXVIII

TITOLO TESI

Dataflow Based Design Suite for the Development and Management of Multi-Functional Reconfigurable Systems

Settore/i scientifico disciplinari di afferenza

ING-INF/01

Presentata da: Carlo Sau

Coordinatore Dottorato Prof. Fabio Roli

Tutor Prof. Luigi Raffo

Esame finale anno accademico 2014 – 2015

Ph.D. in Electronic and Computer Engineering
Dept. of Electrical and Electronic Engineering
University of Cagliari

Dataflow Based Design Suite for the Development and Management of Multi-Functional Reconfigurable Systems

Carlo Sau

Advisor: Prof. Luigi Raffo
Ph.D. Coordinator: Prof. Fabio Roli
Curriculum: ING-INF/01

XXVIII Cycle
A.A. 2014-2015
March 2016

Acknoledgements

Carlo Sau gratefully acknowledges Sardinia Regional Government for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.).

Abstract

Embedded systems development constitutes an extremely challenging scenario for the designers since several constraints have to be meet at the same time. Flexibility, performance and power efficiency are typically colliding requirements that are hardly addressed together. Reconfigurable systems provide a valuable alternative to common architectures to challenge contemporarily all those issues. Such a kind of systems, and in particular the coarse grained ones, exhibit a certain level of flexibility while guaranteeing strong performance. However they suffer of an increased design and management complexity.

In this thesis it is discussed a fully automated methodology for the development of coarse grained reconfigurable platforms, by exploiting dataflow models for the description of the desired functionalities. The thesis describes, actually, a whole design suite that offers, besides the reconfigurable substrate composition, also structural optimisation, dynamic power management and co-processing support. All the provided features have been validated on different signal, image and video processing scenarios, targeting either FPGA and ASIC.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to the large diffusion of powerful embedded electronics, nowadays the world is in the midst of the portable multi-media era. From customer electronics to internet of things and wearable devices to medical implantable platforms, systems need to cope with providing extremely high real-time performance on battery operated scenarios. Technology scaling and, in turn, the growth of transistors number on the same die have normally guaranteed enough computational power to fit with the increased amount of target functionalities and the required performance level. Smaller gates allow the integration of more logic within a single die, additional logic that can be exploited for providing the needed flexibility and high performance. However, on the one hand the Moore's law in the last years is slowing down its prevision, since the doubling of transistors in a dense integrated circuit is now taking more than two years. On the other, portability has raised a critical constraint on the embedded systems development, the power consumption, that can not be easily addressed with the mere technology scaling. The scaling of technology alone is no longer sufficient to address this strongly challenging scenario and designers are required to act since the early stages of the design flow in order to develop flexible, high performance and power efficient devices.

Usually flexibility, performance and power efficiency are colliding requirements in the embedded systems design. Strongly flexible platforms are hardly efficient while executing the single functionalities. At the same time, devices capable of achieving maximum performance on a given task difficultly manage to serve different processing requests and, anyway, nor at a similar performance rate. In this sense, an appealing architectural trend is represented by reconfigurable devices, that can reach a compromise between the extreme flexibility of general purpose units and the strong performance guaranteed by application specific circuits. In this thesis work, coarse grained (CG) reconfigurable systems have been addressed. With respect to other reconfigurable architectures, CG devices guarantee fast configuration times while waiving some flexibility, that is limited to a little amount of supported functionalities. This makes CG reconfigurability suitable for application specific real-time contexts, where only few different behaviours are required. CG reconfigurable solutions can be adopted either in ASIC, incrementing the intrinsic flexibility of this high performance circuitry, or in FPGA architectures, to provide a further level of flexibility besides

the native fine grained one. In the FPGA case, CG reconfigurability could be combined with the dynamic partial reconfiguration to achieve very small chips capable of switching, at run-time, among different sets of functionalities. The main drawback of reconfigurable devices turns out to be the complexity of their design under several aspects: resources mapping, optimisation, hardware design, run-time management. Moreover, reconfigurability alone is still unable to solve the power issue. In order to aid designers in the systems development, taking care of multiple project constraints, automated tools can provide a valid support during all the phases of the design flow.

In the last decade, the adoption of dataflow models of computation by electronic systems designers has increased, driven by the diffusion of multi-core and multi-processor platforms. Dataflows are modular specifications that can natively manage execution concurrency typical of multi-processor environments. Besides that, dataflow modularity can also be exploited in order to speed-up system development, especially dealing with coarse grained reconfigurable devices: a direct correspondence between dataflow entities and hardware blocks can be considered. Dataflows can also enable early stage analysis and optimisation, helping in the hard task of meeting the constraints for some extremely strict contexts (e.g. signal processing for implantable biomedical devices). For these distinctive features, dataflow modules constitute, together with CG reconfigurability, the pillar of the approach adopted in this thesis.

1.1 Objectives of the Thesis

The main objective of this thesis work is the development of an automated methodology for the design and management of CG reconfigurable systems. The methodology, leveraging on the dataflow models of computation, will be able to provide the register transfer level description of a reconfigurable substrate starting from the high level specifications of the desired functionalities. Exploiting the dataflow models features, the generated reconfigurable substrate will be able to autonomously manage its own run-time configuration, relieving the designer by this onerous task.

Beyond the main objective, several secondary goals are expected to be achieved:

- the methodology has to be fully automated, eventually exploiting already existent dataflow oriented tools, so that all the steps of the reconfigurable systems design flow (resource mapping, optimisation, hardware design and run-time management) will be supported;
- a smart generation of the reconfigurable substrate (adopting low level information if required) has to be provided, making it possible to address specific application contexts that are characterised by extremely strict constraints;
- advanced strategies for limiting power consumption have to be supported, so that the resource redundancy typical of reconfigurable devices will not restrict the applicability of the proposed methodology in battery operated environments;

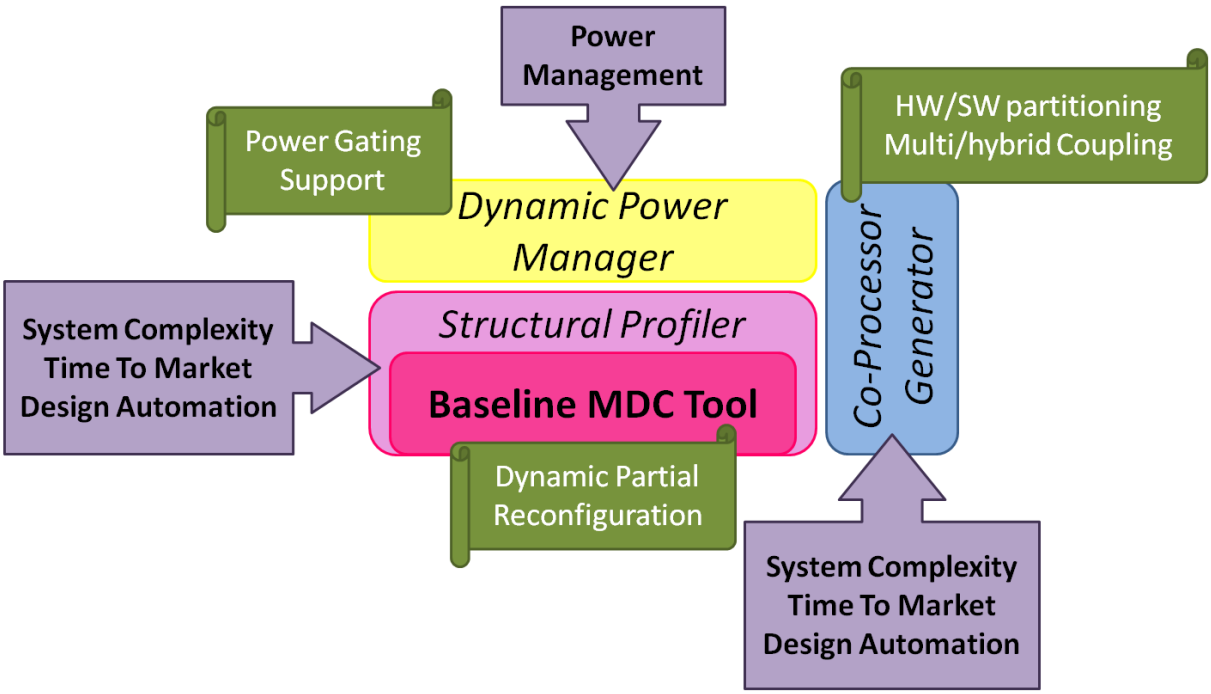


Figure 1.1: Multi-Dataflow Composer CG reconfigurable systems design suite overview.

- rapid prototyping and integration of the generated reconfigurable systems have to be favoured by providing, apart the reconfigurable substrate, also a proper interface and support logic letting the system a ready-to-use hardware accelerator.

1.2 Subject of the Work

During the thesis work, an automated methodology for the development of CG reconfigurable system has been studied. The main outcome of the study is the Multi-Dataflow Composer (MDC), a software tool in charge of designing and managing CG reconfigurable systems with the minimal designer effort. Actually, besides the simple MDC, a whole design suite is available, as depicted in Figure 1.1. The suite core is represented by the MDC tool, capable of composing a reconfigurable datapath from a set of input functionalities, provided as dataflow graphs, and taking care of the run-time configuration of the generated system. Around this baseline functionality, different extensions or additional features have been studied and built in order to obtain a more complete and powerful approach to the problem of designing and managing CG reconfigurable systems.

MDC has been equipped with a structural profiler that, exploiting some target specific low level information, can drive the designer to the best structural solution, in terms of graph topology, according to the selected design goal. A dynamic power manager has been necessary to address specific scenarios where meeting the typical design constraints is made extremely challenging by strict system consumption requirements. Finally, the co-processor generator characterises a software and hardware logic layer around the reconfigurable datapath generated by the baseline MDC

functionality, so that a ready-to-use co-processing IP for Xilinx environments is provided. Note that, in Figure 1.1 green parchments report some hints about possible future improvements of the developed suite features.

1.3 Thesis Structure

The thesis structure firstly provides, in Chapter 2, a brief state of the art of the concepts at the basis of the proposed approach: reconfigurable computing, dataflow models of computation and power issue. The rest of the thesis follows the main blocks composing the design suite shown in Figure 1.1:

- Chapter 3 will discuss the baseline functionality of the design suite, consisting in the composition of a reconfigurable multi-functional dataflow model, starting from a set of dataflows expressing all the wanted functionalities, and in the generation of the corresponding register transfer level description;
- Chapter 4 will complete the treatment about the baseline functionality by showing as, thanks to the integration of the developed suite with some other dataflow oriented tools, it has been assembled a fully automated tool chain for the design and management of reconfigurable systems, where the only effort that designers have to spent is in the single functionalities modelling;
- Chapter 5 will present the structural profiling additional feature of the suite, that brings the capability of optimising the generated reconfigurable substrate according to the project effort selected by the designer and exploiting some low level (post-synthesis) information retrieved from the netlists of the isolated input functionalities implementations;
- Chapter 6 will propose the dynamic power manager extension of the baseline feature, in charge of avoiding the waste of power typically of reconfigurable architectures (due to mutually exclusive logic among different functionalities) by means of a coarse grained automated implementation of the clock gating technique;
- Chapter 7 will describe the co-processor generator, additional feature provided to the design suite with the aim of speeding-up the prototyping and integration of the composed substrate through the generation of proper interface and support logic, resulting in a whole ready-to-use reconfigurable hardware accelerator.

Lastly, Chapter 8 will report some considerations about the thesis work, going down in the detail for all the presented features and giving some directives for future improvements.

Chapter 2

Literature

In the last decades the modern society has been pervaded by a huge variety of portable embedded devices, such as smartphones, media players and tablets. These devices integrate multiple functionalities and have typically to deal with multimedia data, thus facing problems such as the growing amount of information and the continuous evolution of coding standards. Considering the video media, it is emblematic the evolution from the CIF (352x288 pixels) resolution of the H.261 coding standard [2], dated 1988, to the Full Ultra HD (8192x4320 pixels) one supported by HEVC [3] since 2013. In order to address this complex scenario, the embedded systems are required to achieve every day higher performance, to provide a wider flexibility and to be very easily evolvable. The performance is necessary to quickly (eventually in real-time) manage the whole data amount, the flexibility to implement all the wanted functionalities and the easy evolvability to have a fast response in relation to new standard releases.

Generally speaking, the execution performance of a given functionality is strictly related to the specialisation of the system for the same functionality. A system highly specialised in executing a certain functionality can involve ad-hoc logic or it can exploit the parallelism of the same functionality by providing multiple parallel processing channels. In this way, the specialised system may achieve a very good execution efficiency for the considered functionality. On the contrary, if the system is generic, that means it mainly involves logic that is not exclusively used in a specific application but that can be employed for different operations, it probably will not reach high performance when executes a single application, even though it is able to be flexible by providing different functionalities.

The evolvability is a bit more complicated aspect of embedded devices. It can be seen as the capability of the system to be quickly updated according to modifications of the supported functionalities. In particular, in order to obtain an easy evolution of the system the resources re-usability among different updates has to be maximised, so that only the parts of the system that have actually changed will be re-designed. A modular system, where atomic operations or functionalities pieces are encapsulated onto bounded blocks, may strongly favour evolvability. In such a kind of systems modules that are no longer useful or that have to be modified can be easily and rapidly replaced with new ones while requiring a minimum integration effort.

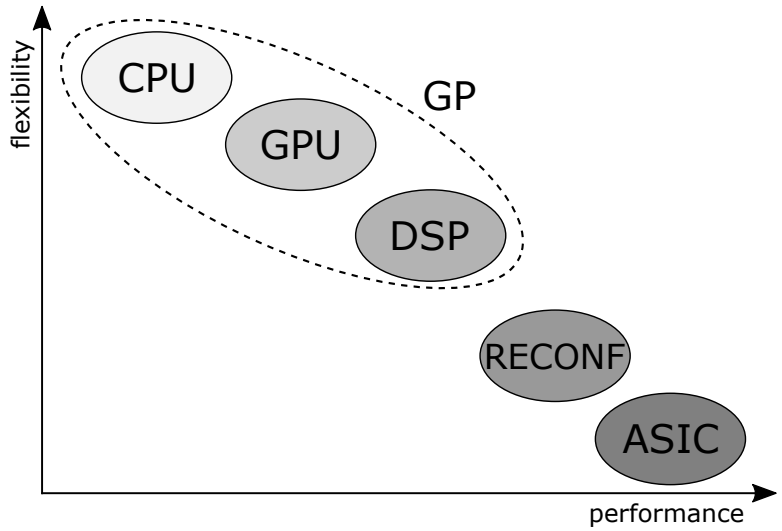


Figure 2.1: Flexibility versus performance graph of the main architectural alternatives for processing systems.

In literature there is a wide variety of processing systems that provide different degrees of performance and flexibility. Figure 2.1 depicts a graphical overview of how the existing processing systems are spread between the axes representing these two metrics. Closed to the flexibility axis there are general purpose systems, such as the Central Processing Units (CPUs) commonly adopted in personal computers. These devices are able to execute a huge quantity of functionalities, since they support different programming languages that, through proper compilers, are translated in machine code. The machine code is the sequence of instruction calls of the CPU that leads to the execution of a specific functionality. The instructions supported by CPUs are generic operations, like the transfer of a data from the main memory to one of the processor registers or a simple addition between the values contained onto two registers. They are not conceived for a particular application. Actually, by means of programming languages the user can model its custom applications with a degree of flexibility that is the maximum achievable for an electronic device.

Leaving the flexibility axis, there are some devices that can be seen as general purpose ones, but that are more suitable for a specific kind of applications. Examples of this category of devices are Graphical Processing Units (GPUs) and Digital Signal Processors (DSPs). The GPUs are processing systems dedicated to the execution of tasks related to the image and video processing. Their architecture is designed for the exploitation of the data parallelism that typically characterises this kind of applications. Despite GPUs are conceived for a specific class of operations, they are able to provide a huge variety of functionalities due to their support for several programming languages. However, the best performance is achievable only with the image and video processing applications and/or if the application is written in order to optimally exploit the GPU architecture. At this purpose several API and frameworks aiming at the usage of GPUs for general purposes (GPGPUs) by modifying, automatically or not, the generic code of a given application have been proposed in literature (e.g the CUDA architecture from NVIDIA [1] or the OpenCL library [8]).

DSPs are another kind of almost general purpose devices that provide specific

machine instructions for signal processing operations. The most common additional instructions provided by DSPs are multiply and accumulate, typically useful in matrix operations, filtering and transforms. Very often DSPs natively support fixed and floating point arithmetic through dedicated logic units, thus speeding-up the operations involving these number formats. Furthermore, the DSP architectures can also be able to exploit data parallelism that may be present in the signal processing applications. Like for GPUs, DSPs have dedicated compilers to support programming languages and, theoretically, to run any kind of program, but they achieve the best performance only if the executed code contains common signal processing operations and/or if it is written in order to fully exploit the system potentials (DSP primitives may be necessary).

All the mentioned devices, CPUs, GPUs and DSPs, can be grouped within the general purpose systems since, by means of programming languages, they can execute any kind of application. However, for the same reason, they are not uniquely designed to perform a specific task. The execution of a single instruction in general purpose systems is composed of different stages: instruction fetch, instruction decode, execution, memory and write back. Each stage can last one or more clock cycles and an application can be composed of several thousands of instructions. Even though the standard execution stages can be pipelined or replicated, it is straightforward that this is not the most efficient way to execute a given functionality. To push the performance, the design approach has to be reversed: if for the general purpose systems the wanted functionalities are adapted to a prefixed architecture, in maximum performance systems the architecture has to be adapted to the wanted functionality. This is what occurs for the Application Specific Integrated Circuits (ASICs) on the opposite side of the flexibility versus performance graph of Figure 2.1. ASICs, as the same acronym reveals, are systems that are designed exclusively for the execution of a given functionality. All and only the involved resources are needed to perform the functionality and the architecture is forged accordingly to the native execution flow of the application. Obviously, the design can be affected by other constraints, such as the minimisation of the consumed power, besides the performance maximisation. This may lead to different versions of ASICs for a given application. ASICs are not flexible at all because no other operations can be performed, apart the one around which the system has been built.

A huge difference between the adoption of general purpose systems or ASICs for the execution of applications stands on the designer effort, costs and required skills. Dealing with general purpose systems the designer has firstly to choose the target architecture among a wide variety of available devices (also cheap solutions are possible), and then to shape its code accordingly. He has not to worry about the system architecture design, so hardware designing skills are not necessary. Dealing with ASICs is different, since the architecture itself has to be characterized, thus requiring very specific hardware designing expertise. The development of ASICs is generally a longer, more expensive and more complex process than adopting general purpose systems.

Additional choices are possible between the two described extreme points, general purpose systems and ASICs, of the flexibility versus performance graph in Figure 2.1. An appealing option, constituting a trade-off among the previous systems, is given by reconfigurable computing. Reconfigurable devices are circuits highly spe-

cialised in the execution of a limited set of functionalities: they show strong execution efficiency features while providing a certain degree of flexibility. Unfortunately, even though reconfigurability fits the flexibility and specialisation needs, still it does not suffice stand-alone to solve the whole scenario issue, since it suffers of the same problems than ASICs. The development of reconfigurable systems, as will be discussed in Section 2.1, is complex and requires a specific expertise, limiting its wide application and quick evolution. For this reason, reconfigurable devices are hardly capable of meet the strict time to market typical of electronics.

In order to overcome the problems of the reconfigurable systems development and evolution, the present work tries to exploit the dataflow paradigm. Dataflows own distinctive features, like modularity and parallelism highlighting, that may help designers targeting a particular class of reconfigurable systems. A detailed description of the dataflow paradigm and its main features, with a focus on its existent links with reconfigurable computing, is provided by Section 2.2.

Dealing with modern embedded devices, one important aspect to be taken into consideration is the system consumption. As a matter of fact, the portable era has raised the problem of power dissipation to the first positions of the design constraints. Battery life limits force devices to exploit each single energy particle in the maximum efficient way. So that, a dedicated power management strategy is usually defined within the project of modern embedded systems. Section 2.3 will provide a deep discussion of power management with respect to embedded devices, with a particular emphasis on the main design trends for the consumptions reduction.

The rest of the chapter is organised as follow: Section 2.1 will describe the main aspects of reconfigurable computing and will show a survey of the main reconfigurable architectures proposed in literature. Section 2.2 will present the dataflow paradigm and the particular formalism that has been adopted in this thesis work. Section 2.3 involves a deepening on the very actual power issue and on the main techniques to reduce consumption in embedded electronic devices. Lastly, Section 2.4 will discuss the main advancements of the present thesis work with respect to the state of the art.

2.1 Reconfigurable Computing

Reconfigurable computing refers to a class of digital electronic system architectures that combine the flexibility typical of software programmed systems to the high performance of the hardware implementations. They involve typically a fixed set of, often programmable, processing logic connected by routing elements that are also programmable [38]. The processing logic is managed in a time multiplexed manner, in order to employ the same resource for different applications at different execution instants. Reconfigurable systems are often called adaptive, meaning that the logic units and interconnects of the system can be modelled to fit a specific functionality by programming it at hardware level [91].

Basically, reconfiguration may be applied at design-time, before the system execution, or at run-time, during the system execution. The reconfiguration at design-time is less difficult to manage and it is supported especially by architectures that require long configuration phases. The run-time reconfiguration, often called dy-

namic reconfiguration, needs a closer look on the system internal state during the functionality switches. Besides for providing multiple functionalities, dynamic reconfiguration can be exploited for tuning the system basing on real-time measurements on the system itself, thus obtaining an auto-adaptive control loop.

Field Programmable Gate Arrays (FPGAs) are the most common example of reconfigurable platforms. They typically provide a very high degree of flexibility through a large density logic substrate. However FPGAs configuration requires a huge amount of information and time. For this reason the reconfigurability mainly provided by this kind of devices is at design-time. Since the last ten years, FPGAs are able to support also a sort of run-time reconfiguration through the so called partial reconfiguration [96] [12], also known as dynamic partial reconfiguration. Partial reconfiguration is based on the configuration of a portion of the design during the execution by exploiting a set of previously generated configurations stored on a dedicated memory that is accessed from a configuration module. Due to their flexibility FPGAs are widely adopted in literature for different purposes, ranging from the prototyping of ASICs (before their costly fabrication) to the acceleration of the most computationally intensive sections of an application that runs on a general purpose machine.

In the following two main aspects of reconfigurable systems will be detailed: architectures will be discussed in Section 2.1.1, dealing with the physical structure of these devices and proposes a little survey of the main trends in literature; the design flow will be described in Section 2.1.2), where the typical steps and issues of the reconfigurable platforms development will be underlined.

2.1.1 Architectures

The core of a reconfigurable system is the reconfigurable fabric. As previously introduced, it is composed by a set of optionally reconfigurable processing elements (PEs), a reconfigurable interconnect and a flexible interface to connect the fabric with the external world. Each of the fabric components, PEs or interconnect, is characterised by the same trade-off (flexibility versus efficiency) introduced in the beginning of this chapter. Generally speaking, the more a component is flexible, and then the more it is able to fit the applications requirements, the slower it is with respect to a less flexible component, that can easily turn out to be also smaller in area and less power consuming [93].

Besides basing on the colliding degrees of flexibility and efficiency, PEs and interconnects can be classified in terms of granularity and composition. The granularity defines the scale of the components and it is typically classified onto coarse grained and fine grained. The granularity of the PEs is commonly assumed as the granularity assigned to the whole reconfigurable fabric. Composition refers to the components typology and it can be homogeneous, when all the components are identical, or heterogeneous, for fabrics with mixed kinds of components.

Usually, only a subset of the application functionalities is accelerated within a reconfigurable system. The code sections that cannot be efficiently accelerated on hardware, such as variable length loops and branch control statements, are usually executed on a general purpose device. High density computation code sections are more suitable to be accelerated with reconfigurable cores, since an appreciable speedup is achievable for their execution, if compared with general purpose devices

[38]. The whole architecture of a reconfigurable system then involves one or more processors, one or more reconfigurable fabrics and one or more memories. Basing on the coupling between the host processor and the reconfigurable fabric, a further classification for reconfigurable systems can be done. They can range from a loosely coupling solution with the reconfigurable fabric connected to the processor by means of the standard communication channels (e.g. the system bus), to a tightly coupling one where the reconfigurable fabric and the processor are close and rely on dedicated links.

Granularity

The granularity is a key feature of the reconfigurable fabric, since it determines the level of detail that each PE can manage. Granularity heavily affects the system configuration process because, depending on the amount of stuff that has to be configured within the substrate, the needed information and, in turn, the configuration time are determined. Additionally, the granularity is strictly related to the system flexibility, meaning that if a reconfiguration can be performed at a higher level of detail, the system will fit better around the wanted functionality. As previously introduced, there are two main granularity classes for reconfigurable systems: fine grained (FG) and coarse grained (CG).

Fine Grained architectures FG reconfigurable systems involve PEs that perform simple functions on a single or a small number of bits. Similarly, the FG interconnects can differentiate the data routing at the same bit level than the PEs one. The most common FG PEs are the small Look-Up Tables (LUTs) adopted to implement combinatorial logic on commercial FPGAs. These circuits are able to perform any logic function for the supported number of inputs, once configured with a proper bitstream. FG reconfigurable systems involve a huge amount of PEs in order to implement complex functionalities. In turn, the amount of data necessary to configure these PEs can easily become very big and the configuration phase, that is the downloading of the configuration data, gets slow. For this reason the run-time reconfiguration of the FG devices is hardly achieved.

The most common FG reconfigurable devices are FPGAs. The latest top range commercial FPGAs [99] [14] can embed more than one million LUTs with up to eight input each. LUTs in FPGAs are commonly packed in groups along with some bit-wise memory elements (Flip-Flops (FFs)). The blocks involving LUTs and FFs are the effective PEs of FPGAs, by the point of view of the reconfigurable architecture. Thanks to their strong flexibility, FPGAs can be adopted on the practice in different ways [88]: as reconfigurable glue logic for high performance interfacing layers form data sources to processors [28], as hardware accelerators for a single functionality supporting high speed data exchange [43] or as flexible accelerators able to speed-up several software applications within a wider server structure. The strong efficiency, both in terms of performance and power, of these kind of devices is very close to the one typical of ASICs, especially if compared to general purpose systems like CPUs and GPUs [10].

However the same main strength of FPGAs, the flexibility, is also one of their weakness, due to the long time required by the configuration phase. As previously

mentioned, the partial reconfiguration provides a way to partly overcome this issue. Kohn [59] exploited the partial reconfigurability in order to develop a hardware accelerator for Sobel and Sepia filtering within a full high definition video pipeline. In this case partial reconfiguration allowed the reduction of the configuration bitstream size from more than 4 MB to less than 135 kB, leading to a saving time over 95% with respect to a full system reconfiguration.

Besides FPGAs, other typologies of FG reconfigurable systems have been proposed in literature. Chiu et al. [55] developed a FG architecture whose PEs involve two four-ports LUTs and one FF, similarly to the common FPGA ones. The architecture, called FMRPU, is organised as an array of 4096 overall PEs with a hierarchical interconnect layer that guarantees short combinatorial paths. FMRPU has been conceived for high throughput and data parallel applications. It has been validated and compared with some similar devices on motion estimation and digital signal processing operations. Other solutions, like the one proposed by Agarwal et al. [9], differentiated the FG PEs by packing inside only combinatorial logic: four 3-input LUTs and three 4-bit ripple-carry adders. The PEs can be configured (43 configuration bits are required for each PE) in order to implement generic logic functions with up to five inputs, one 4-bit adder in parallel with an 8-bit one, one 4-input 4-bit adder or a 4-bit multiplier. The FFs needed to store operands and intermediate results are integrated in the interconnect layer as a register bank with 64 entries 32-bit each. A unique register bank is provided to serve six different PEs. The architecture, prototyped on a 32 nm CMOS technology, is intended to provide scalable and high efficiency computing power on microprocessor platforms for the digital signal and media processing.

Due to the large diffusion of FPGAs, FG custom reconfigurable architectures are not common, since the technology, flexibility and design tools provided by the main FPGA vendors are yet strongly optimised and not very expensive.

Coarse Grained architectures In order to overcome the problem of slow configuration phases on FG systems, CG architectures waive some flexibility by adopting bigger PEs. CG PEs are typically constituted by arithmetic and logic units (ALUs) along with a significant amount of storage. They can still be reconfigurable and their interconnect is typically at the same coarse granularity. CG architectures can also achieve higher area efficiency and simpler placement and routing phases than FG ones [48]. The modern FPGAs, even though being substantially FG architectures, provide some CG functional blocks and memories. Indeed, when adopted to implement arithmetic functions, FG structures require more area, latency and consume more power than the corresponding CG blocks [93]. Actually, although FPGAs are the most common FG reconfigurable architectures, they should be better described as mixed grained reconfigurable architectures, since they involve both FG and CG blocks.

In literature several different CG reconfigurable systems have been proposed. Hartenstein [48] introduced a classification for CG architectures based on the layout and interconnections of the involved PEs: linear arrays, mesh-based and crossbar-based. CG reconfigurable architectures organised as single or multiple linear arrays are very suitable for the mapping of execution pipelines. Within linear arrays, natural interconnects are the ones connecting each PE with the two nearest neighbour ones,

the predecessor and the successor in the pipeline. However, in presence of forks, additional links spanning among the whole array and proper routing resources are required. A straightforward example of a linear array CG reconfigurable architecture that maps a pipeline is given by Smaragdos et al. [84]. The authors face the problem of low reliability in modern systems due to the transistors size decrease. In order to overcome this issue they propose a fault tolerant multi-core architecture, where the fault tolerance is achieved by means of CG reconfigurability: the pipeline stages of the cores are interleaved by switching elements so that, if a stage within a core becomes faulty, it can be replaced by the same stage belonging to a different core, thus guaranteeing the instruction execution completion. In this case the CG architecture PEs are the stages constituting the pipelines (five stages are provided for each pipeline), while the interconnects are implemented by local crossbars that can connect each stage with all the previous and the successive ones located on the system cores. Linear array CG reconfigurable architectures are not always used to directly map a pipeline. In some cases, linear arrays are rather used to implement CG reconfigurable platforms similar to VLIW processors, such as for the Montium processing tile [49]. This reconfigurable processing unit involves five identical 4-inputs 16-bit ALUs and ten 2-inputs memory blocks with 512 16-bit words. ALUs are also provided with a local storage by means of an input registers bank. The interconnect is able to serve each input of each ALU by all the memory ports and by the outputs of any ALU, including itself. One ALU, the two overlying memories, the related registers bank and interconnect constitute the so-called *processing part*, the same as VLIW issues. Differently form a standard VLIW processor, the Montium processing tile ALUs can be also connected in cascade in order to implement more complex functionalities.

The most common class of CG architectures for reconfigurable systems is the mesh-based one. It is basically a rectangular two dimensional array of PEs, where the most common interconnects are 4 (North, South, East, West) or 8 (including also the diagonal directions) nearest neighbour short links. Examples of these kind of architectures are the works of Niedermeier et al. [68] and Paul et al. [72]. In [68] an eight by eight two dimensional mesh of PEs, with an additional first column of memory elements, is proposed. PEs perform arithmetic and logical operations on integer or fixed point numbers. They are provided with 8 nearest neighbours local links, global connections to the remaining nodes and broadcast channels from external inputs. The architecture reconfiguration is achieved by means of a dataflow representation of the mapped functionalities. This representation is basically a sequence of finite state machines that can be easily (one to one correspondence) implemented by the involved PEs. Paul et al. [72] have developed another kind of two dimensional mesh type CG reconfigurable architecture: reMORPH. This platform is target specific, since it has been built upon a Xilinx FPGA. reMORPH exploits DSP48 and BRAM CG blocks of the FPGA as cores of the array PEs. Thanks to a small additional LUTs amount, DSP48 constitute a 5 stage pipeline ALU with 512 data registers and 512 instruction registers. Each so assembled PE, provided also with a local BRAM memory block, can rely on four nearest neighbour communication channels. Reconfiguration is achieved by rewriting the PEs local memories and by reorganising the interconnects through the FPGA partial reconfiguration feature. The authors propose also a programming model for the architecture, aiming at minimising the reconfiguration overhead. Dealing with large dataset applications, reMORPH is affected by a common issue of the

CG reconfigurable systems, that is input/output memory bottleneck.

A particular architectural view of a two dimensional mesh for CG reconfigurable systems aiming at overcoming the memory bottleneck is Layers [77]. As the name suggests, the structure is ordered in layers according to the nature of implemented activity: processing, communication and memorisation. The lower layer is the processing one and it is basically a 4-rows 4-columns array of reconfigurable PEs (ALUs supporting Q-format fixed point arithmetic, including MAC) with 4 nearest neighbour connections and additional upstream and downstream lines with the overlying communication layer. This latter involves a cascade of routing resources in charge of providing an efficient interface between processing and memory elements. The upper layer contains the storage elements that guarantee 32 downstream links, for the operands reading, and 16 upstream links, for the result writing. Having an architecture with a distinct layer for each kind of resource allows the control and optimisation of the implemented functionalities separately, thus better facing the storage access bottleneck problem.

CREMA [44] is another mesh-based CG reconfigurable architecture that is structured as a two dimensional array with four rows and eight columns of PEs. Each PE is composed by a processing core, routing resources (multiplexers) and a configuration memory. The processing core is able to perform on the two 32-bit operands several functions, such as addition, multiplication and shifting, and supports floating point numbers. The two operands of each PE can be given by 4 nearest neighbours (West, North-West, North, North-East) or by the output of the same PE. Further connections allow the communication with farer PEs, in particular the ones standing on the same row or column. Reconfiguration is stored within the configuration memory and can be performed in a single clock cycle. Besides this run-time reconfigurability, CREMA can provide design-time reconfigurability through a dedicated configuration tool. This feature is intended to tailor the CG reconfigurable system, accordingly to the characteristics of the applications that have to be mapped, in order to obtain a strongly resource efficient system. Bergman et al. [21] developed a CG reconfigurable architecture called QUKU that, similarly to CREMA, provides both design and run-time reconfiguration. QUKU is a variable size two dimensional array where each PE is a small computational module, whose bit width and supported instructions are configured at design-time. PEs embed also input and output memories along with a configuration controller, responsible for the run-time reconfiguration. Like CREMA, QUKU can rely on a whole automated flow aiming at an easy mapping of applications on the reconfigurable architecture and aiding the designer during the management of the two supported levels of reconfigurability. AVATAR [53] constitutes a step up of CREMA, expressly shaped for the real-time processing of Fast Fourier Transform in some telecommunication applications. These use cases required superior computational power than the one achievable by the original CREMA architecture. In order to fit the mapped functionality, the columns of the two dimensional array have been increased from eight to sixteen. Two data memories of 512 words 32-bit each, partitioned onto 32 double ports blocks, are provided, one for the input data and one for the results. The interconnects allow the supply of each first row PE by any input memory block and the access of each of the last row PE to any output one. The derived application specific AVATAR architecture is able to execute different lengths and types of Fast Fourier Transforms.

Another application specific CG reconfigurable system has been presented by Miyoshi et al. [65]. The authors address the issue of accelerating the stream data processing in modern database systems. At this purpose they designed DR-SPE, a run-time reconfigurable architecture supporting query modification, addition, and optimisation. The system PEs are simple ALUs performing arithmetical and logical instructions. They are linked with seven nearest neighbour PEs by means of switch boxes and the whole system relies on a global input/output stream controller. Configuration registers placed inside the PEs, the switching boxes and the stream controller make it possible the system reconfiguration in terms of performed operations and data routing.

Ansaloni et al. [16] tried to move further through the enlargement of the reconfigurable platforms granularity. The authors proposed a coarser grain of reconfigurability called *expression* grain. Their expression grained reconfigurable array (EGRA) is essentially a two dimensional mesh of PEs, where each of these latter is not longer a simple ALU, but rather a cluster of 32-bit ALUs organised in rows (number of rows, number of ALUs per row and functionalities of the ALUs are customizable) interleaved by switching bars. EGRA can embed also different kinds of PEs, such as memories and multipliers. The interconnect is 4 nearest neighbour based, with additional row and column global buses. The array dimension is still customizable and the design process is aided by a dedicated tool that performs a complete design space exploration in order to optimise the mapping of applications in the resulting system.

Other works [15] [86] explored the possibility of taking advantage by adopting different kinds of granularity on the same substrate, in order to achieve both high flexibility guaranteed by FG architectures and strong performance obtained by CG ones. Amagasaki et al. [15] focused their studies on the development of a variable grain logic cell (VGLC) encapsulating either a 4-bit arithmetic unit and random logic. In particular each VGLC has four units, each one involving a two input 1-bit full adder and a two input LUT which share some common logic. These components let the VGLC having five different operation modes. The authors provided also a mapping tool in order to support the designer during the mapping of applications onto the presented architecture. A similar approach has been adopted for the development of DeSyRe [86] that, leveraging on a mixed grain texture, can manifest adaptivity and fault tolerance features. In particular CG logic is not very defect tolerant but its efficiency in terms of resources and power is excellent; on the contrary, FG logic can be strongly defect tolerant but it has relevant performance, power and cost overheads. The possibility of implementing functionalities on both CG and FG logic makes it possible the quick adaptation of the system to different faulty situations or to design constraints and application requirements, such as throughput or load balancing.

The last typology of CG reconfigurable architectures introduced by Hartenstein [48] is the crossbar-based, that relies on a crossbar switch, the most powerful communication network. Despite its adoption in FG reconfigurable systems is very common, a full crossbar is only rarely employed in CG architectures. The crossbar is rather used in different reduced configurations. It is the case of PADDI-2 [102], one of the first works on crossbar-based CG reconfigurable architectures that adopts a two level communication structure composed by a local network, connecting four PEs within a unique cluster, and a global crossbar network, connecting the various clusters. The crossbar is not full, but provides a reduced number of programmable

switches that can split a single long bus in some shorter ones without area penalty. The PEs of PADDI-2 are 16-bit ALUs that can perform arithmetical and logical functions, included multiplication. As previously said ALUs are grouped in clusters and can be pipelined in order to implement more complex or faster instructions. PADDI-2 is aimed to process complex digital signal applications and leverages on a distributed data-driven control based on dataflow graphs in order to manage the architecture in a scalable and modular way. Inoue et al. [54] explored the trade-off of having different crossbar configurations within a system involving the same VGLC as [15]. The authors showed that, by reducing the routing tracks of a full crossbar solution, the area and the same routing performance can be improved. The cost that has to be paid for this improvement is a decreasing of the processing speed in terms of maximum operating frequency.

The classification provided by Hartenstein [48] can aid in giving an overview of possible architectures for CG reconfigurable systems, but can also be adopted for FG or mixed grained solutions (see Table 2.1). Very often, the same architecture can belong, partially or globally, to several or none of the considered classes. For instance Yan et al. [101] proposed a system hierarchically structured in two main levels: the highest detail reveals a linear array architecture composed by five configurable PEs accessing a shared memory area and provided with a unique control and configuration unit; this sub-system is called reconfigurable processing unit (RPU). RPUs are then clustered four by four onto reconfigurable processing nodes (RPN) disposed as a two dimensional mesh. Connectivity is managed by a dedicated central bus inside the linear array of RPUs, while horizontal and vertical channels are exploited within the two dimensional array. The same apply for the work of Heysters et al. [49] where, if also the memory blocks are considered as PEs, despite the Montium processing tile architecture has been grouped within the linear array class, it belongs to the crossbar-based class. Indeed, the communication between ALUs (computational PEs) and memory blocks (storing PEs) is obtained by means of a configurable crossbar structure. Furthermore, in order to overcome memory bottleneck issues in some application fields (e.g. turbo coding), the authors suggest the adoption of more than one processing tile: the disposition of the tiles can lead the system to an additional class, maybe mesh-based, for CG reconfigurable architectures.

Modern trends of embedded devices that have to deal with several and strong constraints are pushing the CG reconfigurable systems to non conventional architectures. In particular, the above mentioned approach inversion that occurs going from general purpose systems to ASICs may affect also the CG reconfigurable systems design. The applications that have to be mapped within the CG platform are not longer fitted within the prefixed architecture, eventually exploiting the provided configurability. Rather, the CG system is shaped exactly around the wanted set of functionalities, maximising the efficiency in terms of power, resources usage and performance. Obviously, the flexibility versus efficiency trade-off still applies and the price paid with this strong specialisation is an equally strong decrease of flexibility. At any rate, extremely application specific CG reconfigurable systems lead to non conventional architectures that are not classifiable to any of the categories proposed by Hartenstein [48]. As it will be clearer in the following chapters, this thesis work focuses on such extremely application specific CG reconfigurable systems.

By the presentation of the CG reconfigurable architectures proposed in literature

clearly emerged a common issue of this kind of systems: the complexity of mapping applications and control their execution. This issue had already turned out in the older presented architectures, like Montium [85]. Newer works, in order to overcome the problem, provide along with the reconfigurable system also a dedicated software able to support the designer during the system development and management. The main trends of the development tools will be better explained in Section 2.1.2. Some interesting works aimed at relaxing the mapping and management issues by distributing, within the architecture, self control logic. It is the concept of *elasticity* proposed by Huang et al [51]. Basically, an elastic CG reconfigurable array exploits a typical dynamic dataflow control, that does not need a static scheduling of the applications, but data propagates throughout the system along with the respective control token. In this way, complex scheduling situations due to latency variations (e.g. when a memory access occurs) are handled transparently with token propagation.

Composition

Depending on the typology of the involved PEs, reconfigurable architectures can have different compositions. In particular it is easy to distinguish two main kinds of systems: homogeneous architectures, that involve a unique typology of PEs, and heterogeneous architectures, that exploit several kinds of PEs. Homogeneity is very common in reconfigurable architectures, since it eases the process of mapping the applications within the PEs. Among the architecture examples previously presented there is plenty of homogeneous architectures either for FG ([9] [55] and CG ([49] [68] [72] [77] [65]) systems. Homogeneity is also somehow related to the flexibility versus efficiency trade-off, common argument of this chapter. On the one hand, the availability of identical and replicated PEs limits the degree of specialisation, and in turn efficiency, that the architecture configurations can achieve if compared with dedicated heterogeneous blocks. On the other hand, homogeneous PEs are more generic than heterogeneous ones and can implement a wide number of functionalities.

The most common homogeneous PEs are ALUs. They generally perform arithmetic (sum, subtract) and logical operations (shift, compare) on two or more integer and in some cases fixed point numbers. Sometimes also the multiplication is supported, but it is also often performed by dedicated nodes in heterogeneous architectures. Depending on the kind of addressed applications, the ALUs can also support floating point arithmetic. PEs often embed some memory resources, in terms of input/output registers bank or as a small local memory.

As the requirements of efficiency for applications execution has become stronger, the diffusion of heterogeneous architectures increased. FPGAs started integrating dedicated memory and multiplier blocks on their texture in order to avoid the waste of resources needed to implement these functionalities on generic logic. The introduction of these additional elements within the array boosted the performance for some applications, such as digital signal processing ones. Newest FPGA devices integrate also more complex blocks, like digital signal processors, able to perform a variety of multiply and accumulate functions.

Again, the border between the two composition classes, homogeneity and heterogeneity, is very thin and strongly depends on the point of view of the observer. The architecture proposed by Niedermeier et al. [68] involves a two dimensional array of identical computational PEs, plus an additional first column of memory resources. If these latter are considered as PEs belonging to a different typology than the computational ones, this architecture can be seen as an heterogeneous system. The Montium processor tile [49] introduced before and so far considered as homogeneous architecture is, actually, an heterogeneous structure if the memory blocks are considered as PEs; again it can be considered homogeneous if the view is taken on an array of processing tiles. Also in DR-SPE [65] it is possible to observe the system on two different hierarchical levels (both with the same kind of composition): a lower one is composed by homogeneous PEs, simple ALUs called operation units, disposed as a two dimensional array; an upper layer is composed by another homogeneous two dimensional array whose tiles are operation units grouped within a so called operation unit block.

Some presented CG reconfigurable architectures give the possibility of customising the PEs accordingly to the applications that have to be executed [44] [21]. In these cases it is provided a template of the whole system where PEs are modular ALUs whose operations are implemented by different blocks. Through a dedicated tool, the set of wanted functionalities is analysed and mapped within the architecture. Once mapped all the functionalities, the resources of the system are optimised for their execution by the pruning of all the unnecessary logic. Each ALU is analysed and all the modules that perform unused operation are removed from the design. Also the interconnect is shaped around the mapped functionalities, so that redundant connection are avoided and multiplexer/demultiplexer sizes are minimised. In this way, starting from a homogeneous substrate, an heterogeneous strongly efficient system is obtained.

Note that granularity and composition are complementary aspects of a reconfigurable architecture. Morpheus [92] constitutes an heterogeneous reconfigurable systems on chip (SoC) that integrates three different granularities of reconfigurable arrays: FG, medium-grained (CG for the considered 2 ways classification) and CG. The local composition of these three computing cores can be again defined independently from the composition of the global system: the medium-grained device is an homogeneous architecture, differently from the CG one that is heterogeneous.

Coupling

Often in the practice reconfigurable architectures are not adopted alone, but rather one or more reconfigurable architectures are included within a whole system involving one or more processors and one or more memories. Reconfigurable architectures are then used as hardware accelerators, that are units in charge of aiding the processor in speeding-up the execution of a certain set of operations. An example of a whole system integrating reconfigurable architectures is the REMUS processor [104]. This reconfigurable SoC has been able to execute a whole H.264 video decoder in real-time. REMUS embeds two ARM processors, a dedicated ASIC for entropy decoding and two CG reconfigurable arrays, these latter involving overall 512 processing engines. Only regular and calculation intensive portions of the application (macro-

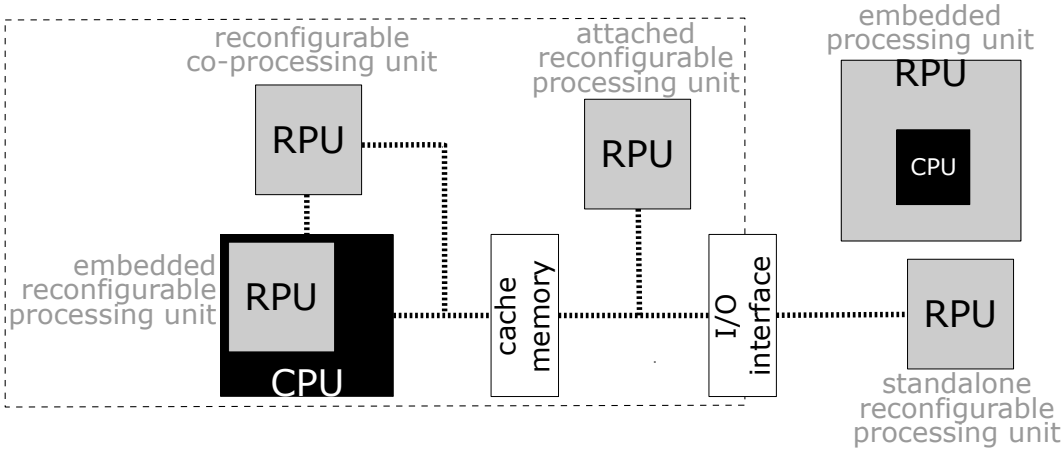


Figure 2.2: Different levels of coupling for reconfigurable architectures.

block decoding) are executed on the CG reconfigurable hardware, while the most control flow ones, apart from the entropy decoding that is processed by the ASIC, are executed by the processors.

Compton et al. [38] provided a classification for reconfigurable hardware accelerators according to the degree of coupling between the same accelerator and the processor (see Figure 2.2). In particular a device that is far from the host processor is said to be in a loosely coupling configuration, whereas an accelerator that is close to the host processor is in a tightly coupling one.

The loosest degree of coupling is represented by an external standalone reconfigurable processing unit that is accessed by the processor through the existing communication channels, such as the system bus. In this situation the communication between the two units is typically slow and applications that frequently require the processor intervention are not very suitable. It is the case of a wide set of presented architectures [55] [9] [49] [68] [72] [77] [44] [102]. In standalone coupling configurations, the data to be processed by the reconfigurable accelerator is often stored within a main memory, that is accessible throughout system bus by both the same reconfigurable accelerator and the host processor. If the amount of data that have to be transferred from the main memory to the reconfigurable accelerator and from this latter to the main memory is huge, the reconfigurable accelerator can be provided with a dedicated Direct Memory Access (DMA) module in charge of transparently handle all the memory transactions [53] [16] [92].

Moving closer to the processor, the reconfigurable hardware accelerator can be placed either between the cache structure and the standard communication channels or between the processor itself and the cache structure. The former case is labelled as attached reconfigurable processing unit and the latter as co-processor. Co-processors can be accessed through the same link that connects the processor to the cache structure or by means of dedicated channels. In both these intermediate cases, ad hoc modifications of the processor ordinary interface and cache structure are required. However, the communication costs decrease due to the bypassing of the standard communication channels. An historical example of co-processor is given by MorphoSys [82], an homogeneous two dimensional mesh of 16-bit ALUs with 4 nearest neighbour local links and three additional global interconnects. Mor-

phoSys is coupled with a RISC processor through a dedicated link. The processor, besides running general purpose operations, is able to configure and manage the execution of specific operations delegated to the accelerator and to supervise the memory transfers. Dealing with more recent devices, we have mentioned the QUKU architecture [21], coupled as a co-processor with the host processor that is, in this case, a the Microblaze soft-core by Xilinx [98]. The reconfigurable accelerator has a dedicated communication channel, the Xilinx Fast Simplex Link (FSL) [95], used for data transfers, while the configuration is sent by the system bus. The FSL is a point-to-point interconnect managed in a FIFO way. It can guarantee the exchange of a continuous stream of data between the host processor and the co-processor. Actually, the term co-processor is widely adopted to refer to generic hardware accelerators, while the level of coupling with the host processor is explicitly specified. Also in the following, and in particular in Chapter 2.2, the term co-processor will be used to identify two hardware accelerators with different levels of coupling.

The last, tightest solution is achieved by embedding the reconfigurable accelerator within the processing unit. In this case the accelerator represents one of the processor issues, making it possible the definition of custom instructions. Communication latency is minimal but the internal architecture of the processor is strongly affected by the insertion of additional sub-modules. A straightforward case of this tight level of parallelism is the RAW microprocessor [89]. It is a four by four array of 8-bit multi-grained PEs (with a local cache memory and configurable either as 8 stage MIPS style processors or as 4 stages pipelined floating point unit) connected with 8 nearest neighbours switched local links. RAW allows the shaping of the instruction set accordingly to the application that has to be processed within the processor, in order to achieve execution performance and energy efficiency at the same time. Other devices let the designer the possibility of customising only a subset of the whole processor instruction set. The Xtensa architecture by Tensilica [90] is an extensible processor with a fixed base instruction set that can be extended through the definition of additional custom instructions, implemented on a programmable logic. The EGRA CG reconfigurable architecture [16] supports the Xtensa extensible host processor and can thus be coupled at a very tightly level by exploiting custom instructions. Also the fault tolerant reconfigurable pipelines proposed by Smaragdos et al. [84] and integrated within DeSyRe [86] can be grouped with the embedded reconfigurable accelerators, even though the reconfiguration is exploited in order to achieve the fault tolerant behaviour.

A further class for the coupling between reconfigurable hardware accelerators and processors has been proposed by Todman et al. [93]. The reconfigurable hardware accelerator, usually an FPGA, is a standalone unit that incorporates a general purpose processing unit. The processor can be a hard-core or a soft-core. A hard-core processor is built from dedicated silicon, always physically present within the reconfigurable device. The most common cases of hard-processors are the the Zynq UltraScale+ multiprocessor SoC from Xilinx [100], that can rely on a Quad ARM Cortex-A53 ARM and on a Dual Cortex-A53, and the Stratix 10 SX SoC from Altera [14], that embeds a Quad-core ARM Cortex-A53. Soft-core processors refers to processing units that are implemented through the resources of the underlying reconfigurable fabric. Again, dealing with the main FPGAs vendors, Xilinx and Altera, the main examples are given by the Microblaze [98] and the Nios II [13] processors respectively.

Hard-core processors are usually more efficient than soft-core ones, since they are optimised and not limited by the processing fabric speed. However they are fixed or anyway not so flexible devices, though they can exploit the surrounding reconfigurable logic. Soft-core processors, despite being not so fast due to the reconfigurable logic limitations, can be customized and tuned freely. Furthermore, they allow the assembling of multi-core architectures, accordingly to the available resources.

For the sake of completeness, Table 2.1 resumes the architectures mentioned in this section along with a possible classification in terms of granularity, layout (it refers to the classification proposed by Hartenstein for CG devices[48]), composition and coupling with the host processor. As repeatedly discussed previously, an exact classification is often not possible due to the low borders among the different classes. So that, Table 2.1 has to be intended as an indicative survey summary of the state of the art reconfigurable devices.

Table 2.1: Summary of the presented reconfigurable architectures along wiht the related classification in terms of granularity, layout (as proposed in [48], composition and coupling.

architecture	granularity	layout	composition	coupling
Xilinx 7 series [99]	FG(CG)	mesh	homo(hetero)	-
Altera Stratix 10 [14]	FG(CG)	mesh	homo(hetero)	-
FMRPU [55]	FG	mesh	homo	standalone
[9]	FG	linear	homo	standalone
[84]	CG	linear	hetero	embedded
Montium [49] [85]	CG	linear(crossbar)	homo(hetero)	standalone
[68]	CG	mesh	homo(hetero)	standalone
reMORPH [72]	CG	mesh	homo	standalone
Layers [77]	CG	mesh	homo	standalone
CREMA [44]	CG	mesh	homo/hetero	standalone
QUKU [21]	CG	mesh	homo/hetero	co-processor
AVATAR [53]	CG	mesh	hetero	standalone
DR-SPE [65]	CG	mesh	homo	-
EGRA [16]	CG(EG)	mesh	hetero	standalone
VGLC [15] [54]	FG/CG	crossbar	homo	-
DeSyRe [86]	FG/CG	mesh	hetero	embedded
[102]	CG	crossbar	homo	standalone
Morpheus [92]	CG/FG	mesh	homo/hetero	standalone
REMUS [104]	CG	mesh	homo	standalone
MorphoSys [82]	CG	mesh	homo	co-processor
RAW [89]	CG/FG	mesh	homo	embedded

2.1.2 Design Flow

The design flow of reconfigurable systems consists of three main tasks: mapping, optimisation and management (see Figure 2.3). The resources mapping refers to the choice of which resources have to be adopted to execute every atomic operation of the given applications. This process is typically performed at design-time, but

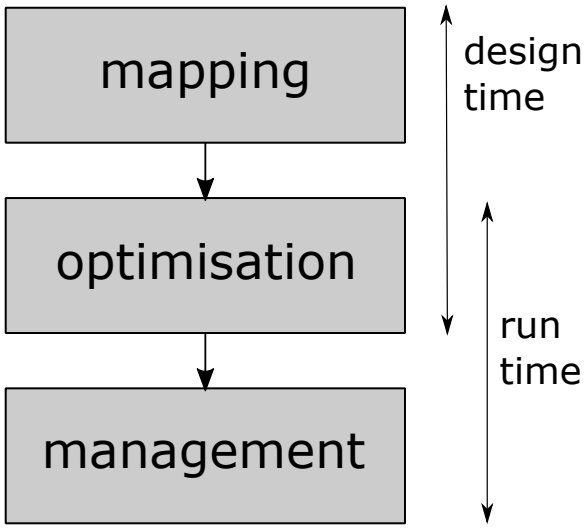


Figure 2.3: Typical design flow steps for reconfigurable systems. The reported time information are indicative.

sometimes the resources employed for each operation can be identified at run-time. The optimisation is the process of making the architecture as suitable as possible for the mapped applications, basing on predefined constraints or requirements. Optimisation can be performed only once the mapping has been defined. Design-time optimisations are more common than run-time ones, since the overhead of this process, that is often achieved through a whole design space exploration, can easily lead to unacceptable latencies. The management is the run-time process of controlling the execution contexts, with particular attention to the system internal state, and the rearrangement of configurable PEs and connections that has to be as quick and efficient as possible [32].

Developing a reconfigurable system turns out to be a very complex process. Firstly it requires specific hardware designing expertise for the definition or, anyway, for the full exploitation of the reconfigurable substrate. This feature has to be coupled with the applications knowledge in order to properly map the single operations to be performed onto dedicated, eventually programmable, processing logic. In this sense, basing also on the development team skills, there are two main approaches: shaping the hardware according to the given set of applications or fit the applications to a prefixed reconfigurable substrate. With the former approach it is possible to achieve the best efficiency on several aspects (area, power consumption, performance), since the architecture can be modelled exactly around the wanted functionalities: all and no more than the resources necessary to fit the required constraints are adopted. The latter approach, on the contrary, is limited by the fixed architecture and, in order to fit the required constraints, needs to perform a proper mapping or, if necessary, to modify the applications according to the available resources. However, choosing an existent architecture is, generally, less time and cost expensive than developing a new one.

Once mapped the selected functionalities within the available resources, the system has to be optimised. The optimisation phase is not always required: if the system is shaped around the applications the optimisation phase usually coincides with

the mapping one, since the developed PEs are yet optimised for the execution of the operations they have been conceived to. On the contrary, if applications are shaped over an existing device or if the developed PEs let the designer a certain degree of flexibility, the optimisation phase is necessary to fit the constraints. The optimisation can be focused on the resources, by pruning all the unnecessary logic in order to minimise the area, on the performance, by exploiting all the features that the reconfigurable texture offers, or on the energy, by finding the less consuming configuration. Note that the mentioned metrics can be also constrained all or partially at the same time.

The system management is a run-time aspect that has to be carefully taken into consideration during the reconfigurable platform development. The run-time reconfiguration in particular allows the user to change the system configuration and perform different functionalities in different times. In this way, besides achieving flexibility, it is possible to provide some kind of virtual hardware [38], since all the supported functionalities would be implemented by a bigger system than the one obtained through the reuse of the same PEs for different purposes at different times. The change of functionality within the system is performed through the configuration, a bit-stream in charge of properly customising all the flexible components of the fabric, implementing a specific functionality. The multiplexing in time of the resources can lead to area and power saving with respect to a system involving all the necessary PEs without reconfiguration. Each possible configuration of the system is also called execution context. Execution contexts are changed, more or less frequently, during a single program execution. Architectures can support single, partial or multiple execution contexts, depending on the number of configurations that can be stored at the same time on the device (thus speeding-up the context change) and the area interested within the reconfiguration process.

As emerged by Section 2.1.1 dealing with all these aspects related to the design and management of reconfigurable systems, and in particular of CG ones, is often too hard to be done by hand. Apart for very simple cases, in the real world the design complexity requires the adoption of proper automated tools that support and facilitate the designer activity.

Design Automation

In order to facilitate the design and management of reconfigurable systems eventually within a whole architecture or architectural template, a dedicated software design environment is usually provided. These kind of tools are able to aid the developer during all the design phases. In particular, depending on the level of aiding, they provide two alternatives with respect to fully manual design flows: fully automatic solutions or automatic/manual compromises [38]. As can be seen by Figure 2.4, in the former case the designer has only to develop the desired application at an high level of abstraction (e.g. in C code), while the design environment automatically performs the hardware/software partitioning, in order to decide which functionalities have to be executed by the reconfigurable accelerator and which ones by the host processor, the High Level Synthesis (HLS), that is the generation of the hardware description (e.g. in RTL code or hardware netlist) starting from the high level source code, the mapping into the selected technology target and the management of the

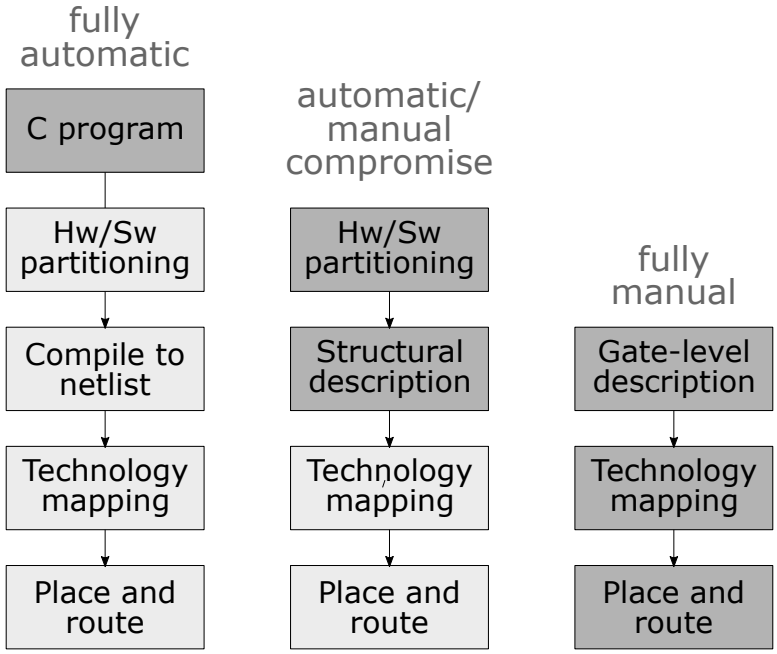


Figure 2.4: Different levels of design flow automation for reconfigurable systems.

contexts switching and interconnects rearrangement. In the latter case, the designer is required to perform the functionalities modelling, hardware/software partitioning and hardware description generation by hand, while the design environment automatically maps the functionalities within the reconfigurable architecture and manages the contexts changing and the system configuration.

In the practice, architecture Description Languages [64] have been extensively used in order to simplify and speed-up the development and deployment of embedded systems. Both in academia and industry, such languages have been exploited for automatic tool generation, processor verification and HLS [76].

HLS is an automated process that generates an hardware or software description of an high level system specification. As an example, Xronos [23] and Synflow [94] provide a Xilinx compliant and a target independent RTL description, respectively, both starting from different high level dataflow program specifications. Within the FPGA domain commercial tools, such as Vivado from Xilinx [97] and C2H Compiler from Altera [11], provide HLS capabilities offering C based IP generation. These tools target fine grained bit-level reconfiguration and generally do not allow the automatic composition of different disjointed specifications. Therefore, multi-functional scenarios are hardly managed, especially in terms of mapping [60]. Similar issues, not limited to the FPGA scenario, apply to the works of Schreiber et al. [81] and Bond et al. [27] that, starting from C/C++ specifications, present different methodologies to generate non reconfigurable co-processing units. Flexible and multi-functional scenarios have been addressed by Rutten et al. [79] where common functions, considered as *computational kernels*, are identified by exploiting the dataflow formalism. Kernels are successively mapped into different non reconfigurable hardware accelerators, alternatively invoked when required by the current execution.

In [42] high level structural languages, compilation-based, library-based and skeleton-based approaches are compared. With high level structural languages the designer

is required to write the hardware infrastructure, which may be extremely efficient at the cost of a lot of designer effort. Compilation-based approaches exploit HLS, which normally requires designers to manually iterate using directives, code transformations and vendor specific coding styles [97]. Library-based approaches have been designed to mitigate the limitation of compilation-based ones, since the vendor specific coding styles, directives and code transformations are pre-built into function libraries [67]. The drawback is that the number of functions in these domain specific libraries is limited and, normally, just one implementation is provided. In [42] a skeleton-based approach is adopted. The skeleton are templates to be used for exploration and implementation. According to the characteristics of the applications/kernels to be accelerated different skeletons have to be provided to optimally serve them. None of these approached considers the idea of exploiting CG reconfiguration to multi-functional scenarios. Basically, all of them implement FG accelerators.

The Datapath Merging Problem

Dealing with CG reconfigurable architectures, an interesting approach consists in building the reconfigurable substrate starting from the a set of isolated implementations of the wanted functionalities. The isolated implementations, also called datapaths, are brought at a high level of abstraction through the adoption of graphs in charge of describing the related functionalities. The graphs are merged in order to derive a reconfigurable datapath graph that can be synthesized in hardware according to a one-to-one mapping between graph nodes and hardware modules.

This is basically the approach that has been followed in the present thesis work, as discussed in Chapter 3. In particular, dataflow graphs have been adopted as high level specifications of the functionalities. Dataflows allows the expression of an application behaviour through a set of functional units, corresponding to the graph nodes, that interact through unidirectional, point-to-point, communication channels, that are the graph edges. More details about dataflows will be provided by Section 2.2.

The exploitation of a graph based formalism favours the resource re-use, both in terms of hardware modules and interconnects, since commonalities among the different functionalities can be easily identified. The best case for the obtianed reconfigurable hardware datapath is the one with the minimum area footprint, corresponding to the merged graph with the minimum amount of nodes and edges. Finding the so defined optimal reconfigurable datapath starting from a set of input functionalities described by graphs is known in literature as the datapath merging problem (DPM) and it has been demonstrated to be an NP-hard problem, so that it is not known if there is a solution able to efficiently (in a polynomial time) solve it.

Figure 2.5 can better explain the DPM through a graphical example of merging between two datapath graphs, G_1 and G_2, derived by the work [87]. G and G' are two possible merged graphs where the switching elements, allowing the sharing of resources in common between G_1 and G_2, are not represented for simplicity. Both G and G' map the nodes a_1 and a_5 of G_1 onto the same hardware modules where are mapped respectively b_1 and b_3 of G_2. In G, the b_2 node of G_2 is mapped in the same module than a_3 of G_1. Differently, in G', is a_2 that shares the hardware resource with b_2. The two possible merged graphs involve an equal number of hardware blocks,

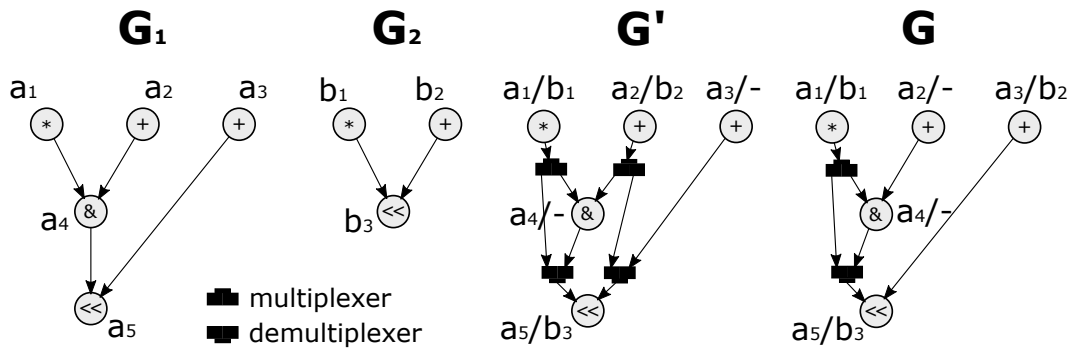


Figure 2.5: Example of the merging between two datapaths, G_1 and G_2, leading to two different reconfigurable solutions, G' and G.

but they have a different amount of edges. In particular, G has one less edge with respect to G', since the original edges linking a_3 with a_5 in G_1 and b_2 with b_3 in G_2 are overlapped (the related G edge is in bold). In the practice it means that, once implemented in hardware, the G' merging graph will involve two more switching elements than G, one for the outgoing edges of the a_2/b_2 node and one for the incoming edges of the a_5/b_3 node.

In literature, several heuristic methods have been proposed in order to solve the DPM. Moreano et al. [66] presented a technique where DPM is brought back to another problem known at the state of the art: the maximum clique problem. The authors introduced a detailed formalism for the graphs specification (a similar notation has been adopted for the present work as will be discussed in Chapter 3) and, on the basis of this formalism, they build up a *compatibility graph*. The compatibility graph is a graph whose nodes represent possible mappings between the edges of two merging graphs, according to a labelling function that, for a given graph node, returns the hardware block that is able to implement its functionality. The (undirected) edges indicate that the two linked mappings are compatible. Mappings are not compatible if and only if they map the same node of one merging graph to two different hardware modules. Solving the maximum clique problem for the compatibility graph means finding the maximum number of mappings that are compatible to each other, so that edge sharing among the merged graphs is maximised, resulting in a minimisation of the employed resources (including switching elements) within the final hardware architecture. The technique proposed by Moreano et al. considers two graphs at a time: at every iteration a new compatibility graph is composed and a new maximum clique problem is solved. Note that the maximum clique problem is an NP-complete problem, thus a polynomial time heuristic algorithm is adopted to solve it [18].

Another historically relevant algorithm used to solve the DPM is the bipartite matching heuristic method [52], that is based on the problem of finding a maximum weight matching in a bipartite graph. Also this heuristic method considers two datapath graphs at a time iteratively. The bipartite matching heuristic method is based on the construction of a bipartite undirected graph whose node set is obtained as the union of nodes belonging to the two merging graphs. The edges can connect one node of one merging graph to one node of the other merging graph, and link only the nodes that can be mapped in the same hardware module, according to a labelling

function similarly to the work of Moreano et al. [66]. A weight, expressing the benefits of the related mapping through a predefined cost function, is assigned to each edge of the bipartite undirected graph. The maximum weight matching of the bipartite undirected graph provides the node mappings that maximise the considered cost function.

De Souza et al. [87] compared the two presented DPM solving heuristic methods, exploiting also an integer programming formulation of DPM that allowed the computation of lower bounds for the considered DPM benchmarks. By means of the found lower bounds, the Moreano's heuristic method and the bipartite matching one have been evaluated. Moreano's method revealed to be the best algorithm for suboptimal solution of DPM for all the benchmarks at the price of a small computational time overhead with respect to bipartite matching. In the present thesis work a problem very similar to the DPM has been faced. As will be deeply discussed in Section 3, an empiric method has been adopted in order to solve the problem. The support for the best DPM solving method, that is Moreano's one so far, is still under development and will be integrate in the future.

2.2 The Dataflow Paradigm

The dataflow is a well known computing paradigm that has been firstly proposed in the early 1970s with the works of Dennis [39] and Kahn [45]. The work of Kahn [45] in particular introduced a distributed model of computation called Kahn Process Network (KPN). KPN consists in the execution of concurrent processes that communicate by means of unbounded unidirectional First-In-First-Out (FIFO) channels. Each process is executed sequentially, while the overall computing at the processes level is parallel. The Dataflow Process Networks (DPNs), firstly proposed by Lee et al. [63], constitute a special case of KPNs that describe programs through the interaction of logical entities, the *actors*, analogue to the KPN processes but providing conditions on which an actor is ready to execute (*firing rules*). The interactions among actors are specified by means of a direct graph where the nodes are the same actors, while the edges represent loss-less, order-preserving point-to-point connection channels (see Figure 2.6). The communication is based on the exchange of sequences of atomic data packets called *tokens*. The actors are abstract representations of PEs that encapsulate their own internal state and generate output tokens from the respective inputs, asynchronously concurring to the whole computation. Also the communication between actors is asynchronous, since it is driven by the production and consumption of tokens. Once triggered for processing (*fired*), actors execute a sequence of steps called *actions* that can result in:

- the consumption of one or more input tokens;
- the production of one or more output tokens;
- the change of the actor internal state.

Actors are implemented by any host language able to specify the actions firing rules. Modularity lets it possible to combine and make communicate actors described

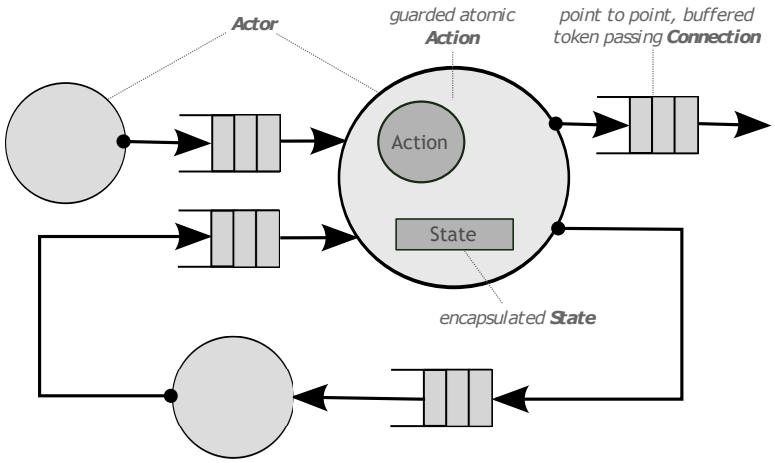


Figure 2.6: Dataflow Process Network design example.

through different specification languages such as Intellectual Properties (IPs) coded in HDL, low-level software actors written in C and high level software actors written in Java. Actors may be atomic actors or sub-graphs encapsulating in turn a dataflow network in a hierarchical fashion.

Such a kind of model is very suitable to manage the concurrency due to parallelism that one application may intrinsically have. Indeed, thanks to the token mediated communication policy, race conditions among actors are avoided. Furthermore, dataflows are highly modular specifications naturally amenable to block diagrams, therefore perfectly fitting to signal processing applications. Modularity strongly favours the code reuse, speeding-up the time to market needed for modelling updated versions of existent applications or new functionalities by scratch. For these distinctive features, the dataflow paradigm constitutes a valid alternative to the standard imperative programming model when concurrency occurs (e.g. in multi-core platforms). The imperative paradigm implements concurrent programs through threads, but leaves the onerous task of managing the concurrency among them to the programmer. In a dataflow program, concurrency depends on the token availability and each actor may fire independently without taking care of the other actors execution state. In this way, the program can be distributed over different PEs and the application parallelism is easily exploited. All these distinctive features make dataflows very suitable for programming highly parallel, also heterogeneous systems, like multi-processor SoCs or CG reconfigurable arrays.

2.2.1 Dataflow Models of Computation

In literature, starting from the presented generic dataflow model, several variations, also referred as dataflow Models of Computation (MoCs) have been proposed. A dataflow MoC defines a coordination language between actors. The differences between dataflow MoCs are related to the actors internal communication and actions scheduling. In particular the number of exchanged tokens can be fixed, variable, not specified or it can depend on parameters. Furthermore actors can have an external control flow.

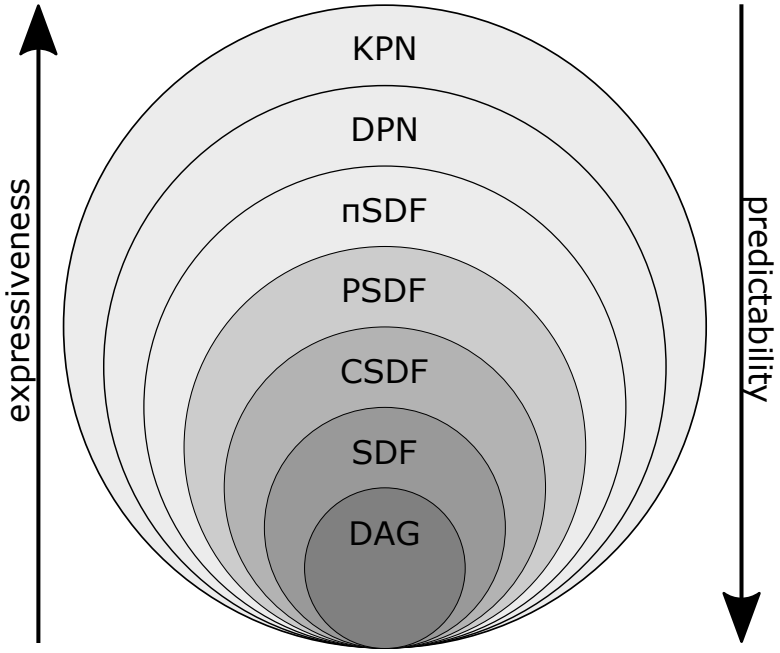


Figure 2.7: Venn diagram of the dataflow Models of Computation.

Figure 2.7 depicts an overview of the main dataflow MoCs. Each MoC represents a trade-off between expressiveness and predictability. Expressiveness quantifies how many different behaviour types the model can specify, how optimized and how concise is the provided specification. Predictability is related to the possibility of predicting the system behaviour, in terms of produced and consumed tokens, and it is characterised by the time it can be performed: at compile-time or at run-time. The most expressive dataflow MoCs are KPNs and DPNs, but they are also the less predictable ones. On the contrary the most predictable dataflow MoC is represented by the Directed Acyclic Graphs (DAGs), graphs with directed edges and without cycles. Nodes within DAGs are atomic operations while directed edges represent data dependencies between operations. DAGs can be extended to DAGs with periods and deadlines, where computation tasks are annotated with execution deadlines and periods. Moving towards more expressive dataflow MoCs there are Static DataFlow networks (SDF) [62], a special case of DPNs where actors have static firing rules: the number of tokens produced and consumed by each action is fixed. SDF MoC makes it possible to determine if the program can be scheduled at compile-time by means of a static code analysis. The static (compile-time) analysis is based on the checking of consistency and schedulability, that is the ability to come back to the initial FIFO states. The former is checked though the extraction and resolution of the topology equation, by the rank of the related topology matrix. The latter is checked by verifying that enough initial tokens have been set. The static firing analysis produces, if the SDF is consistent and schedulabile, a static schedule (a predefined sequence of actor firings) with a bounded memory usage and without deadlocks. Starting from SDFs, Cyclo-Static DataFlows (CSDFs) [26] have been derived by providing cyclo-static firing rules, thus achieving more expressiveness at the price of less predictability with respect to the same SDFs. In this case the number of tokens produced and consumed varies cyclically among a finite set of possible values (cyclic

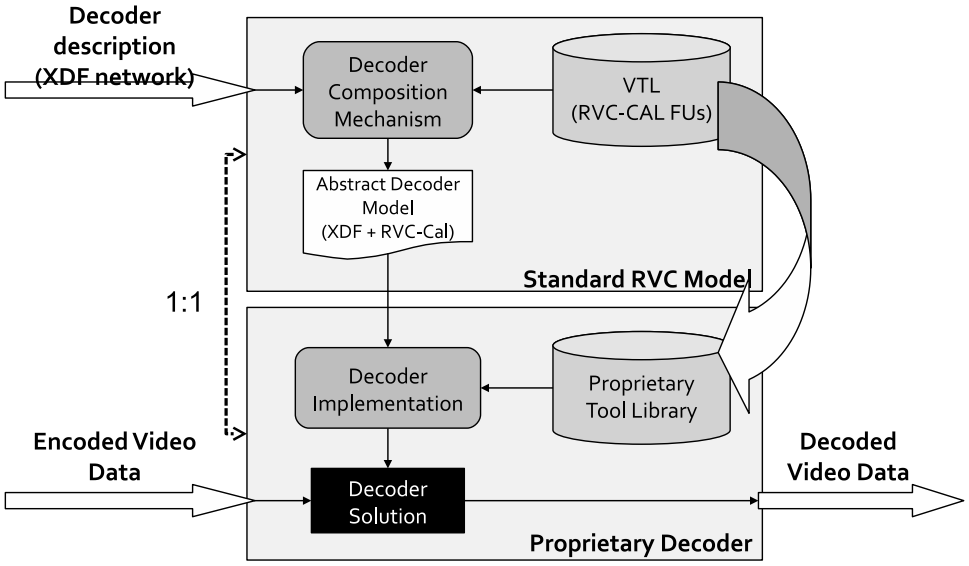


Figure 2.8: Overview of the MPEG Reconfigurable Video Coding (RVC) framework.

pattern). Like SDFs, CSDFs allow static analysis and scheduling. Parameterized Synchronous DataFlow (PSDF) is a further variant of SDF MoC that allows actor tokens production/consumption rates to be parameteric, while πSDF [40] introduces also interface based hierarchy [74] and dependences among different parameters in order to reconfigure the production and consumption token rates of actors at run-time.

2.2.2 MPEG-RVC Framework

Due to their distinctive features, dataflow MoCs are widely adopted in different and complex application fields. In the video coding scenario, the complexity of the developed systems is growing generation after generation, since increasingly complex algorithms are adopted. Newly codec versions often keep some parts already present on the previous solutions. For instance the Discrete Cosine Transform (DCT) is employed both within the MPEG-2 and the MPEG-4 codec specifications. In such a context, if already optimised solutions of the common resources can be easily exploited, the design process will be strongly speed-up, better coping with the typically short time to market of this application field. Pushed by this intent, the MPEG committee [25] has adopted a dataflow oriented language to describe video codec applications relying on the solid pillars of modularity, reconfigurability and, in turn, re-usability provided by the same dataflow MoC. Reconfigurable Video Coding (RVC) [5] allows designers to specify new codecs or different versions of already existent ones by providing a set of components, the Video Tool Library (VTL) [6], common to the most of video coding applications. Actually RVC constitutes a whole development framework (see Figure 2.8) on which designers can compose dataflow high level models of codecs that better match their application-specific constraints by adopting components belonging to VTL and/or new ones. Starting from the dataflow high level model, specific proprietary software, hardware or mixed implementations of the codecs can be derived.

The MPEG RVC involves several ISO/IEC standards that are based on a DPN MoC

(more precisely on a DPN variant that allows token peek) for modelling the applications. DPN has been adopted in video coding, as well as in other different multimedia areas, due to its high expressiveness and to the availability of a formal programming language, the Caltrop Actor Language (CAL) [41], supporting all its features. CAL has been originally developed and specified as a sub-project of the Ptolemy II project in the University of California at Berkeley. It is a C-like textual programming language developed in order to be easy to use through the adoption of a minimal semantic core. It is intended to be a domain-specific language, implementation independent and able to highlight the relevant design knowledge, such as parallelism. The language adopted by RVC, hereafter referred as RVC-CAL, is only a subset of the real CAL language and it has been normalized by ISO/IEC as a part of the RVC standards. The restrictions are aimed at simplifying the development of CAL synthesis tool for the support of either hardware and software target code. For the specification of the interconnections among the DPN actors, RVC adopted an XML dialect called XML Dataflow Format (XDF), that is part of the related standards too.

Around the MPEG RVC and the adopted CAL programming language, a wide variety of design tools have been developed in literature. Most of them leverage on the Open RVC-CAL Compiler [7], a compilation infrastructure in charge of generating descriptions in several languages (software, hardware or mixed for co-design [83]) starting from RVC-CAL actors and XDF networks. At the moment Orcc is provided as an Eclipse plugin written in Java and relies on an Intermediate Representation (IR) of the DPNs that is still specified in Java. The IR can be exploited to feed several other tools such as Turnus [37], which offers simulation, profiling and design space exploration capabilities, and Xronos [23], in charge of providing HLS for Xilinx FPGAs. The Orcc compilation infrastructure, as well as the MPEG-RVC framework itself, is continuously evolving in order to support more and more advanced features and to produce more and more dynamic systems.

2.2.3 Links with Reconfigurable Computing

Reconfigurability is one of the basis of the MPEG-RVC and, in general, it is naturally suitable to be coupled with dataflow MoCs for several purposes: mapping, optimisation and management. Considering the works analysed in Section 2.1, dataflows have been adopted several times. Huang et al. [51] adopted a dataflow based control in order to manage complex scheduling situations throughout the propagation of control tokens along with the data to be processed. Also Niedermeier et al. [68] exploited dataflow principles for controlling the flow of data and configuring PEs of their CG architecture. In particular they configure each PE by means of a finite state machine, whose stages are defined as dataflow actors with input and output token patterns. The ProDFA proposed by Yan et al. [101] is able to implement one different dataflow structure on its lowest grain PEs (the RPUs).

Dealing with the RVC-CAL formalism, the work of Beaumin et al. [19] proposed a reconfigurable co-processor in charge of executing different DPNs. The co-processor has been organised in different network units, each capable of computing one single DPN. Network units involve entities that implement CAL actors, the processing units, and the communication channels, provided as FIFO memories. Processing units can be configured for the execution of different actors at different times. It is possible

also to reconfigure the links between actors and, thus, between actors and FIFOs by means of a full mesh interconnect. During the evaluation of the system, authors found out the need of a FIFO controller in order to dynamically tailor the FIFO depths and reduce the resource demand that, otherwise, would be overestimated at design-time. FIFO sizing is one of the main and well known problems of dataflow based systems. In Chapter 3 it will be explained as one of the above mentioned tools of the MPEG-RVC community, Turnus, is able to perform FIFO optimal sizing for RVC-CAL dataflow networks by means of a design space exploration.

One other important work aiming at providing reconfigurability for RVC-CAL applications has been proposed by Gorin et al. [46]. Authors presented the Just-In-Time Adaptive Decoder Engine (JADE) that is a reconfigurable decoder capable of dynamically creating and configuring video decoders. JADE targets different platforms, such as X86, ARM and CELL, and can exploit multi-core architectures. It is based on a virtual machine and it requires as input one decoder configuration, that is the XDF specification of the wanted decoder, and a library of CAL actors, that can be the VTL. The flexibility provided by JADE for the support of, theoretically, all the decoders specified as RVC-CAL dataflow networks is paid in terms of performance, since it results lower than a manually optimised decoder.

2.3 The Power Issue

The huge diffusion of portability in embedded devices has strongly raised the importance of the power consumption constraints for the designers due to the battery life limits. The power issue has become so critical that it has been defined the "new timing constraint", since timing has historically been the most important constraint for the embedded system development. However the raising of power constraint, especially within specific application fields (e.g. biomedical image processing), does not relax the other main design requirements related to area and performance (that is tightly coupled with timing). In general, area minimisation leads also to power minimisation, so that meeting area and power constraints together may not be very hard. On the contrary, the performance maximisation is typically in contrast with the requirement of limiting power consumption and the realisation of a power aware execution efficient system could turn out in a real nightmare.

A lot of techniques have been proposed in literature to minimise the consumption in electronic devices. Generally speaking, they can act on the technology or on the behaviour of the system. The former techniques are focused on lowering the power dissipation equation contributes by acting on the involved technology quantities, such as the transistors voltage threshold (multi-threshold libraries), the power supply (multi-supply libraries) or the bias voltage (reversed biased channels). The latter relies on the system execution flow in order to avoid unnecessary dissipation of energy by slowing down not critical resources (multi-frequency) or by shutting off logic areas when unused (clock and power gating).

$$P_{tot} = P_{static} + P_{dynamic} \tag{2.1}$$

In digital systems, power consumption can be divided onto two main contributions: static and dynamic (see Equation 2.1). The former is always present since it

is due to leakage currents (non idealities of the transistors). The latter is dissipated only when logic transitions occur, so that it is related to the switching activity during the system execution. Dynamic power has always been several order of magnitude larger than the static one, so that designers focussed their power saving effort mainly on the dynamic contribution. However, in the last decade the transistor size is getting smaller, causing the static power to increase its weight in the power equation due to its own growth and to the contextual decrease of the dynamic power. This led power designers to take care, besides of the dynamic consumption, also of the static one.

Furthermore, technology scaling leads also to an increase of gate density on a single die, resulting in more complex systems. This aspect forces designers to have every day more global and high level consumption aware approaches across the whole design stages in order to keep chips cold [75]. Indeed, while limiting the consumption of a register transfer level (RTL) design by hand is tedious but possible, it is no longer doable for wider, complex systems, such as SoCs. The development of automated methodologies for power management is thus of paramount importance to assist designer in reducing the costs related to debug and deployment. In scuh a context, the previously mentioned power saving techniques can be still effective and, in some cases, they are already automatically implemented, at a low-level, by synthesizers. Cadence SoC Encounter and Synopsys Power Compiler support low-level power management with the automatic application of fine-grain clock gating, or by providing multi-threshold and/or multi-supply physical libraries. Nevertheless, this turned out not to be sufficient and model based solutions applied at higher-level [29] demonstrates the possibility of more efficiently manage the system consumption since the early steps of the design stack.

2.3.1 Static Power

The static contribution of the power consumption has only recently started to being strongly taken into consideration by the designers. This dissipated power is only due to the fact that the system is powered on. As previously said, the dissipation occurs since, within a CMOS device, transistors present non ideal leakage currents between different areas of the CMOS transistors. The static power can be quantified as the product between the overall leakage current of the device, $I_{leakage}$, and the supply voltage, V_{supply}.

$$P_{static} = I_{leakage} * V_{supply} \tag{2.2}$$

The overall leakage currents involves three main contributes: sub-threshold currents flowing when transistors are off, tunnelling currents throughout the gate insulator (especially for very thick channels) and leakage current due to reverse-biased diodes (between diffusion regions, wells and substrate). Depending on the adopted technology, each of these three terms becomes more or less relevant.

The main approaches that aim at reducing static power consumption act mainly on the supply voltage V_{supply} and on the transistors threshold voltage V_{th}. The reduction of the supply voltage clearly leads to a reduction of the whole static dissipation. In particular the main approaches acting on V_{supply} are the multi-supply voltage, that

partitions the device in different areas driven by different supply-voltages, and the power-shut off or power gating, that avoid unnecessary static consumption by shutting off the logic when unused. Note that, despite being beneficial under the power aspect, reducing the supply voltage makes transitions slower: at this purpose multi-supply voltage approaches typically keep higher the voltage of the logic involved on the system critical path.

While the impact of the V_{supply} in reducing static power is straightforward, since it is directly involved in Equation 2.2, the effects of acting on V_{th} are not so immediate. Generally speaking the reduction of V_{th} can be beneficial to the system performance, since transistors earn speed during their active state. However, static power has an inverse proportionality with respect to the threshold voltage: in terms of consumption, higher V_{th} are generally beneficial. As for the supply voltage, a common solution is to have different threshold voltages for different logic areas, basing on their criticality in terms of timing.

The implementation of power awareness systems that involve static consumption limiting techniques is usually not trivial. As a matter of fact, static saving approaches require strong resources overhead and dedicated support from the target technology library. For instance power gating employs three different kinds of dedicated cells (sleep transistors, isolation and state retention cells) that can be instantiated several times, depending on the design. While commercial tools provide the support for automatic implementation-level adoption of low overhead techniques (e.g. clock gating), for static power reduction strategies they do not provide automation facilitations. Designers have to plan and manage static power reduction techniques completely by hand. Nevertheless, these techniques have been already adopted in many commercial devices. The ARM big.LITTLE processor is an example of off-the-shelf product that involves two different sets of cores optimised for different purposes and that can be shut off in a mutually exclusive way [58].

2.3.2 Dynamic Power

The dynamic contribution of the power dissipation exclusively occurs when system nodes switch their state. It involves two main terms:

$$P_{dynamic} = P_{trans} + P_{cap} \tag{2.3}$$

where P_{trans} is the transient power consumption, due to the short cut currents between power supply and ground when the gate is changing its state. P_{cap} is the dissipation caused by the charging and discharging of the parasitic capacitances during logic transitions. Both P_{trans} and P_{cap} are directly proportional to the switching activity (total number of output switchings per gate), to the clock frequency and to the square of the voltage supply V_{supply}. All the techniques acting on this latter quantity, previously presented as beneficial for static power, can bring a positive impact also on the dynamic contribution. Anyway, the most approaches aiming at reducing the dynamic power act on the system clock. In particular the main trends are related to shutting off the clock signal at all or to provide multi-frequency environment. Shutting off the clock signal is (commonly known as clock gating) is one of the most adopted power saving solutions, since the clock net is responsible of more than

the 40% of the whole dissipation in digital systems [103]. Clock gating can be applied at different granularities: fine-grained clock gating works at the registers level (to avoid the consumption due to the refreshing of FFs and to the combinational logic that depend on their values); on the contrary, coarse-grained clock gating works at a higher-level by shutting down whole design units or groups of design units, making it possible to limit the dissipation of the clock tree. This last approach has been adopted during this thesis work and revealed its huge potentials also compared with the fine-grained automatic clock gating provided by commercial synthesizers.

The multi-frequency approaches push at the maximum frequency only the most critical resources (the resources with the longest execution latency) within the system, while they slow down the not critical ones. This strategy may lead to a solution with the same performance of the design where all the logic is driven by the same maximum frequency, but with limited consumption. Acting on the system clock requires typically a very low overhead, even though different components are needed according to the chosen targets. In ASIC designs the simplest solution is adopting AND gates directly on the clock signal to disable it for clock gating, while multi-frequency doesn't require any logic overhead, besides providing a different clock wire for each supported frequency. Dealing with FPGAs, it is not allowed to modify the clock tree with custom logic, but a fixed amount of dedicated blocks (BUFG in Xilinx devices), partially customizable for different purposes, is provided.

2.3.3 Links with Reconfigurable Computing and Dataflow MoC

The power issue is common to all the electronic devices, but considering reconfigurable architectures it requires a greater attention. Due to the different execution contexts, very often reconfigurable architectures involve redundant resources that are not always adopted in the computation. It is straightforward that the useless dissipation of power within these resources has to be avoided. In such a scenario, clock gating, either targeting FPGA or ASIC, has been revealed to be very suitable in order to reduce power consumption at run-time [20] [73].

The dataflow MoCs demonstrated to have distinctive features also for planning and managing power consumption. Within the MPEG RVC framework, some works tried to exlpoit these features in order to apply power saving techniques at a high level since the early stages of the design flow. Casale-Brunet et al. [33] proposed a methodology in charge of applying multi-frequency on a dataflow based design by analysing the actions dependencies of DPN actors and exploiting the related latency information. However it doesn't address reconfigurable systems. More recent studies [22] led to a coarse-grained clock gating implementation within a dataflow based design. The authors exploited the FIFO mediated communication of the dataflow network in order to selectively switch off actors that cannot execute due to the absence of enough input tokens in the incoming FIFOs or of enough available locations in the outgoing ones. Again, this approach doesn't support reconfigurable architecture. Despite that, both the works [33] and [22] are theoretically transversal to the reconfigurable paradigm and could be applied to such a kind of systems.

Ren et al. [78] adopted performance monitor counters in order to estimate energy consumption of RVC-CAL video codec specifications and consequentely plan a proper energy management strategy. The authors consider software solutions, but

basically the energy consumption estimation by means of monitor counters can be extended also to hardware platforms and tools. Obviously monitor based hardware estimation methodologies, such as the hardware performance monitoring infrastructure proposed by Schmidt et al. [80] have to be coupled with suitable power saving approaches in order to bring benefits on the system consumption.

2.4 Advancement with Respect to the State of the Art

In this thesis work it has been proposed an automated methodology in charge of designing, optimising and managing CG reconfigurable architectures. The targeted architectures do not belong to any fixed scheme structure presented in Section 2.1.1, but rather they are specific hardware substrates shaped around the set of functionalities that have to be implemented. The proposed methodology, by exploiting a dataflow formalism for the description of the wanted functionalities, aims at:

- overcoming the typical issues that affect the design flow of such a kind of systems, relieving the designer from the onerous mapping and configuration processes (it encapsulates the system run-time management, allowing fast, ideally in a single clock cycle, context switches by properly setting one unique device input);
- processing basically any dataflow MoC and addressing any device or technology, while allowing the adoption of any hardware PE, either developed by hand or automatically synthesized;
- pushing the system performance to its limits by exploiting the heterogeneous highly customizable topology of the substrate (instead of a fixed mesh) and the dataflow parallelism highlighting capabilities.

Around the baseline feature for the composition of reconfigurable datapaths, it has been developed a whole design suite of additional features that offer structural optimisation, automated power saving capabilities and quick integration in processor co-processor environments. Through the exploitation of a priori post-synthesis information related to the isolated desired functionalities, an automated optimisation methodology can drive the designer towards an optimal solution for the reconfigurable substrate, according to the selected design effort. This feature is intended to address scenarios where the design constraints are very strict. In this sense, thanks to the integration of the developed suite with the MPEG-RVC framework, besides the structural one, several levels of optimisations can be performed by other MPEG-RVC tools, pushing the resulting system to its limit.

Power issue is also tackled by providing a high level automated support that, coupled with the clock gating power saving technique (well known in literature), can lead to extreme benefits for the system consumption. Similar approaches, dealing with the clock gating automatic implementation have been proposed in literature. This technique is also available on the most common commercial synthesizers. The proposed dynamic power manager introduces a fully automated high level flow that

is distinctive for two main factors: it applies the clock gating at a coarse granularity (functional blocks level), making it possible to have more efficient savings if compared to fine grained approaches where clock gating is implemented at the gate level; moreover, it addresses reconfigurable designs and the MPEG-RVC framework, where the automation of power saving techniques has been only partially addressed.

Rapid prototyping of CG reconfigurable systems is possible through the co-processor generator, an additional feature in charge of automatically generate ready-to-use reconfigurable hardware accelerators. With respect to similar state of the art automatic flows for reconfigurable hardware accelerators, the one proposed in this thesis provides a fully automated methodology where the designer has only to model the wanted functionalities as dataflow networks. Furthermore, the presented flow is the unique that addresses CG architectures.

Despite being adoptable within any dataflow based scenario, this thesis work mainly focuses on the MPEG-RVC context. Before the current work, the MPEG-RVC application field:

1. lacked of a support for reconfigurable architectures;
2. involved profiling and design space exploration tools that did not exploit low level metrics, but rather functional related ones;
3. did not take care about the overall power issue, neither the single static and dynamic terms;
4. targeted typically multi-core platforms, while application specific ad hoc systems involving one or more dedicated hardware accelerators were never been considered.

All these new features introduced on the MPEG-RVC field are generally in line with the main challenges of the reconfigurable systems.

Chapter 3

Baseline Multi-Dataflow Composer Tool

In this chapter, the baseline functionality of the proposed suite for the design and development of coarse grained (CG) reconfigurable platforms will be described. The pillar of the proposed approach stands in the exploitation of the dataflow paradigm. Figure 3.1 depicts an overview of the straightforward coupling between dataflow and CG reconfigurable architectures. As discussed in Chapter 2, within a CG architecture a processing element (PE) can be of any granularity. Given a high level dataflow network, it is basically possible to derive the corresponding CG hardware architecture by simply mapping each actor to a different PE. If the aim is to develop a reconfigurable system, multiple functionalities have to be supported: once provided two or more dataflow networks that specify the wanted behaviours, it is possible to map them onto the same CG reconfigurable hardware architecture. Common actors between different dataflow networks will be implemented by the same shared hardware PE, while the execution correctness will be guaranteed by dedicated switching elements allowing the exploitation of shared PEs by all the functionalities that require them. For instance, the CG reconfigurable hardware platform obtained in Figure 3.1 involves one shared actor, C, accessed by both the incoming dataflow branches, respectively $A-B$ and $D-E$ through the switching element SB.

The situation described in Figure 3.1 is only a trivial example of the proposed approach. Real world is pervaded by complexity: bigger number of dataflow networks involving several actors constitute a typical scenario. In such a context, the generation and management of the CG reconfigurable platform is not longer doable by hand, but design automation is required. At this purpose it has been developed the Multi-Dataflow Composer (MDC), an automated tool in charge of providing a full support for the generation of CG reconfigurable systems from dataflow networks.

MDC aims at aiding developers in the non trivial issues related to the implementation of run-time reconfigurable architectures. These issues, described in detail in Chapter 2, are constantly growing by the increasing of platforms and applications complexity, resulting typically in huge time and resources demand. In order to address this challenging scenario, MDC exploits the dataflow models of computation (MoCs) and in particular the RVC-CAL formalism standardised by MPEG. Dataflow al-

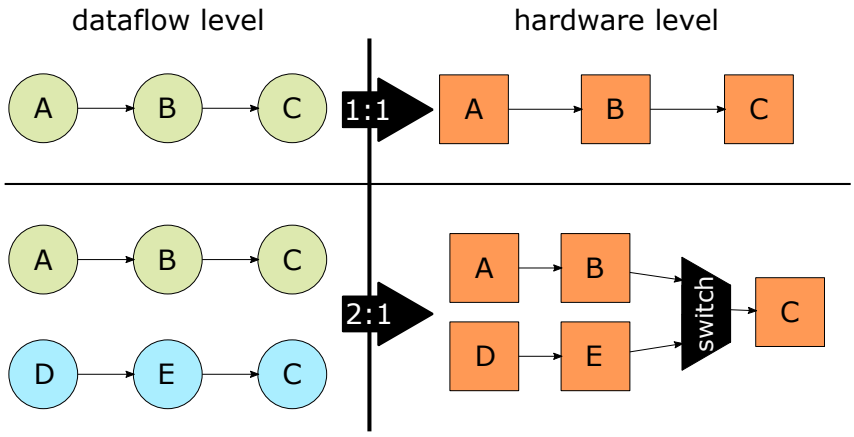


Figure 3.1: Combination of the dataflow model of computation and the coarse grained reconfigurable hardware design paradigm.

lows MDC to abstract the design process through a powerful, target independent and highly expressive model. The main capabilities of the proposed approach are:

- flexibility, ensured by the addressed CG reconfigurable architectures;
- high performance, achieved throughout the native specialisation capabilities of the dataflow modular approach;
- long term adaptivity, guaranteed by the re-usability of the specifications at the dataflow actor level.

The rest of the chapter is organised as follows: Section 3.1 will introduce MDC, the tool in charge of composing CG reconfigurable platforms from dataflow specifications. The discussion will firstly present the starting point of the work, that is a previous version of MDC. From this latter, during the thesis work a new version has been derived. The baseline functionality of the new MDC and the differences with respect to the previous version will be described in the same Section 3.1. Section 3.2 will expose some experimental results that demonstrate the effectiveness of the proposed approach and the differences between the two versions of the tool. Finally, Section 3.3 will conclude with some considerations about this chapter.

3.1 Multi-Dataflow Composer: Reconfigurable Platforms Assembling

The MDC has been conceived for the automatic generation of dataflow based CG reconfigurable systems [71]. It mainly focuses on hardware resources minimisation, inherently leading to area and power consumption savings. The work of this thesis started from a preliminary version of MDC, hereafter called MDC 1.0. MDC 1.0 is a standalone tool developed in C++ and adopting Bison, in charge of generating the input specifications parser, coupled with Flex, that creates the lexical analyser. It takes as input dataflow specifications belonging to the dataflow process network

(DPN) MoC and described in terms of Network Language (NL), an XML dialect standardised by MPEG and based on the Open Dataflow format.

Starting from a set of input NL dataflow specifications, MDC 1.0 combines them within a high level multi-dataflow specification and then it maps this latter onto a CG reconfigurable system, where each actor is implemented by a different PE. Resource sharing among the different input DPNs is maximised and fast reconfiguration of the supported functionalities is guaranteed by the introduction of low overhead switching modules, called switching boxes (Sboxes), that are simple combinatorial multiplexers and demultiplexers. Sboxes are placed at the crossroads between different paths of data and they are able to differentiate the data flows according to the wanted functionality [32].

Please notice that, despite being born within MPEG research (NL has been standardised within the RVC framework), MDC application should not be limited to the video codec domain. As demonstrated in [30] [31], MDC has been already successfully adopted to accelerate in hardware biomedical signal processing algorithms.

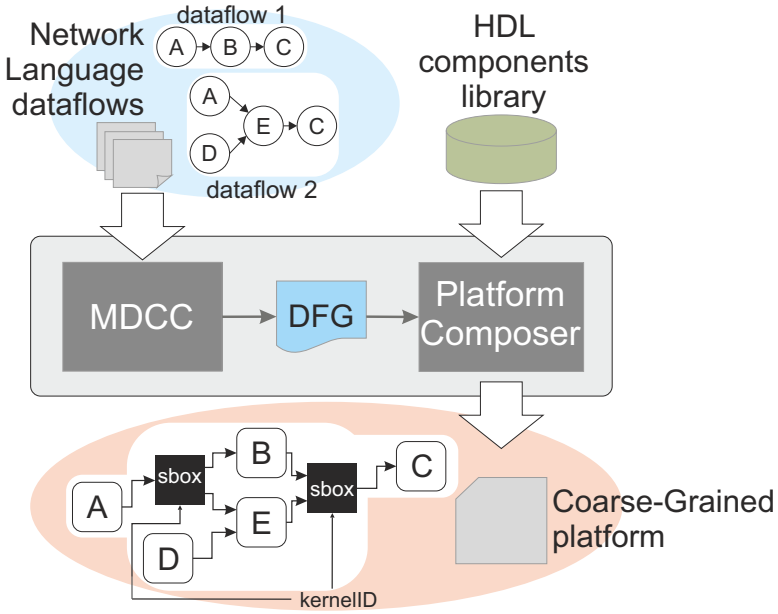


Figure 3.2: MDC 1.0: original tool flow.

An overview of MDC 1.0 is depicted in Figure 3.2: it is mainly composed of a front-end, the *Multi-Dataflow CAL Composer* (*MDCC*), and a back-end, the *Platform Composer*. The front-end is responsible of acquiring and combining the input DPN specifications: it identifies the resources that have to be shared among DPNs and consequently places Sboxes. The outcome of the front-end is a high level multi-dataflow specification, provided as a directed flow graph (*DFG*), that can provide all the input DPN functionalities taking care of the computational correctness subsistence. The back-end is in charge of mapping the high level multi-dataflow generated by the front-end onto a CG reconfigurable hardware platform. MDC is not able to perform high level synthesis from DPN to hardware modules. Hence an HDL library of components corresponding to the actors involved in the input DPNs is required by the back-end to generate a complete hardware system specification.

MDC 1.0 provides very fast, ideally single cycle, reconfiguration, resulting particularly suitable to real-time scenarios. Conceptually, yet the 1.0 version of MDC perfectly matches RVC-CAL compliant tools, like Orcc, Turnus and Xronos. At this purpose they could be integrated together in order to assemble an automatic tool chain for the creation of optimised CG reconfigurable devices. Despite that, the achievement of such a tool chain (that will be described in Chapter 4) required a re-design of MDC. Turnus and Xronos are already natively interfaced with Orcc, so that Orcc has been chosen as entry point also for MDC, in order to elaborate the dataflow networks. This required a re-design of the MDC front-end that made the tool coupled with the same Orcc. The coupling with Orcc, along with several others improvements that will be described in Section 3.1.1, led to the new version of the tool: MDC 2.0.

3.1.1 MDC 2.0

Besides allowing the integration within powerful tool chains, MDC updating and alignment with MPEG-RVC framework has several benefits for the tool. Due to the immediate interfacing of MDC with all the non normative tools belonging to the framework, input DPNs can be profiled and simulated in order to achieve early optimisations of the system and different high level synthesisers, from RVC-CAL to RTL, can be adopted to generate and customise the required components library.

The updating and alignment of MDC with MPEG-RVC framework required a re-design of the original MDC 1.0, resulting in the new 2.0 release of the tool. MDC 2.0 is not longer a standalone tool, as the previous version is, but it is coupled with Orcc (it is now a Java Eclipse plugin) as depicted in Figure 3.3. More precisely, MDC exploits Orcc in order to parse the set of input dataflow specifications whose functionalities have to be implemented by the final CG reconfigurable platform. Thus, input specifications are RVC-CAL DPNs provided as CAL actors and XDF networks. MDC works at the level of the Orcc Intermediate Representation (IR): the Orcc front-end parses the input RVC-CAL DPNs and generates the corresponding IRs that are forwarded to MDC in order to feed its front-end. Despite adopting a DPN MoC for being integrated within the automated tool chain, MDC 2.0 is theoretically able to process any type of dataflow MoC, depending on the front-end it is connected to.

Front-End

As for MDC 1.0, the tool front-end processes the input DPNs two at a time and produces a multi-dataflow high level specification able to implement all the input DPNs functionalities. The multi-dataflow specification is now provided through an Orcc IR (in Java), but MDC 2.0 can optionally generate a DPN compliant with the RVC-CAL formalism, so that it can be processed in turn by the MPEG-RVC tools, including the same Orcc. Within the multi-dataflow, common actors among different input DPNs are shared with the aim of minimising the employed resources, resulting in area and energy savings.

An IR dataflow network, and in general any DPN specification, can be seen as a DFG $G<V,E>$, where the vertices $v \in V$ represent input/output ports, simple actors or sub-networks, while the edges $e \in E$ are connections between two vertex of V ($E \subseteq V \times V$. Differently from input and output ports, vertices referencing an actor

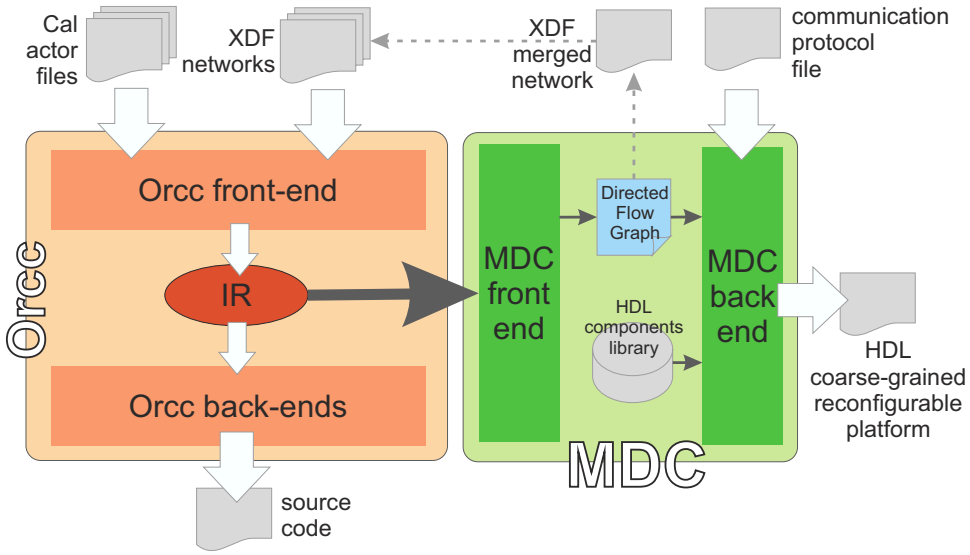


Figure 3.3: Overview of the 2.0 version of MDC.

or network are called *instances* of the actor or network and may present, in turn, input and output ports. Connections, besides being specified by a source and a destination vertex, provide source and destination ports that identify respectively the output port of the source vertex and the input port of the destination one. If one of the connection vertices is an input or output port, it is referenced as source or destination vertex and the source or destination port is unspecified.

Once the input dataflows are acquired and parsed into Orcc IR networks, the MDC front-end can start the elaboration. However a pre-processing step is required: the networks are flattened by replacing all the vertex instances that reference sub-networks with the respective referenced networks, so that the resulting input IR networks will be composed only by instances of atomic actors, besides input/output ports and connections.

At this point can finally begin the multi-dataflow network composition process, called *merging process*. It implements an empiric iterative algorithm (a detailed pseudocode is reported in Appendix A) in order to solve the problem of minimising the resulting dataflow resources, both in terms of actors and connections. This problem is known in literature as datapath merging problem, and it has been demonstrated that it is NP-hard (see Chapter 2). Each iteration involves two networks: the *merging network (MN)* $G_m < V_m, E_m >$ ($v_m \in V_m$ and $e_m \in E_m$ meaning an edge between two vertices in V_m), with the resulting one and the *result network (RN)* $G_r < V_r, E_r$ ($v_r \in V_r$ and $e_r \in E_r$ meaning an edge between two vertices in V_r). During the first iteration one of the input DPNs is taken as result network G_r. Each of the following iterations is composed by three main steps:

1. merging of the input and output ports;
2. merging of the actor instances;
3. merging of the connections.

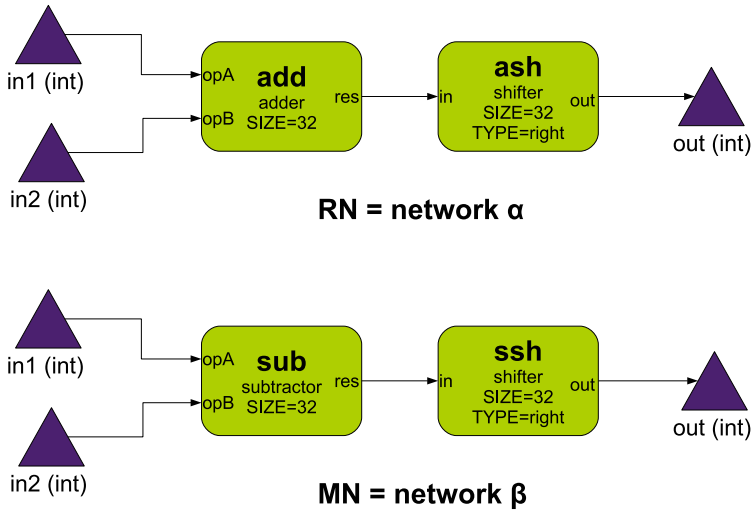


Figure 3.4: Step-by-step example: considered input dataflow networks.

All the steps of the merging algorithm leverage on a labelling function L of V ($L : V \rightarrow T$, where T is the set of labels representing hardware PE types) that expresses the compatibility among two network vertex, thus driving the mapping process. In particular, given two vertices $v_m \in V_m$ and $v_r \in V_r$, if $L(v_m) = L(v_r)$ then v_m and v_r can be mapped into the same shared PE. If the vertex is a port, the labelling function is related to two main aspects of the IR ports: the port name and typology (e.g. boolean, integer, float, etc.). If two ports have the same name and typology, then they have the same label. Dealing with actor instances, the labelling function considers the referenced actor, uniquely specified by a CAL file, and the parameters. Having two actor instances with the same label implies that they reference the same CAL file and that they have the same value for each IR parameter. Note that function labelling can be applied also to the ports of an actor instance (all the considerations made for the network I/O ports labelling apply) and to the entities involved in the connections (two connections have the same label if their source, source port, target and target ports have matching labels respectively).

A step by step example of the merging algorithm will be shown in the following by adopting two simple dataflow networks: α and β (see Figure 3.4). Both the networks perform an arithmetic operation on the tokens coming from the two input ports (it is a sum for the former and a subtraction for the latter); then the result is shifted and provided to the output port. α will be considered as merging network MN, while β represents the result network RN.

Merging of the Input and Output Ports The first and simplest step of the merging algorithm is the combination of the input and output ports of the two networks. During this step, the merging algorithm compares the label $L(v_m)$ of each I/O port belonging to the MN with the label $L(v_r)$ of the I/O ports of the RN: if there is one port of the RN whose labelling function leads to the same label of the considered MN port ($L(v_r) = L(v_m)$), then this latter is mapped onto the compatible shared port of the RN. Note that, due to the check on the port name performed by the port labelling function, it is possible to have one and one single matching RN port for each port of the

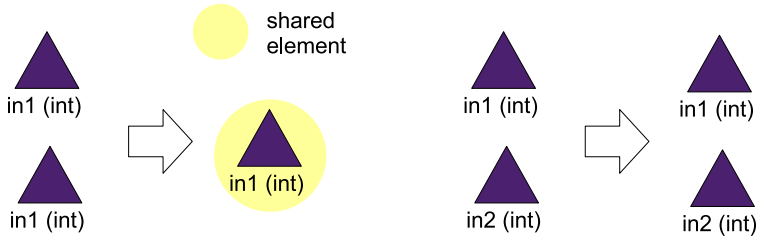


Figure 3.5: Example of the merging of input and output ports step: matching (left) and non matching (right) cases.

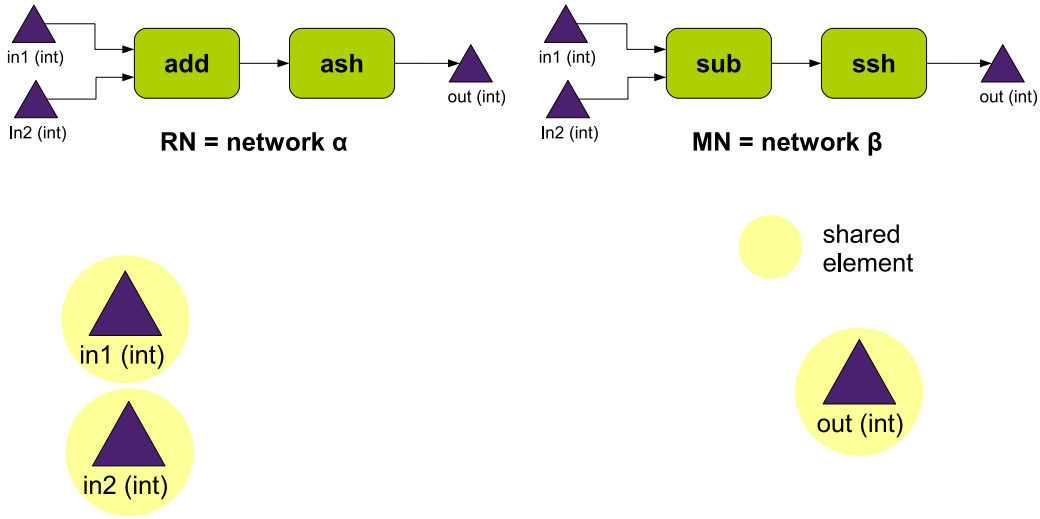


Figure 3.6: Step-by-step example: merging of input and output ports step.

MN network. Indeed, multiple ports with the same name are not allowed within an IR network. If a compatible port on the RN is not found, a new port with the same name and typology of the considered MN one is added to the RN.

Figure 3.5 illustrates two examples of the port merging step where port labels match (on the left), with a consequent port sharing, and where port labels do not match (on the right), requiring the insertion of a new port in the RN. Figure 3.6 shows the input and output ports merging step for the α and β networks considered for the step-by-step example. All the input and output ports of the MN find a compatible port on the RN: the only typology present is the integer one and each MN port (*in1*, *in2* and *out*) has its own corresponding port on the RN. The resulting RN involves only three I/O ports and all of them are shared between the two combined dataflows.

Merging of the Actor Instances The next step of the merging algorithm deals with the merging of actor instances. Due to the pre-processing flattening phase, if a vertex of the datflow network is an instance, it can only reference an atomic actor. Similarly to the merging of input and output ports, the label $L(v_m)$ of every instance v_m of the MN is compared with the label $L(v_r)$ of the RN instances v_r. In this case, since the actor instance labelling function considers only the referenced actor CAL file and the actor parameters, multiple matching labels are possible on the RN. If multiple matching labels are found, the MN instance v_m is mapped according to the

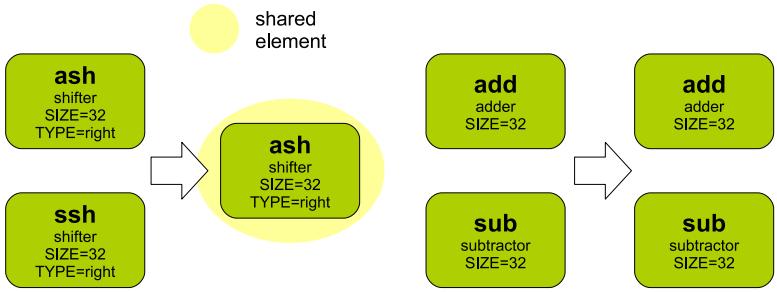


Figure 3.7: Example of the merging of actor instances step: matching (left) and non matching (right) cases.

number of matching connections: the RN matching label instance v_r that is involved in the maximum number of connections matching with v_m connections of the MN is selected for the mapping. As previously explained, the matching between two connections is given by the matching between the involved entities labels: the labels of the connections sources, source ports, targets and target ports have to match together. The adoption of the number of matching connections for the identification of the best shareable instance within the RN has an impact on the successive step of the merging algorithm. During the merging of the connections it guarantees the maximum sharing of connections among the two networks of the iteration and, consequently, the lowest number of Sbox insertions.

Again, if one actor instance v_m of the MN does not find a suitable matching instance within the RN, it is mapped onto a new actor instance of the RN that references the same actor and has the same parameter values of v_m. The evaluation of the matching label instances by considering the number of matching connections constitutes an enhancement of the merging algorithm introduced in MDC 2.0: the previous version of the algorithm mapped a MN actor instance onto the first RN matching one, without considering eventually multiple matchings.

Figure 3.7 provides an example of label matching for two actor instances (on the left) and an example of actor that does not find a shareable mapping instance (on the right), resulting in the placement of a new instance on the RN. Figure 3.8 resumes the step-by-step example: in this case, only the *ssh* MN actor instance finds a matching label within the RN instances (the *ash* references the same actor, the *shifter*, and has the same parameter values, 32 for *SIZE* and $right$ for *TYPE*). The labelling function gives one unique matching for this MN instance, thus the matching connections comparison is not required. The *ssh* instance of the MN is mapped onto *ash*. The other instance of the MN, *sub*, references the *subtractor* actor that is not referenced at all by the RN instances. The new actor instance *sub* is then added to the RN and, like *add*, it is not shared between the two dataflows.

Merging of the Connections The last and most complex step of the merging algorithm is the merging of the networks connections. It is actually the core of the merging algorithm. Basically, the structure is the same of the previous steps: for each connection e_m of the MN a matching connection is searched within the RN ones e_r, according to the labelling function. If a matching connection is found, it will be shared between the two considered networks, otherwise a new connection will be added in

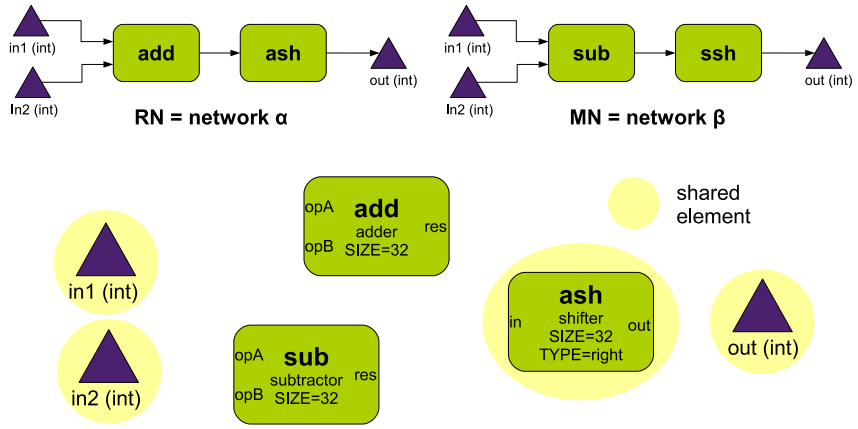


Figure 3.8: Step-by-step example: merging of actor instances step.

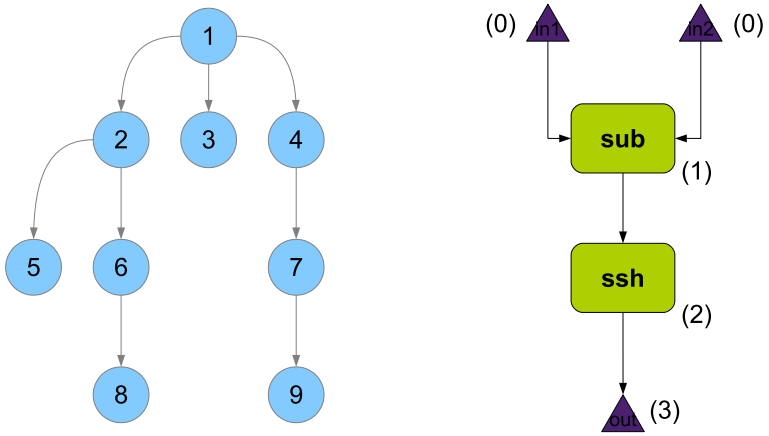


Figure 3.9: Breadth-First Search algorithm application on a graph (left) and on the step-by-step example merging network (right).

the RN. However, since connections represents links between two actor vertices, the definition of an analysis order is straightforward in order to avoid multiple mappings of the same connection or infinite iterations within closed loops. At this purpose, the last version of MDC considers a Breadth-First Search (BFS) algorithm [61] for the connection analysis. The BFS is a well known graph searching algorithm that performs a hierarchical exploration of the network nodes (see Figure 3.9) and achieves a good trade-off between depth and spread.

The merging of the connections explores at every step a different MN node and analyses its predecessors. The first step, that should explore the input ports of the dataflow, is not performed since the input ports have not predecessors. For this reason in Figure 3.9 the input ports of the MN for the step-by-step example (on the right) have an index equal to 0, meaning that the related step is not executed. According to the BSF order, the first MN analysed vertex will be *sub*, followed by *ssh* and lastly by the output port *out*.

For each analysed vertex and for each related predecessor, all the connections are considered for the merging process. The first phase of the connection merging is the candidacy and aims at the identification of all the connections (they may be more than one since each actor instance may have more than one I/O port) between

the vertex and the predecessor. For each identified connection, it is created a candidate connection between the two vertices of the RN which map the considered MN vertex and its predecessor. The candidate connection represents a connection that is candidate for being added to the RN. Resuming the step-by-step example dealt during this section, the candidate connections for all the MN vertices are:

- $in1 -- > sub.opA$ (between *sub* instance and *in1* predecessor);
- $in2 -- > sub.opB$ (between *sub* instance and *in2* predecessor);
- $sub.res -- > ash.in$ (between *ash* instance and *sub* predecessor);
- $ash.out -- > out$ (between *out* port and *ash* predecessor);

where the dot separates a network vertex (on the left) from its port (on the right) involved in the connection, if any, and the $-->$ separates the connection source (on the left) from the target (on the right). Prior to add the candidate connections to the RN, possible shareable connections already present on the RN are identified. If there is an existing RN connection e_r that matches with a candidate one e_m, according to the mapping of the involved MN vertices and to the labelling function (applied to the actor instance ports), the candidate connection is mapped on the existing one, that becomes shared among the respective dataflow networks. It is not possible to have more than one matching connection since the MN vertices have been previously mapped onto unique RN vertices and, if they are actor instances, the related ports must have different names. In the step-by-step example the only shareable connection is the one between the *shifter* instance (*ssh* and *ash* respectively for the MN and RN) and the output port *out*. This connection will not be inserted in the RN, while three more connections are not shareable at the end of this phase.

Only the candidate connections that have not been shared are added to the RN. Nevertheless, also this process implies some issues, since the vertices of the candidate connections (mapping the ones of the original MN connection) may be involved in connections already present within the RN network. When it occurs the Sboxes have to be inserted in order to make it possible the sharing of the vertex among different networks. For each not shareable candidate connection, all the RN connections are considered in order to identify possible collisions. A collision is found when the candidate connection and the considered RN connection have:

- the same source/target vertex, when the vertex is a port;
- the same source/target vertex and the same source/target port, when the vertex is an actor instance.

Depending on the fact that the collision occurs on the source or on the target side, a source collision or a target collision is found. One candidate connection can be in collision on both source and target sides. The merging algorithm, for each candidate connection, firstly identifies possible collision on the source side and lastly on the target side. For the considered step-by-step example, all the not shareable candidate connections have collisions. The first collision (see Figure 3.10) is a source collision occurring on the *in1* input port (it is connected to *add.opA* in the RN and to *sub.opA* in the MN). Similarly, a source collision is present on the *in2* input port (linked to

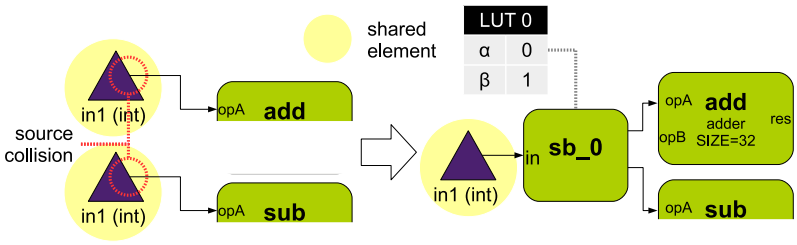


Figure 3.10: Step-by-step example: source collision on the input port *in1*.

add.opB in the RN and to *sub.opB* in the MN). The last collision is target side and involves the *in* port of the *shifter* instance, that has to be linked with the *adder* instance for the RN and with the *subtractor* instance for the MN.

If a source collision occurs, MDC inserts a Sbox with one input and two outputs within the RN. The Sbox is in charge of forwarding the input token to one of the two outputs, according to the currently wanted functionality. The information related to the forwarding is stored in a dedicated Look-Up Table (LUT). Along with the Sbox, MDC add to the RN a new connection that links the colliding source with the Sbox. The colliding connection already present on the RN will be updated by replacing the old colliding source with one of the Sbox outputs. Also the old candidate connection is updated by replacing the old colliding source with the Sbox output that has not be associated to the colliding connection of the RN.

At this point the target collision search can start. If a target collision is identified, MDC inserts a Sbox as well as for source collisions. The inserted Sbox allows the sharing of the target vertex and memorises the configuration information within a proper LUT. It has two inputs and one single output and can forward input tokens coming from one or the other input port to the output. A new connection that links the Sbox to the colliding target is placed into the RN. The collision connection that is already present on the RN is updated in order to target one of the Sbox inputs, while the candidate connection, once updated by targeting the other Sbox input, is added to the RN.

Dealing with the step-by-step example, shown in Figure 3.11, three different Sboxes are introduced, two one-input two-outputs Sboxes (two source collisions have been found) and one two-input one-output Sbox (one target collision has been found). Figure 3.11 depicts also the LUTs associated to the Sboxes. It can be verified as the functionalities of the original dataflow networks are effectively implemented by setting the related configuration values.

The iterative execution of all the three steps of the MDC merging algorithm leads to the composition of a multi-dataflow system self-containing the configuration information useful to implement the functionalities of all the input original dataflows. It is important to highlight that Sboxes are standard actor instances of the IR specification and that they reference the proper CAL file, provided by MDC as output. This implies that Sboxes can be managed like all the other entities of the IR, making it possible the adoption of the multi-dataflow by the Orcc back-ends and, if properly stored as standard XDF network, by the whole MPEG-RVC community.

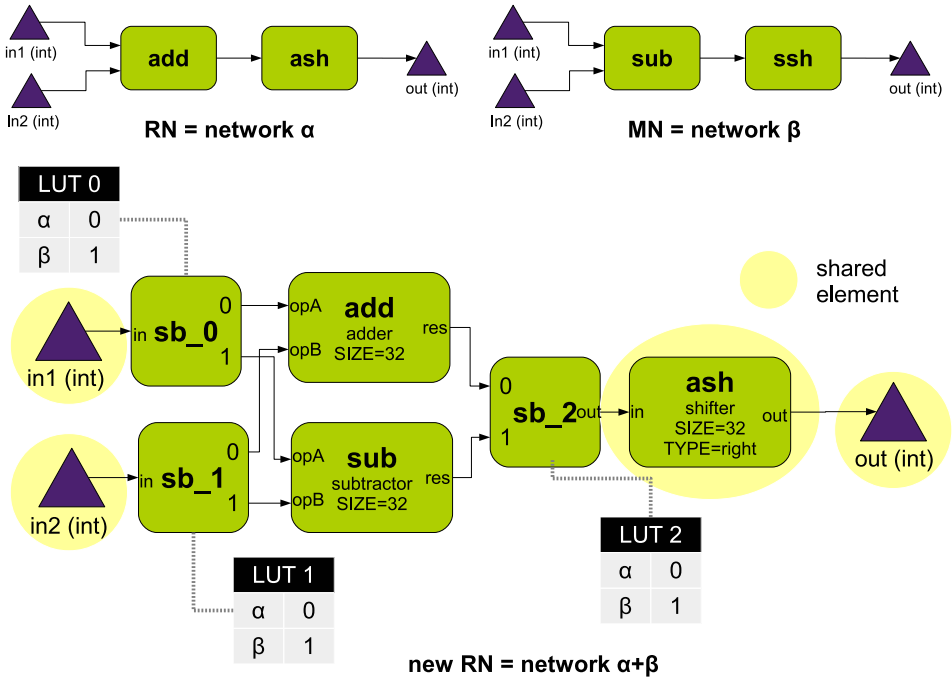


Figure 3.11: Step-by-step example: merging of the connections.

Back-End

The MDC 2.0 back-end has basically the same purpose of the MDC 1.0 one, apart from the different input description that is now an Orcc IR. The back-end still requires a components library containing the dataflow actors HDL implementation to map each of them in a different hardware PE. The components library can be created by hand or synthesised from the RVC-CAL input specification by adopting Xronos [23] or other high level synthesis tools. The new version of MDC also offers the possibility of customising the hardware communication protocol among the actors in order to embrace a wider set of components implementations.

The MDC back-end takes as input the multi-dataflow specification generated by the front-end to generate the corresponding CG reconfigurable RTL hardware description in terms of HDL code. Each dataflow actor is mapped onto a different hardware PE, whose hardware specification is provided within the components library. The Sboxes hardware PEs make the platform reconfigurable, since they allow modifying the interconnections at run-time. Their HDL code can be provided either by the components library or by MDC itself. The user can decide at design-time which hardware description will be adopted in the system.

One of the main improvements of the new version of the MDC back-end is related to the possibility of customising the hardware communication protocol. This latter has to be provided as input by means of a dedicated file, where all the signals involved in the handshake among two linked PEs have to be specified. Obviously, the HDL descriptions of the PEs provided within the components library have to be compliant with the given communication protocol. The Sboxes hardware description, independently of its source, has to be carefully generated since it guarantees the correctness of the protocol signals switching.

The inputs of the MDC back-end are then two: the multi-dataflow IR specification resulting by the front-end processing and the hardware communication protocol description. The outcomes of the back-end are two RTL files: the CG reconfigurable system top module and the LUTs module, that stores the Sboxes LUTs values and it is responsible of the system run-time configuration. The top module, besides describing the external interface of the system, instantiates all the PEs corresponding to the dataflow actors, including Sboxes, and specifies the connections between the same PEs and between the PEs and the external interface. Also the LUTs module is instantiated: it is in charge of properly setting the Sboxes selectors according to an input signal, *ID*, that uniquely identifies one of the implemented functionalities. As for the MDC front-end, in the following a step-by-step example (it is actually the same of the front-end discussion) will be presented in order to clarify the back-end behaviour. The starting multi-dataflow network considered for this purpose is then the one depicted in Figure 3.11.

As previously introduced, the customisation of the hardware communication protocol is one of the MDC 2.0 new features. The protocol is specified through a text file that describes two main kinds of signals: the global signals and the connections-related signals. The former are assigned once for each PE and usually are resets or PE enables. The clock signal is treated as a special global signal, since it is exploited for some MDC low power additional features, and has to be explicitly defined. The latter are associated to the IR connections, so that they are replicated on each link of the dataflow network. Furthermore, for both global and connection related signals, it is possible to specify if the signal has to be associated only to sequential PEs or also to the combinatorial ones. The combinatorial or sequential nature of a dataflow actor is specified straight on the CAL code. The clock signal is always associated only to the sequential PEs.

The protocol file syntax expects one different signal for each file line. The clock signal is indicated through a line containing its name (e.g. clk) followed by a comma separator and then the keyword indicating that it is the system clock ($clock$). Global signals lines are divided into three fields: the first one contains the signal name (e.g. rst), the second one the dimension in bits (e.g. 1) and the third one indicates whether or not it has to be associated only to sequential actors ($synch$ or not_synch respectively). The separator between the first two fields is a symbol that specifies the signal direction: $>$ indicates that the signal is an input for the PE, while $<$ indicates that it is an output. Differently, the separator between the second and the third field is again a comma. Connection related signals provide four fields within a line: the first defines the signal name on the source PE (e.g. $dout$), the second gives the size in bit of the channel (e.g. 32), the third specifies the signal name on the target PE (e.g. din) and the fourth indicates the kind of PEs (only sequential or both sequential and combinatorial) the signal has to be associated with, as well as for global signals. The separators among the first three fields of the connection related signals are the same $>$ or $<$ symbols adopted for the global signals. If the signal goes from the connection source to the connection target $>$ is adopted. Otherwise, if the signal goes in the reverse direction with respect to the dataflow connection, that is from target to source (e.g. for acknowledgement purposes), the adopted symbol for both the separations is $<$. The third field is separated by the fourth by means of a simple comma. Signals size in bit can be parametric and dependent on attributes of the IR

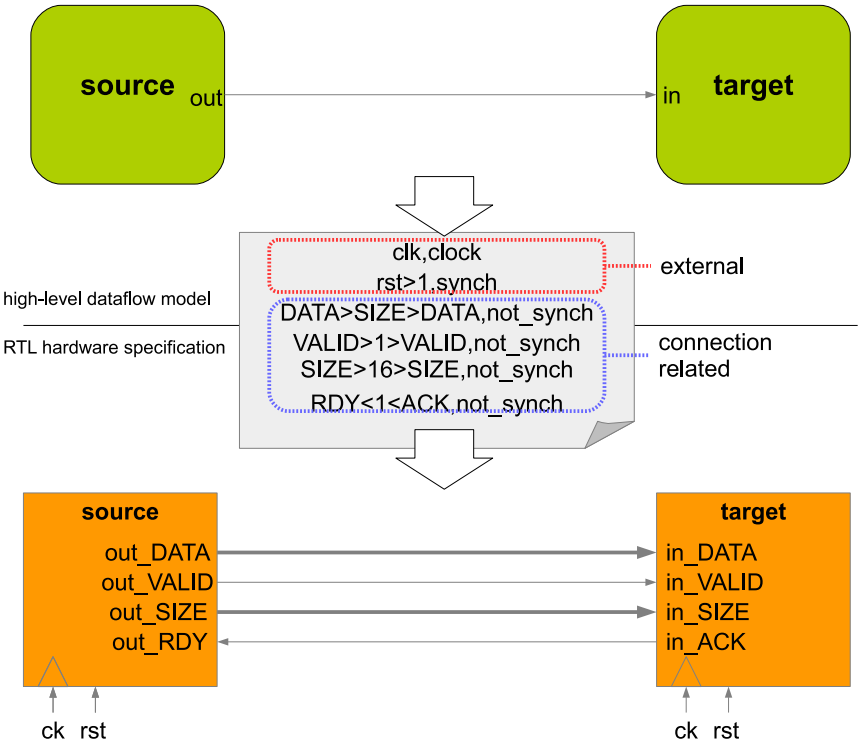


Figure 3.12: Application of a custom protocol during the MDC back-end hardware generation.

entities, like actors and connections, they are related to.

Figure 3.12 gives an example of real hardware communication protocol file and of its usage during the HDL code generation. The considered protocol provides, apart the special clock signal, one single global signal, rst, and four different connection related signals:

- $DATA > size > DATA, not_synch$, data channel with parametric dimension depending on the IR connection attribute *size*;
- $VALID > 1 > VALID, not_synch$, flag indicating when the signal on the data channel is valid;
- $SIZE > 16 > SIZE, not_synch$, signal that specifies the number of data sent during one single transaction (in successive clock cycles);
- $RDY > 1 > ACK, not_synch$, acknowledgement flag that communicates when the target PE can has received the previous data and can accept a new one.

For each IR actor or IR connection the back-end maps on the hardware system all the signals specified by the protocol for the corresponding PE or wire. As previously discussed, the protocol plays a central role also for the components library definition, since it determines inputs and outputs of the involved HDL modules.

Figure 3.13 shows an overview of the hardware platform generated starting from the multi-dataflow of Figure 3.11, related to the considered step-by-step example.

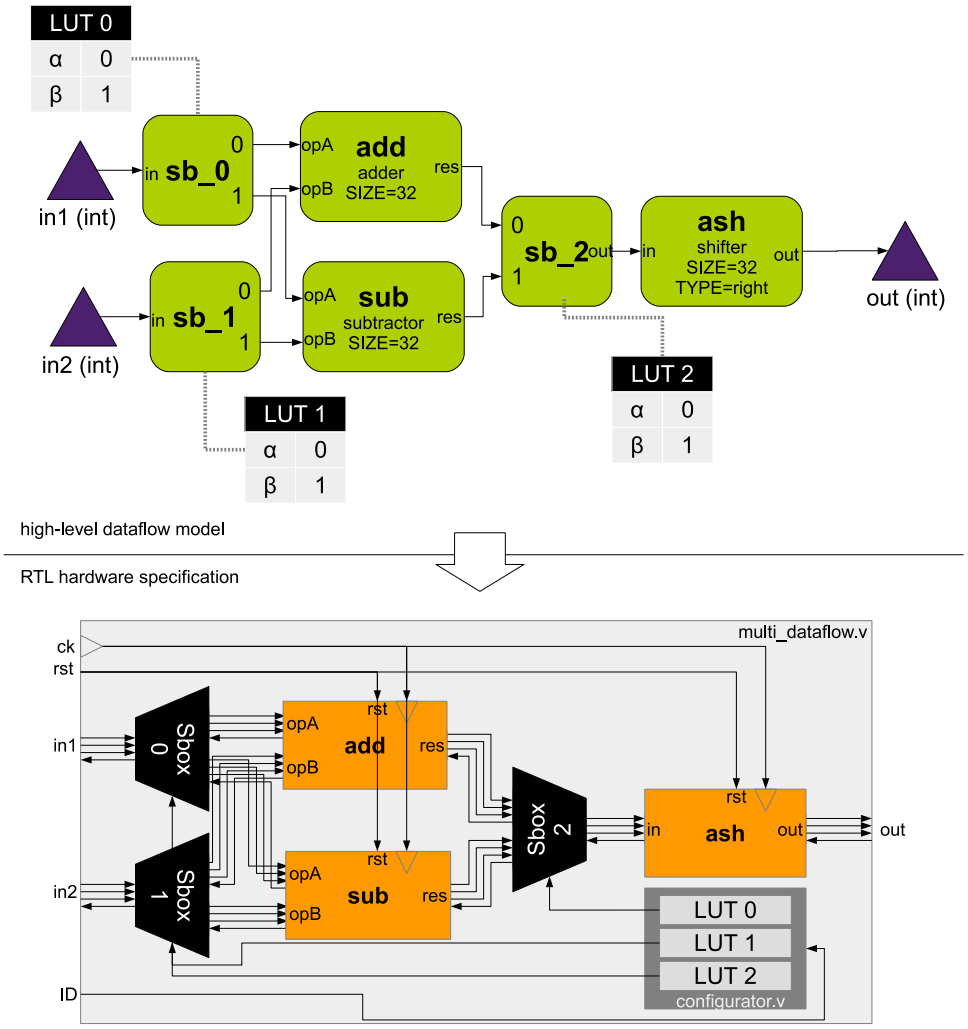


Figure 3.13: Step-by-step example: hardware platform generation step.

The adopted protocol is the one provided in Figure 3.12. For simplicity, the hardware connections belonging to the protocol are grouped and represented as bold arrows going from the source PE to the target ones. The global signals and the clock are associated only to the computational actors *add*, *sub* and *ash*, since they are the only sequential ones. On the contrary, Sboxes are combinatorial and involve only not_synch signals. The LUTs module groups three different LUTs that store configuration information related to SBoxes and that produce the corresponding selectors according to the given ID.

3.1.2 Summary of New MDC Features

The new version of MDC has been deeply described in this Section. In order to highlight the differences with the previous version of the tool and to group all the new supported features, here follows a brief summary of MDC 2.0 characteristics and differences with respect to MDC 1.0:

1. it relies on Orcc for the input dataflow specifications parsing;

2. it adopts the Orcc IR as dataflow formalism for the input functionalities specification;
3. it is compliant with XDF formalism, that has replaced the old NL format within the MPEG-RVC framework;
4. it embeds some optimisations in the merger algorithm, such as the BSF search order and the multiple matching connections for instances merging;
5. it has been generalised to deal with any actor communication protocol that is specified as input;
6. it still generates the CG reconfigurable HDL specification of the multi-functional system, but it produces also the corresponding RVC-CAL multi-dataflow that can be forwarded back to other MPEG-RVC tools;
7. it leverages on a graphical user interface, making it possible, besides the specification of the designer preferences, also the enabling of different extensions: (a) the structural profiler (see Chapter 5), (b) the dynamic power manager (see Chapter 6) and (c) the co-processor generator (see Chapter 7).

3.2 MDC New Version Assessment

This section is meant to assess the results achievable with the new version of the MDC tool. At first the targeted scenarios will be described (Section 3.2.1) and then the experimental data will be analysed and the related consideration will be derived (Section 3.2.2).

3.2.1 Designs Under Test

The targeted scenario is the same adopted in [71]. The main computing core of a reconfigurable hardware accelerator for image processing has been assembled, implementing in hardware several manually isolated processing kernels (one at a time). These kernels belong to four different applications in the mobile multi-media domain:

- Spatial Anti-Aliasing (UC1), to correct the distortion effects of an image down-sampled violating the Nyquist constraint;
- Zoom (UC2), to scale an image by a given zooming factor interlacing pixels of the original image with other pixels coming from adaptive interpolations of the neighbouring ones;
- Motion Estimation (UC3), to detect movements between two successive frames of a video sequence;
- Deblocking Filter (UC4), to remove image distortions coming from compression processes, like blocking and ringing.

Kernels were identified by profiling the given codes on a host processor. This means that the different applications have been executed on the host processor in order to extract the parts of the applications that are more computationally intensive, thanks to some dedicated software counters. The identified application computationally intensive parts correspond to the kernels. The aim is delegating them to a dedicated hardware block to accelerate their execution. It is important to note that, besides their execution cost, these kernels have been selected also having some operations in common. This is quite normal in the image/signal processing field, but it makes also more reasonable their mapping on a multi-functional hardware accelerator assembled with the MDC approach.

Twenty different kernels were identified. Their functions and composition are quite wide: from a simple one-actor kernel, finding minimum and maximum values of an array, to a seventeen-actors kernel, performing the RGB to YCrCb colour planes conversion of a 4×4 pixel image block. Figure 3.14 represents, through a Venn diagram, the commonalities among the different applications. It provides an idea of the actors overlapping, but for a more detailed overview of the applications composition please refer to [71].

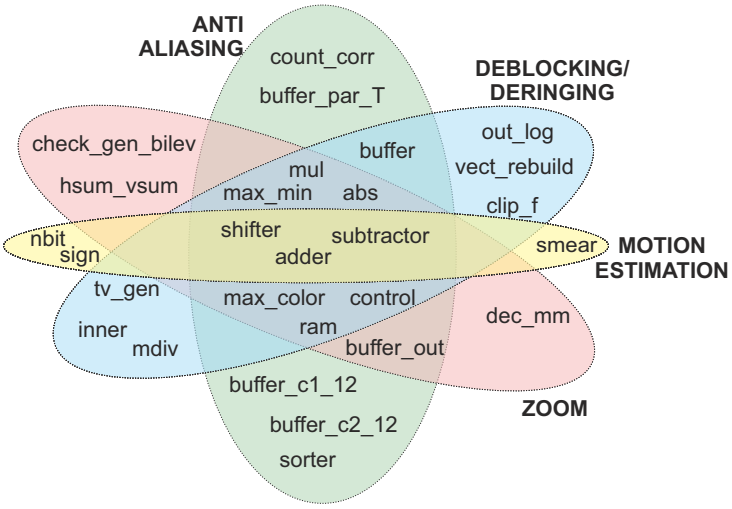


Figure 3.14: Actors composition and overlapping for the given application domains.

Five different scenarios are going to be assessed: the four above mentioned applications stand-alone (i.e. the co-processor will switch among the kernels of just one specific application, UC1-UC4) and a multi-application case (i.e. the co-processor will switch among the ensemble of kernels of all the provided applications, UC5).

Different designs are going to be assembled and compared:

- *static* - the co-processor is assembled without the adoption of the MDC approach (all the kernels are placed in parallel in the same platform without any resources sharing).
- *MDC 1.0* - the co-processor is assembled with the adoption of the MDC approach by adopting the old version of the tool, MDC 1.0.
- *MDC 2.0* - the co-processor is assembled with the adoption of the MDC approach by adopting the new version of the tool, MDC 1.0.

3.2.2 Experimental Data

This section is intended to assess the designs under test described in the previous section. In particular, synthesising the reconfigurable computing core on ASIC, the results achievable with the two baseline versions of the tool (MDC 1.0 versus MDC 2.0 disabling the low power extension) will be compared. ASIC synthesis trials have been performed using RTL Compiler (from the Cadence SoC Encounter commercial release), targeting a 90 nm CMOS technology. Figure 3.15a provides the results in terms of area occupancy. All the considerations in [71] still hold: comparing the *static* accelerator and the MDC based ones (*MDC 1.0* and *MDC 2.0*) large improvements are guaranteed by the MDC approach. *MDC 2.0* accelerators clearly show an effective minimisation of the area and power related to the PEs. As you can notice, *MDC 2.0* area results for UC1, UC2 and UC5 are slightly better. This is due to a reduced Sboxes count, reported in terms of area occupation in Table 3.1. This improvement has been achieved thanks to the adoption of the improved merging algorithm within the MDC 2.0 front-end.

	MDC 1.0		**MDC 2.0**		
App.	**Area**	**%**	**Area**	**%**	**2.0 vs 1.0**
	[μm^2]		[μm^2]		**[%]**
UC1	9054	6.5	7198	5.5	-20.5
UC2	13012	10.8	10476	8.8	-19.5
UC3	392	1.9	392	1.9	-0.0
UC4	12098	6.5	11902	6.4	-1.6
UC5	54352	15.7	44641	13.5	-17.9

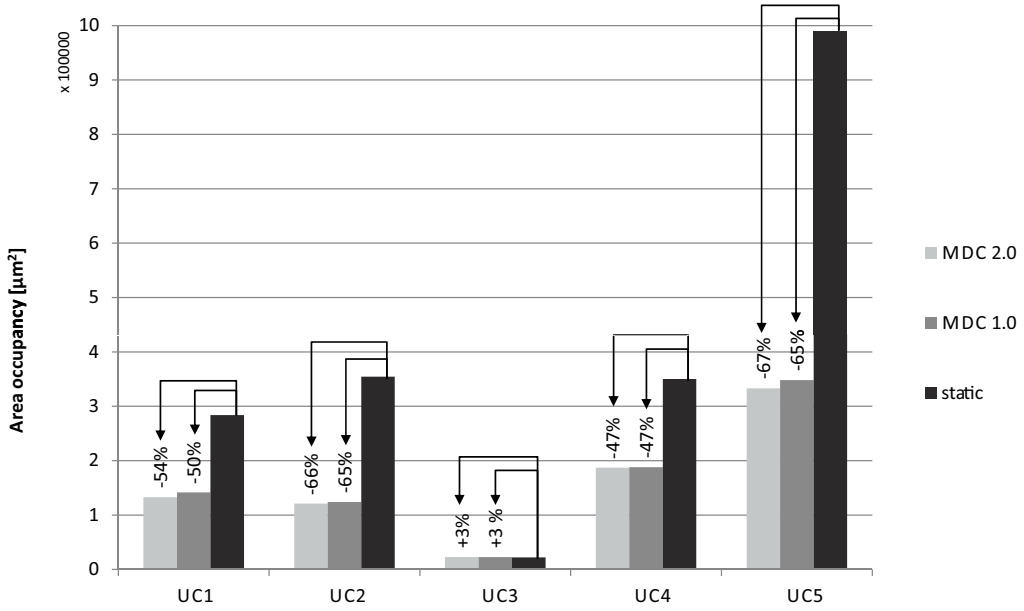
Table 3.1: MDC 1.0 versus MDC 2.0: Overall area occupancy of the Sbox (percentage with respect to the global area amount in MDC 1.0 and MDC 2.0). The *2.0 vs 1.0* column is the global area saving guaranteed by the MDC 2.0.

With respect to power consumption, Figure 3.15b reports the power consumption estimations provided by the Cadence RTL Compiler. These values are not accurate, not taking into account real execution conditions (default switching rates are adopted). Nevertheless, they are depicted in this form for coherency with respect to the work presented in [71]. Also the power consumption histograms show as *MDC 1.0* and *MDC 2.0* performance is comparable.

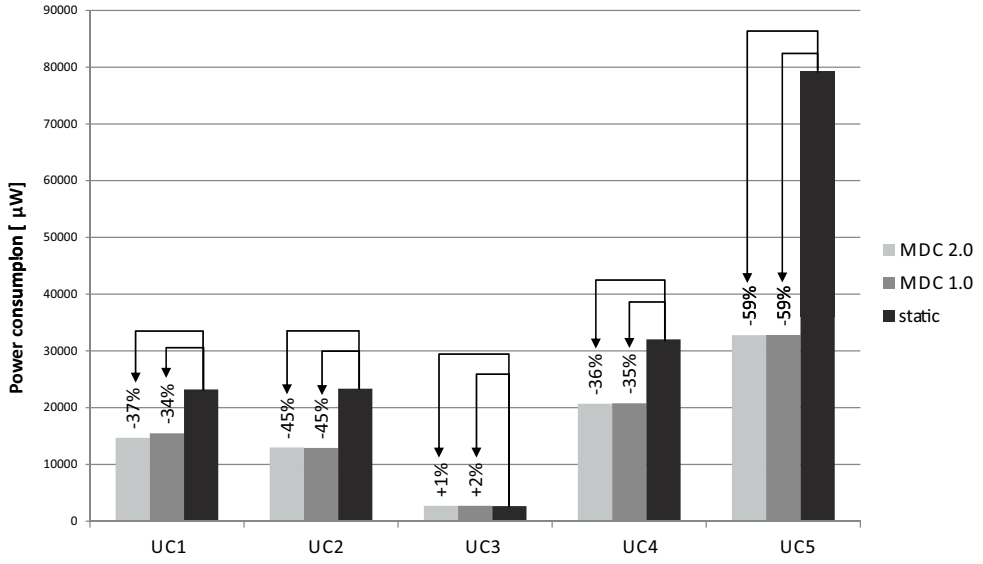
This first comparison has been carried out mainly for the sake of completeness. Its purpose was demonstrating that the new tool flow did not alter the original purposes and achievements of MDC.

3.3 Chapter Remarks

Nowadays, one of the challenges in complex systems development is to integrate flexible and efficient multi-functional systems with a little effort in terms of design costs, intended as re-engineering effort and debugging time. The combination of dataflow based modelling with CG reconfigurable hardware platforms already demonstrated its effectiveness in tackling such issues.



(a) Area Occupancy



(b) Dynamic Power Consumption

Figure 3.15: MDC 1.0 versus MDC 2.0: Assessment on ASIC technology.

With the baseline functionality of MDC, the complexity of implementing multi-functional reconfigurable systems has been challenged. Starting from an already existent preliminary automated tool, it has been re-engineered in order to optimise some weak aspects and provide full integration with the MPEG-RVC framework, resulting in a mutual fruitful influence with this latter. On the one hand, as it will be clearer by Chapter 4, it made it possible the development of a fully automated tool chain for the generation of CG reconfigurable systems. On the other, MPEG-RVC can

now rely on the first tool totally dedicated to reconfigurable architectures. Note that the baseline MDC feature alone is not a completely automated flow for the generation of CG reconfigurable systems, since it lacks of automation for important phases of the development like optimisation and hardware design.

It has been demonstrated as the re-engineering of the proposed instrument has kept the benefits of the previous release, bringing also some improvements under certain aspects. In general, multi-functional systems may potentially always gain by the adoption of the proposed design flow, especially when re-usability and design speed-up are required along with the need to meet stringent area and power constraints.

List of Publications Related to the Chapter

Journal papers

- C. Sau, N. Carta, L. Raffo and F. Palumbo, *Early Stage Automatic Strategy for Power-Aware Signal Processing Systems Design*, in Journal of Signal Processing Systems, Volume 82, Issue 3, March 2016, pp 311-329.

Chapter 4

Reconfigurable Systems Automated Development

The baseline functionality of the Multi-Dataflow Composer (MDC) tool only addresses the reconfigurable architectures composition and mapping. However, as deeply shown in Chapter 2, the compilation tool chain for such highly specialised and flexible platforms is one of the most critical parts of the development since it has to take care of different aspects, such as optimisation and hardware design. Without a full support for the whole design chain, the benefits guaranteed by the standalone MDC for the coarse grained (CG) architectures mapping and management could be limited, since designers have to spend a big effort on the non automated aspects.

At this purpose, this chapter presents an integrated optimisation, synthesis and mapping tool chain for CG reconfigurable systems that exploits several MPEG-RVC compliant tools, including MDC. The integration of MDC within a MPEG-RVC tool chain has been possible thanks to the new version of the tool, deeply discussed in Chapter 3. The whole automated tool chain has been assembled by integrating MDC with three tools of the MPEG-RVC framework: Orcc, Turnus and Xronos. Thanks to the tools integration, on the one hand an optimised application specific reconfigurable device is straight derived from the high level dataflow RVC-CAL models. On the other, the programmability of the device is automatically handled at run-time according to the required functionality, resulting fully transparent to the designer.

The rest of the Chapter is organised as follows: Section 4.1 will discuss the assembling of the automated tool chain for the generation of CG reconfigurable platforms that involves, besides MDC, several instruments belonging to the MPEG-RVC framework. Section 4.2 will expose some experimental results that demonstrate the benefits provided by the assembled automated tool chain considering, firstly, an MPEG-4 multi-decoder proof-of-concept and then a more realistic JPEG multi-quality encoding use case. To conclude, Section 4.3 will discuss some final considerations about the chapter.

4.1 Automatic Generation of Coarse Grained Reconfigurable Systems

This section describes the tool chain for the development of optimised CG reconfigurable platforms that has been assembled during this thesis work. All the involved tools are in-

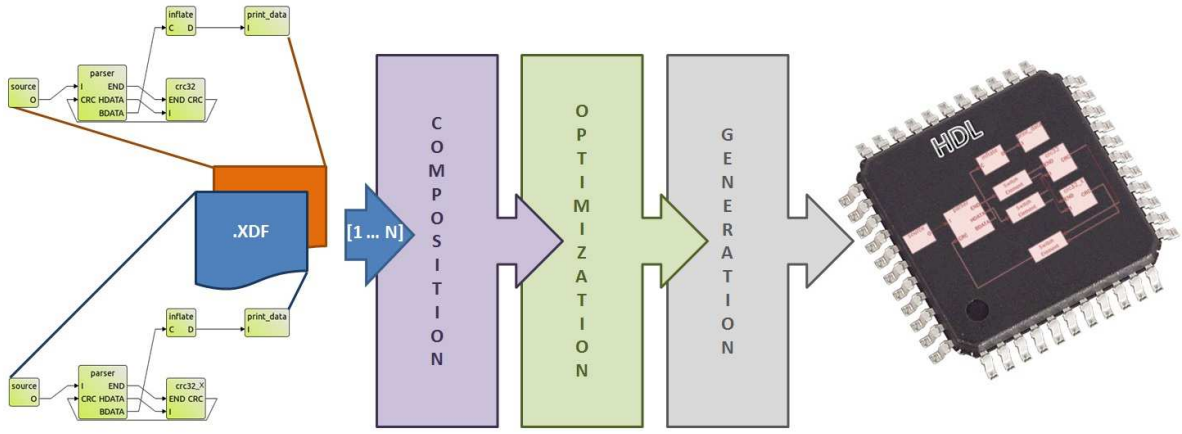


Figure 4.1: Logic Design Flow: high level phases

terfaced to automatically create a highly modular reconfigurable system implementation in charge of providing different functionalities. As explained in Chapter 2, the RVC-CAL formalism introduced by MPEG is supported by several non normative design tools. Very often they rely on Orcc, a compilation infrastructure that can traduce an RVC dataflow process network (DPN) specification, involving both CAL actors and XDF networks, onto the source code for a wide set of target platforms. In particular the tools that leverage on Orcc usually adopt its IR to describe dataflows. The integration among different tools which refer to the MPEG-RVC framework is necessary to challenge different and more complex scenarios than the ones addressable with standalone instruments. Furthermore, integration may lead to mutual fruitful influence among the different entities, resulting in their extensions, enhancements and new releases.

The logical overview of the overall tool chain for the development of CG reconfigurable systems is shown in Figure 4.1. The design flow is mainly composed of three different steps:

- composition - From a set of disjointed high level specifications (RVC-CAL DPNs) of the functionalities that have to be implemented by the platform, it is derived the corresponding high level multi-functional specification (still as an RVC-CAL DPN). This step is performed by the MDC (see Section 4.1.1), that will also be in charge of managing the programmability of the hardware substrate.
- optimisation - This step aims at optimising the system under the aspect of the employed resources by properly sizing the final platform. Turnus will perform the task, as will be described in Section 4.1.2.
- generation - Since MDC is not an high level synthesiser form RVC-CAL to HDL, a library of hardware components that implement the dataflow actors is required. At this purpose Xronos, an RVC-CAL high level synthesiser, presented in Section 4.1.3, has been included within the tool chain.

Lastly, MDC takes care of the overall system integration that will be explained in Section 4.1.4 by detailing the information exchange and interfacing among the involved tools.

4.1.1 Multi-Dataflow System Composition

MDC performs the first step of the automated tool chain for the generation of CG reconfigurable platforms. Its main job is combining an input set of dataflow specifications in order to

generate a unique multi-dataflow system that implements all the input dataflows functionalities.

The input DPNs are acquired by the Orcc front-end, core tool of the MPEG-RVC framework on which several other instruments rely. Once parsed by Orcc, DPNs are transformed into IR dataflows and processed by the MDC front-end. This latter is responsible of the multi-dataflow assembling as well as of the configuration information gathering and storing, as detailed in Chapter 3. The IR of the resulting multi-dataflow is forwarded to the MDC back-end in order to generate the corresponding RTL description. At the same time the multi-dataflow is provided in output as a standard RVC-CAL specification, so that it can be processed by the other entities of the tool chain. The hardware platform generation requires, apart the multi-dataflow IR, also the library of hardware components implementing the dataflow actors and the hardware communication protocol specified by the designer.

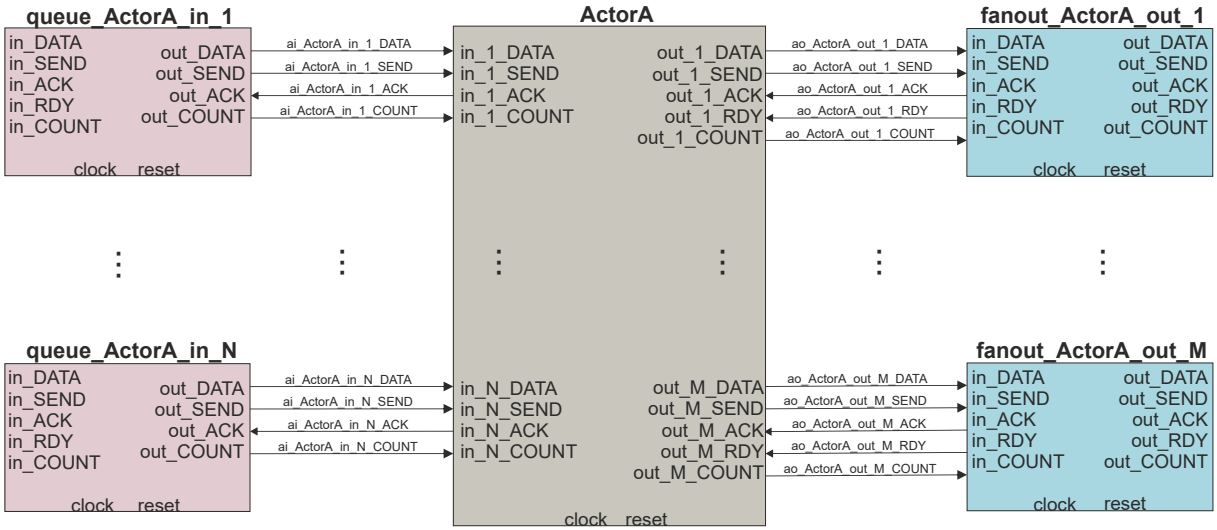
Within the automated tool chain, the components library is provided by the Xronos high level synthesiser (see Section 4.1.3) starting from the RVC-CAL multi-dataflow given by the MDC front-end. The hardware communication protocol, hereafter called RVC-CAL protocol, has been defined right by Xilinx for the MPEG-RVC framework and it physically supports the employment of FIFO memories for the transactions among actors PEs. It involves as global signals the special clock one along with an asynchronous reset. Five different connection-related signals are needed for the management of the FIFO based communication:

- $DATA > size > DATA, not_synch$, data channel whose depth is dependent on the $size$ parameter of the connections;
- $SEND > 1 > SEND, not_synch$, single-bit valid flag indicating the validity of the data channel signal;
- $ACK < 1 < ACK, not_synch$, single-bit acknowledgement flag indicating the token consumption by the target;
- $RDY < 1 < RDY, not_synch$, single-bit flag indicating that the target is available for the reception of new tokens;
- $SIZE > 16 > SIZE, not_synch$, 16-bits channel communicating the length of the token burst that has to be sent.

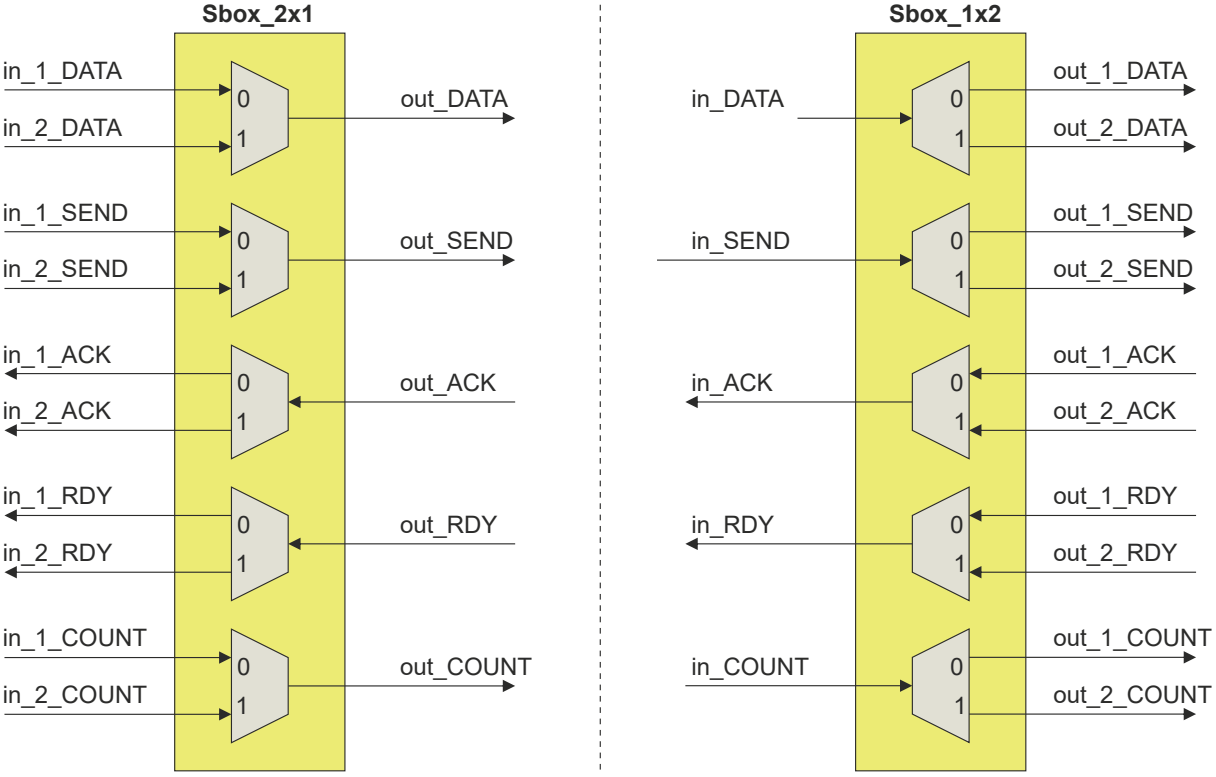
The RVC-CAL protocol is fully supported by the adopted RVC-CAL to HDL synthesiser, Xronos. According to the protocol and once synthesised in hardware, RVC-CAL actors are connected with a different *Queue* module for each input and with a different *Fanout* module for each output, as illustrated in Figure 4.2a. *Queues* are FIFO memories that manage asynchronous communication among actors while *Fanouts* are in charge of forwarding the tokens to the consumer actors.

Figure 4.2a depicts a standard producer actor, where all the interfaces are compliant with the RVC-CAL protocol. Please note that the switching modules introduced by MDC (Sboxes) do not need to be fully compliant with this protocol since, basically, they are not producer actors at all. For this reason, during the CG reconfigurable hardware platform generation, MDC avoids the overhead due to *Queues* and *Fanouts* for Sboxes. Fully combinatorial optimised Sboxes, as the ones shown in Figure 4.2b, are provided by the tool to be inserted in place of the high level synthesised ones.

Nevertheless, the FIFOs of the upstream/downstream actors have to be managed. One-input two-outputs Sbox units, inserted to split a path of data, require one FIFO for each outgoing connection. In the case of two-inputs one-output Sbox units, inserted to access a common shared actor, the FIFO buffers are placed before the Sbox along the incoming



(a) Automatic



(b) Custom

Figure 4.2: Different possible hardware implementations of the Sbox actor: (a) automatically synthesised, (b) custom fully combinatorial module.

connections. Since the Sboxes are fully combinatorial and the FIFO buffers always belong to the other actors, the well known dataflow problem of the FIFO buffers optimal sizing (faced in Section 4.1.2) does not affect the MDC merging process. Input DPNs have only to be properly sized before the MDC execution.

4.1.2 Dataflow Programs Optimisation

Once MDC composes the multi-functional dataflow network, implementing all the input dataflows behaviours, the system has to be optimised in order to achieve the efficiency both in terms of resource occupancy and performance. Within FIFO based dataflow systems, one of the main design issues is related to the proper dimensioning of the FIFO memories adopted throughout the communication channels [36]. FIFOs size obviously influences the resources occupancy, since it determines the amount of memory elements employed by the system interconnections. The FIFOs size impacts also on the system performance: when it is undersized, in the best case it may cause full conditions, thus resulting in additional delays; in the worst case stalls and deadlocks may arise, indefinitely freezing the computation.

Turnus [37] is the tool responsible for the multi-functional dataflow optimisation. It has been developed at the EPFL of Losanna and belongs to the MPEG-RVC framework. TRUNUS is a design space exploration framework for heterogeneous parallel systems based on the dataflow paradigm. It is able to simulate and analyse the system, providing performance estimation and optimisation features.

Turnus firstly performs a basic analysis consisting in a high level platform independent profiled simulation [56]. The outcome of this analysis step is an overview of structure and complexity of the considered system that will be exploited by the advanced features. The Turnus profiled simulation is based on the Orcc simulator: it is a mere Orcc IR interpreter adopted during the early design steps on the CAL applications for algorithmic and functional behaviour validation purposes. The profiling is performed during the simulation by retrieving information about the executed actions. For each of them it is kept trace of the computational load (operators and control statement) and of the data-transfers along with the related storage load (state variable and buffer usage, I/O port transitions and token production/consumption). Additionally, for each run of the profiled simulation, Turnus gathers information about the executed actions and the dependencies among them.

The outcome of such a profiling phase is a data describing the overall system behaviour, called execution *causation trace* [35]. The causation trace can be expressed in a graphical manner by a directed acyclic graph (DAG) where each node corresponds to an action firing, while each arc represents a dependency between two action firings (the arc goes from one action firing to one other that depends on the former). The causation trace nodes are nothing more than a partially ordered set of executed actions. Causation trace brings important information about the execution constraints and it has to be carefully composed since all the following analysis performed by Turnus are based exactly on the causation trace. Multidimensional design space explorations along with the identification of possible execution bottlenecks and unexploited parallelism all rely on the execution causation trace.

It has to be noticed that, generally speaking, for dynamic dataflow networks like the considered DPNs, the execution and, in turn, the causation trace are data dependent (depend on the specific input sequence) [24]. Nevertheless, considering signal processing applications, such as video coding ones, it is possible to adopt a set of input benchmark sequences (available for conformance test purposes) in order to provide a statistically meaningful causation trace [35]

FIFOs optimal sizing

The FIFO optimisation performed by Turnus is based on the post processing of the causation trace. During the post processing the nodes of the causation trace DAG are weighted with the *step computational load*. This weight is a latency value that takes into consideration the time overhead due to the scheduler selection process, the action execution time and the time

necessary to read and write the incoming and outgoing FIFO memories (including possible full cases). Each node, or better each action firing, of the causation trace is weighted with the time that elapses between the instant when it becomes schedulable to the one when all its output FIFOs have been written. The latency weight can be provided by several external tools. The developed tool chain exploits the weights generated by Xronos, corresponding to the number of clock cycles needed for the execution of actions functional parts.

By means of the causation trace weighing, it is possible to estimate the throughput performance of the system. The estimation is based on the identification of the execution trace critical path (CP), that always exists [47] and it is the longest weighted path of steps inside the execution causation trace directed flow graph (DFG). The causation trace CP constitutes a performance metric value and it is adopted by Turnus in order to optimally sizing the FIFOs. For the given dataflow system, CP allows the definition of upper and lower throughput bounds: upper bound (shortest CP) is found by supposing close-to-infinite FIFOs size configuration (write delays are neglected for the CP evaluation), while lower bound (longest CP) is found with a close-to-minimal size configuration, corresponding also to the minimal memory demanding solution. The FIFOs optimal sizing is achieved through a design space exploration (DSE) among all the possible trade-off solutions between the upper bound throughput (best performance) and the lower bound throughput (best resource utilisation).

Initially, Turnus has to evaluate the lower bound throughput solution that constitutes the starting point of the DSE. The lower bound is the minimum FIFOs size configuration that guarantees a deadlock-free scheduling. Dealing with the causation trace DFG, the lower bound corresponds to the nodes ordering that leads to the execution of all the DFG nodes with the minimum FIFOs size configuration. Considering a deadlock-free scheduling, Turnus identifies the lower bound by means of heuristic methods [34] that iteratively reduce the size of every individual FIFO of the dataflow network. Once the lower bound throughput has been identified, the aim is to increase the FIFOs size in order to shorten the resulting CP and make it closer to the upper bound one. In order to perform it, again Turnus adopts heuristic methods [36] based on iterative increase in the size of the FIFO responsible of the highest delay along the CP.

4.1.3 Hardware Platform Generation

The last phase of the developed automated tool chain for the generation of CG reconfigurable systems is the generation of the RTL code corresponding to the optimised multi-functional dataflow obtained through the earlier phases. Since MDC is not able to generate the hardware descriptions of the actors involved in the input dataflow specifications (hardware components library), a high level synthesiser from RVC-CAL to HDL is required to achieve fully automation. The Xronos framework has been identified as the entity in charge of performing this last task.

Xronos is a high level synthesis framework for dataflow or sequential applications derived by the some previous works [57]. It is easily integrable within other tools since it involves several interfacing points. Xronos provides an intermediate layer between the RVC-CAL high level language and the hardware design. It supports signed or unsigned integer and boolean basic data types as well as local and global variables that are readable or modifiable through different basic statement, such as variable assignment, port read and write, etc. Xronos allows also flow control statements, like branches, loops and parallel blocks, and procedural abstractions (sequences of statements) called *tasks*. Tasks rely on direct ports accesses and are allowed to be masters or slaves. The latter can be only called by the former and can not call other slave tasks in turn. A whole Xronos *program* is structured as a set of master tasks

that interact by means of FIFO memories.

4.1.4 The Automated Tool Chain

In this section the whole automated tool chain for the generation of dataflow based CG reconfigurable system will be presented. In particular the integration between MDC and the other tools will be detailed along with all the design steps of the proposed flow: composition, optimisation, generation. Figure 4.3 shows a block diagram of the developed tool chain, while a focus on tool integration with detailed iterations and file exchange is depicted in Figure 4.4. The integration, currently made up with a set of scripts, has been strongly favoured by the compliance of all the involved tools with the Orcc IR.

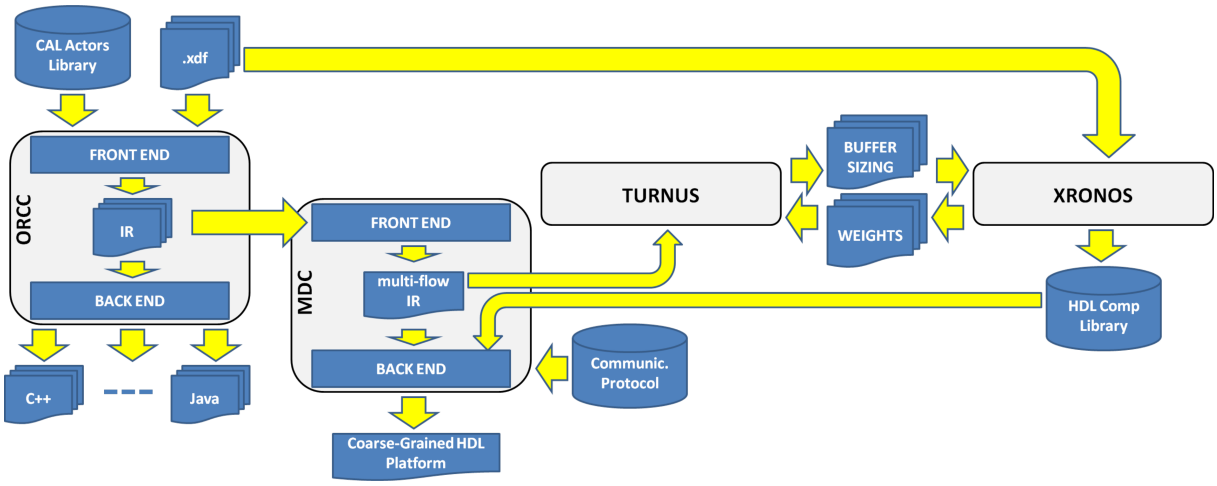


Figure 4.3: Overview of the integrated design flow.

The input of the overall tool chain is constituted by a set of DPN specifications, each one expressing a different functionality to be implemented by the CG reconfigurable system. The input dataflows are all conform to the RVC-CAL formalism, so that they are described through XDF networks and CAL actors. The first step is performed by Orcc that parses the input RVC-CAL specifications and translates them into its own IR. At this level, MDC combines the specifications and composes the reconfigurable multi-dataflow system involving the same actors of the input DPNs, eventually shared by means of Sboxes. MDC generates also an RVC-CAL specification of the multi-dataflow (proper CAL models for Sboxes are provided) that will feed the other tool chain entities.

In order to perform the optimisation step and find the best compromise between throughput and memory resources, Turnus needs two main things:

- the execution causation trace of the system for one possible simulation run (representative inputs are required);
- the related actions weights (the latency of the actions performed by each actor).

Both execution causation traces and action weights can be retrieved by post-processing the multi-functional dataflow specification assembled by MDC. The execution causation trace is calculated by the same Turnus through a platform independent high level profiled simulation of the multi-dataflow specification. On the contrary, action weights are strongly platform dependent and have to be evaluated during a hardware simulation. At this purpose, the multi-dataflow specification is preliminarily synthesised by Xronos. The generated hardware

multi-dataflow platform involves FIFOs with predefined fixed sizes. Since the system execution is data dependent, it is necessary to provide proper input token sequences for the hardware simulation. These sequences are the same adopted for the Turnus high level profiled simulation. They are retrieved from a software execution of the system: the multi-dataflow IR assembled by MDC is translated into the corresponding C code through the C back-end of Orcc. During execution, the C code stores input sequences information onto an ensemble of files, called *FIFO traces*, that are afterwards exploited through hardware simulation in order to consider the right data pattern for the multi-dataflow system.

The described flow is the baseline procedure to optimise dataflow based platforms through Turnus and Xronos. However the introduction of reconfigurable paradigm hardens the process: to obtain an exhaustive characterisation of the system, one couple of software and hardware simulations is required for each involved functionality. This result in a number of execution causation traces and action weights will that is equal to the number of input dataflow specifications.

Turnus exploits the action weights during the causation trace post analysis, aiming at the optimal FIFOs sizing. This configuration is then adopted by Xronos for the generation of the hardware platform. Having different causation traces and action weights for each supported functionality implies also performing a different causation trace analysis and obtaining different optimal FIFOs configurations. Here reconfigurability raises a new issue: shared actors may have different optimal FIFOs configurations depending on the considered functionality. Therefore, the resulting optimal FIFOs configurations are combined through a worst case analysis in order to identify a FIFOs configuration suitable for all the supported functionalities: keeping the worst case FIFOs dimension doesn't lead to underestimated sizes, thus avoiding stalls and deadlocks at run-time.

By means of the worst case analysis, starting from the action weights related to the different input DPNs, a unique set of action weights, providing a worst case size for the FIFOs incoming to shared actors, is derived. At this point Xronos can generate the final hardware platform. Both the Xronos generated Sboxes and the MDC combinatorial ones are supported. During the system prototyping, the designer can opt for the preferred solution by considering the achieved frequency/latency trade-offs: on the one hand, Xronos Sboxes require a memory overhead and can add latency for tokens delivery, since they are provided with incoming FIFOs; on the other, cascades of combinatorial Sboxes may negatively affect the system critical path, thus resulting in a maximum operating frequency drop (see Chapter 5).

4.2 Tool Chain Evaluation

This section aims at evaluating and validating the baseline MDC tool and the developed fully automated tool chain, that involves also other entities of the MPEG-RVC framework (Orcc, Turnus and Xronos). Two different use cases will be taken into consideration:

- an MPEG-4 Simple Profile multi-dataflow decoder, adopted as a proof of concept of the presented approach and showing the tool chain functionalities;
- a multi-quality JPEG encoder, enabling different quality factors for the compression and constituting a possible real application scenario for the proposed approach.

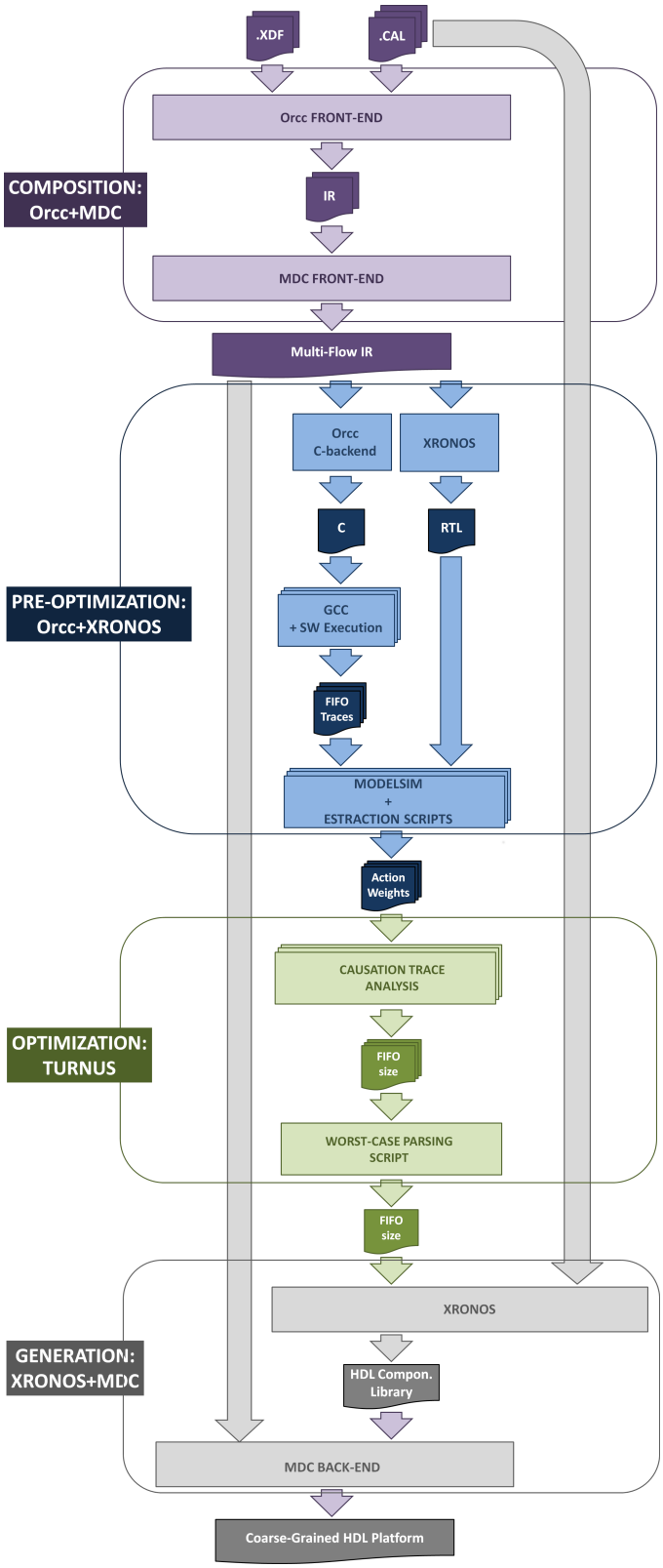


Figure 4.4: Files and information flow of the proposed design framework.

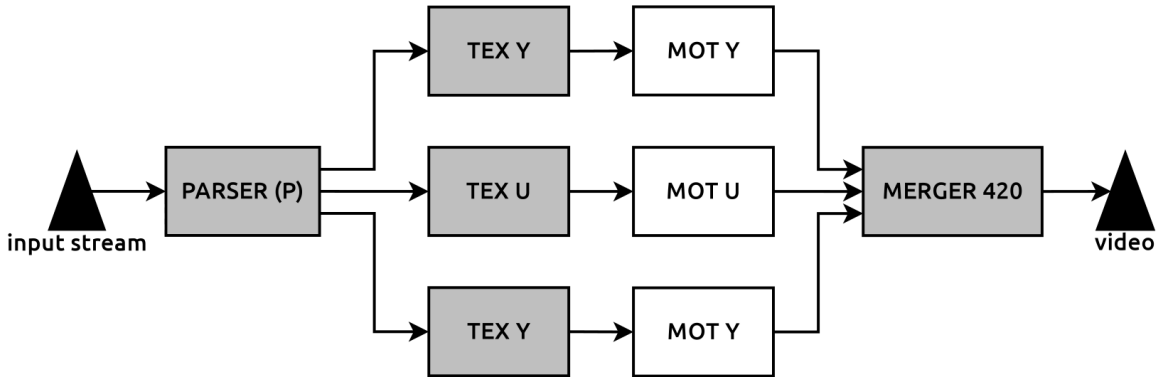


Figure 4.5: The RVC-CAL MPEG-4 SP decoder standardised by ISO/IEC JTC1/SC29/WG11. The grey part represents the Intra part of the MPEG-4 decoder.

4.2.1 Tool Chain Proof of Concept: MPEG-4 Multi-Dataflow Decoder

As a proof of concept of the proposed approach and in order to evaluate the developed automated tool chain, this section describes the experimental results obtained on a multi-dataflow decoder assembled by combining two versions of the same MPEG-4 Simple Profile (SP) decoder:

- the MPEG-4 SP Intra-only-decoder;
- the MPEG-4 SP Inter-decoder.

Both the decoder versions belong to the ISO/IEC standard (a.k.a. JTC1/SC29/WG11) and have been modelled as RVC-CAL dataflow specification. The decoding process provides three parallel channels, each one corresponding to a different colour space component of the YUV representation (Y for the luminance, U and V for the chrominance). The input format is 4:2:0, meaning that each macro-block (MB) of the image involves 16x16 Y pixels, 8x8 U pixels and 8x8 V pixels. MPEG-4 provides two main coding ways: intra and inter frames. An intra frame is coded basing only on its own information. The decoding of intra frames starts with a two dimensional Discrete Cosine Transform (DCT) that brings each MB in the frequency domain. Here they are quantised in order to highlight the lowest frequency contributes, that are the most sensible components for the human eye. Once quantised, the MBs are encoded by means of a Variable Length Coding (VLC) algorithm. An inter frame is coded by measuring the displacement of the frame objects with respect to one or more neighbouring frames of the video stream. Its coded form is the motion vector, a data structure that stores the displacement direction and entity.

The considered MPEG-4 SP Intra-only-decoder can decode exclusively intra frames of the coded video stream. The corresponding RVC-CAL dataflow specification (reported in Figure 4.5) involves in the input path a parser actor, that syntactically parses the incoming bit-stream and decodes it through the inverse VLC algorithm. The parser splits the stream into three parallel dataflow branches performing the texture (or residual decoding) separately on the YUV components. The texture consists of the sequential execution of AC-DC prediction (to foresee the quantisation coefficients form neighbouring frames), inverse scan (to properly scan the MBs), inverse quantisation (to equalise the frequency components) and two dimensional inverse DCT (to bring the MBs back to the spatial domain). The YUV image is then recomposed by a final merging actor. The MPEG-4 SP Inter-decoder basically has the same Intra-only-decoder dataflow topology but it involves motion estimation actors

placed in cascaded to the texture ones along the YUV branches, as depicted in Figure 4.5. The motion estimation actor is in charge of estimating the inter frames MBs by combining the neighbouring decoded frames and by decoding the motion vectors. Summarising, the considered Intra-only-decoder and Inter-decoder have in common the parsing, the texture and the merging actors but they differ for the motion estimation ones, that are present only within the Inter-decoder.

Experimental Assessment

In this section all the experimental data related to the MPEG-4 decoders proof of concept will be presented. In order to assess this use case, a Virtex 5 330 FPGA board by Xilinx has been adopted. The RVC-CAL dataflow models of the Intra-only-decoder and of the Inter-decoder have been combined to compose a multi-dataflow CG reconfigurable decoding platform. At this purpose the presented automated tool chain has been used and results validated the necessity of all the involved tools. Note that the considered scenario is not a real use case, since the MPEG SP Inter-decoder obviously requires as a base the Intra-only-decoder. Actually, it constitutes a proof of concept of the applicability of the assembled tool chain for the deployment of CG reconfigurable systems starting from standalone dataflow specifications of the wanted functionalities, preserving the performance of the isolated designs.

The designs under test, considered in the following for the flow assessment, are:

- *n-reconf*, a non reconfigurable design where the two decoders, generated by Xronos high level synthesiser from the respective RVC-CAL specifications, are placed on the same substrate without any resource sharing;
- *reconf*, a reconfigurable design automatically assembled through the presented tool chain (different version of this design are provided in order to better understand the importance and impact of every involved tool).

The composition of the *n-reconf* and *reconf* designs in terms of number of actors and Sboxes is shown in Table 4.1. Intra-only-decoder requires 32 actors, while Inter-decoder involves 40 actors. The resource sharing provided by MDC leads to the saving of 4 actors, since the Sboxes placement results in 68 overall actors for *reconf* design versus the 72 of the *n-reconf* one. Nevertheless, the proposed MDC approach turns out to be beneficial only if the computational actors, decreased from 72 to 46 units thanks to resource sharing, have a higher technology impact than Sboxes (see columns 256 of Table 4.2).

Table 4.1: Comparison of the actors composition among the implemented solutions: the baseline non reconfigurable one (*n-reconf*) and that implemented leveraging on the proposed flow (*reconf*).

	computational	**Sbox**	**overall**
n-reconf	72	0	72
reconf	46	22	68

In order to evaluate the benefits of the optimisation step performed by Turnus, initially the designs have been implemented bypassing this phase. In particular *n-reconf* has been synthesised by adopting a fixed size of 256 locations for all the involved FIFOs. As previously introduced, on the contrary the *reconf* has been synthesised by considering different setups:

Table 4.2: Sizing of the FIFOs in the *reconf opt* and *red* designs. The designs involve 100 additional FIFO buffers with size equal to 1 that are not reported here.

actor	port	buffer size
splitter_BTYPE	BTYPE	6
Mgnt_BlockExpand	RUN	41
Mgnt_BlockExpand	VALUE	41
Mgnt_BlockExpand	LAST	41
splitter_420_B	BTYPE	5
splitter_Qp	QP	6
-	WIDTH	64
-	HEIGHT	64
Algo_DCRinvpred_luma_16x16	QFS_DC	4
Algo_DCRinvpred_luma_16x16	QP	4
Algo_Add_nomotion_0	BTYPE	4
Algo_Add_nomotion_0	TEX	4
Algo_Add_nomotion_1	BTYPE	2
Algo_Add_nomotion_2	BTYPE	2
Algo_IS_ZigzagOrAlternateHoVel_8x8_0	QFS_AC	252
Algo_IS_ZigzagOrAlternateHoVel_8x8_0	AC_P_DIR	252
Algo_IDCT2D_ISOIEC_23002_1_0	IN	64
Algo_IDCT2D_ISOIEC_23002_1_0	SIGNED	2
Algo_IDCT2D_ISOIEC_23002_1_1	IN	64
Algo_IDCT2D_ISOIEC_23002_1_1	SIGNED	2
Algo_IS_ZigzagOrAlternateHoVel_8x8_1	QFS_AC	3
Algo_IDCT2D_ISOIEC_23002_1_2	IN	64
Algo_IDCT2D_ISOIEC_23002_1_2	SIGNED	2
-	VID	64
Mgnt_Merger420	Y	130

- initially, also for *reconf* Turnus optimisation has been bypassed and the design has been implemented with different FIFO sizes: 1, 16, 64 and 256;
- then, an optimised implementation *opt* of the reconfigurable system has been developed: in this case Turnus performed the FIFO sizing optimisation and Xronos synthesised the resulting CG hardware platform;
- finally, the impact of avoiding unnecessary *Queue* and *Fanout* modules instantiation for Sboxes has been evaluated through the *red* design: it is the result of the automated tool chain involving Turnus optimisation phase and allowing MDC to manage the Sboxes HDL generation with respect to a blind high level synthesis of the corresponding CAL models, as occurs for the *opt* design.

Table 4.2 summarises the optimal FIFOs sizes provided by Turnus and adopted for the *opt* and *red* versions of the *reconf* design. Please note that, despite employing the same FIFOs sizes, *opt* and *red* designs lead to different implementation values.

The resources amounts required by the considered designs are depicted in Table 4.3. Data have been provided by the Xilinx Synthesis Technology (XST) targeting the previously mentioned FPGA board. Reconfigurability guarantees a resource saving with respect to the

baseline non reconfigurable design (see columns 256 of Table 4.3). The maximum operating frequency supported by the all the designs is generally the same, meaning that the Sboxes insertion, and in turn reconfigurability, has not corrupted the very performance of the original designs. Synthesiser maps the FIFOs within FPGA Slice LUTs until they become too large, as in the 256 case, where they are inferred in the device RAM blocks (BRAMs). Thus the number of employed Slice Regs and LUTs grows up to the 64 FIFOs size case, while moving to the next case, 256, they decrease. The *opt* column reveals the positive impact of the Turnus optimisation phase, that ensures a minimal Slice Regs and LUTs demand (only when FIFO size is 1 the adopted resources are less than *opt*) and a limited amount of BRAMs (two more than the minimum value, required by 1, 16 and 64 FIFO size cases). Concluding, the case involving reduced Sboxes HDL modules, *red*, succeeds in saving one BRAM unit at the price of a little overhead in terms of Slices, if compared to *opt*. An additional cost is paid on the maximum achievable frequency that is reduced by the 3.5% for the *opt* design, since the fully combinatorial Sboxes determine a longer critical path within the system.

Table 4.3: Resources occupancy and operating frequency of the implemented solutions: the baseline non reconfigurable one (*n-reconf*) and those implemented leveraging on the proposed flow (*reconf*). These latter include fixed buffering solutions (1 to 256), the optimised buffering one (*opt*) and, finally, that with a reduced number of buffers (*red*).

	n-reconf	**reconf**					
		1	16	64	256	opt	red
Slice Regs	28864	18270	20588	21278	20643	18477	18698
Slice LUTs	66725	40707	45587	46933	46576	41212	42330
BRAMs	136	47	47	47	119	49	48
DSPs	36	18	18	18	18	18	18
Max Freq [MHz]	61.06	62.04	59.73	59.73	59.73	60.40	58.28

Table 4.4: Cycles per frame and frame rate per second estimations of the implemented solutions: the baseline non reconfigurable one (*n-reconf*) and those implemented leveraging on the proposed flow (*reconf*). These latter include fixed buffering solutions (1 to 256), the optimised buffering one (*opt*) and, finally, that with a reduced number of buffers (*red*).

	parallel	**reconf**					
		1	16	64	256	opt	red
cycles	225851	-	-	225851	225861	225869	225865
fps	270.4	-	-	264.5	264.5	267.4	258.0

The performance of a video decoder is usually given in terms of frame rate per second. In this sense, the CG reconfigurable decoder developed during the considered trials is dependent on the original isolated decoder implementations. Once fixed the operating frequency, the reconfigurable design can not achieve a frame rate higher than the one of the standalone implementation, since they are derived from the same RVC-CAL dataflow descriptions. Furthermore, the frame rate of *reconf* can only be smaller than the *n-reconf* one, that involves

the isolated decoders in parallel, due to the additional latency of the inserted Sboxes. Table 4.4 depicts the clock ticks (*cycles*) per frame and the frame rate (*fps*) produced by the considered designs. The *cycles* give a measure of the frame decoding latency, making it possible the evaluation of the Sboxes insertion impact within the *reconf* systems. The reported *fps* value refers to the frame rate of each design when it is driven by the respective maximum supported frequency clock, and it provides a value for the absolute timing overhead introduced by reconfigurability. The reported numbers are obtained as the average values between the ones measured during the execution of the Intra-only-decoder and the Inter-decoder.

The latency overhead related to the Sboxes introduction is quite low, as depicted by the *cycles* row of table 4.4. Also in terms of frame rate (*fps* row of Table 4.4) the difference between the *n-reconf* designs and the *reconf* ones is limited: *red* achieves about 4.5% less frames per second. Considering the effects of replacing standard Sboxes RVC-CAL processing elements with fully combinatorial ones, as occurs going from the *opt* to the *red* design, despite *red* presents less cycles per frame, its frequency drop causes a reduction in the frame rate of 3.5% with respect to the *opt* one. Table 4.4 illustrates also as when the FIFOs are undersized, such as for 1 and 16 size cases, it is not possible to perform any decoding due to deadlock issues.

The MPEG-4 decoders trials revealed that the MDC optimised reconfigurable design, involving fully combinatorial Sboxes, can be the best solution if resource minimisation is the leading design goal. On the contrary, when performance have to be maximised, the system totally generated by Xronos has to be preferred. Please note that, if wider contexts are addressed, more than two input dataflow functionalities can be provided and the conducted trade-off analysis, taking into consideration resources, latency and maximum operating frequency, may result in different conclusions.

4.2.2 Real Use Case: Multi-Quality JPEG Encoder

This Section is intended to provide a more realistic and generic use case for the evaluation of the proposed approach and for the validation of the assembled tool chain. It constitutes an example of how the modularity proper of the CG reconfigurability can be exploited in order to define generalised systems. In particular, the proposed use case focuses on the implementation of a dedicated system for JPEG encoding able to support several quality levels for the compressed image thanks to the reconfiguration.

The designs adopted for this use case refer to the simple profile ITU-T.IS 1091 JPEG encoding standard. The considered encoder supports input images in Raster 4:2:0 YCrCb format, thus it expects two 8x8 chrominance macro blocks (MBs) for every four 8x8 luminance one. Similarly to the MPEG-4 SP encoding described in Section 4.2.1, JPEG encoding is based on the sequential application of a two dimensional DCT, to bring the image in the frequency domain, a quantisation, that cuts the higher frequency components keeping only the lowest ones (those the human eye is most sensitive to) and, finally, a Huffman encoding algorithm, in order to compress data by exploiting the different occurrences of the image pixel values.

The quantisation step is the real hearth of JPEG compression. Quantisation relies on the quantisation tables, 8x8 matrices of coefficients, one for luminance and one for chrominance, that have to be multiplied with each MB. According to the considered quantisation tables, it is possible to achieve different compression ratios and, in turn, different qualities for the respective decoded image. Generally speaking, the higher the compression ratio, the lower the quality of the compressed image. The trade-off among compression ratio and decoded image quality is data dependent, so that it changes depending on the original input image.

Hereafter the quantisation tables provided by the JPEG Independent Group (JIG) [4] will be considered as a reference. Starting from these reference tables, by means of a simple scaling factor Q (it can go from 0 to 100) it is possible to obtain different trade-off levels between quality and compression ratio. The reference tables have a quality factor equal to 50. Higher Q values lead to better decoded image qualities at the price of a smaller compression ratio. Additionally, higher Q values require a more complex computation, resulting in longer execution times.

In this use case, a CG reconfigurable platform for JPEG encoding has been developed. It supports three different quality levels by keeping the execution time constant. Each quality level is achieved by configuring the encoder on a different working point (wp):

- *wp0*, providing a quality factor $Q = 50$;
- *wp1*, providing a quality factor $Q = 65$;
- *wp2*, providing a quality factor $Q = 80$.

Each encoder working point has been modelled through a different RVC-CAL dataflow network in order to feed the tool chain in charge of generating the CG reconfigurable JPEG encoder. Starting from a dataflow model of the reference JPEG encoder (*wp0* with $Q = 50$), higher performance solutions have been derived by placing more resources in parallel to speed-up the critical actors within the pipeline. This simple solution allowed the support of more accurate compressions ($Q = 65$ and $Q = 80$) without an increase of the execution time thanks to the throughput growth.

Figure 4.6 depicts the dataflow networks of the three considered encoding working points. The reference encoder dataflow (Figure 4.6.a on the image) is composed by some big atomic actors, the *quantiser*, *Huffman encoder* and *syntax writer*, and by a sub-network that implements the DCT, involving in turn six small actors. This configuration corresponds to the original working point *wp0*. The other working points, *wp1* and *wp2*, have been assembled once the lowest throughput actor has been identified on *wp0*. This actor is responsible of the bottleneck that limits throughput of the the whole encoding pipeline. The workload of such a critical actor can be theoretically split in order to being processed by a parallel identical actor, thus resulting in doubling the throughput of the whole critical step. The splitting is possible only if the critical actor is stateless or if it has a state evolving cyclically for a fixed number of iterations. Furthermore, it requires two additional actors: a *splitter*, able to separate the token flow, and a *merger*, in charge of properly reconstruct the processed token flow.

The throughput enhancement achieved with the critical actor splitting made it possible the design of *wp1* and *wp3*, supporting higher qualities with the same latency value of *wp0*. In particular *wp1* has be obtained by splitting the *Huffman encoder* actor of *wp0*, that revealed to be the critical one (see Figure 4.6.b). Whereupon, *wp2* has been derived by *wp1* through the splitting of the new critical branch, the whole DCT sub-network, plus an additional splitting of the *Huffman encoder* stage, leading to overall four different instances of the atomic encoding actor as depicted in Figure 4.6.c. In order to support several quantisation tables (scaled by Q factors equal to 50, 65 and 80 respectively), also the *quantiser* and *syntax writer* actors involved by the different working point dataflows are different.

It has to be noticed that the assembled automated tool chain strongly aids the design of multi-throughput dataflow systems, such as for the considered use case. The optimisation step exploits the information about the latency of the involved actions (*action weights*), so that the latency of the whole actors along the execution pipeline can be easily calculated. This means that the critical actors, those with the highest latency, are identified at design-time and the splitting can be directly applied, eventually in an automated way, on the system high level model.

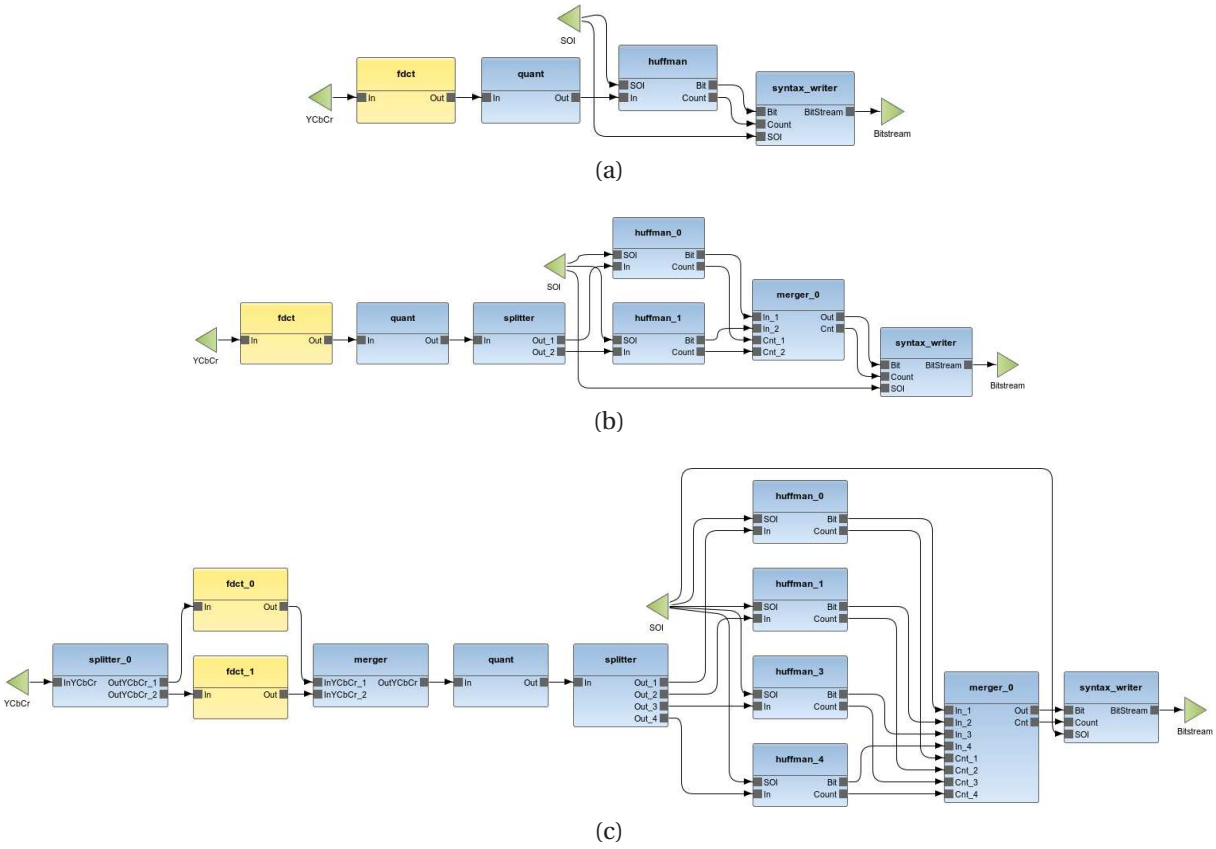


Figure 4.6: High level specification of the adopted encoder working points: a) *wp0*, b) *wp1* and c) *wp2*.

Experimental Assessment

The multi-quality JPEG encoder has been implemented on the same Xilinx Virtex 5 330 FPGA board than the MPEG multi-dataflow decoder. Similarly to the previous trials, in order to assess the proposed approach within a possible real use case, two designs have been considered:

- *n-reconf*, a non reconfigurable design where the all the three encoding working points (generated by Xronos) are placed on the same substrate without resource sharing;
- *reconf*, the corresponding reconfigurable design resulting from the proposed tool chain: optimal FIFO sizes provided by Turnus and fully combinatorial Sboxes given by MDC have been adopted (it has been obtained with the same tool chain setup than the *red* design of the MPEG decoders proof of concept presented in Section 4.2.1).

The actor composition of the two considered solutions is illustrated in Table 4.5. As previously discussed, the more the working point throughput, the more it requires resources. The *n-reconf* design employees a total amount of 43 computational actors against the 29 of the *reconf* design. In this latter, the sharing of 14 computational actors is achieved by the introduction of 18 Sboxes, so that the overall actor amount is greater than the non reconfigurable design one.

Figure 4.7 gathers resource occupancy data coming from the Xilinx Synthesis Technology tool. The reconfigurable design *reconf* saves always more than 30% for all the considered

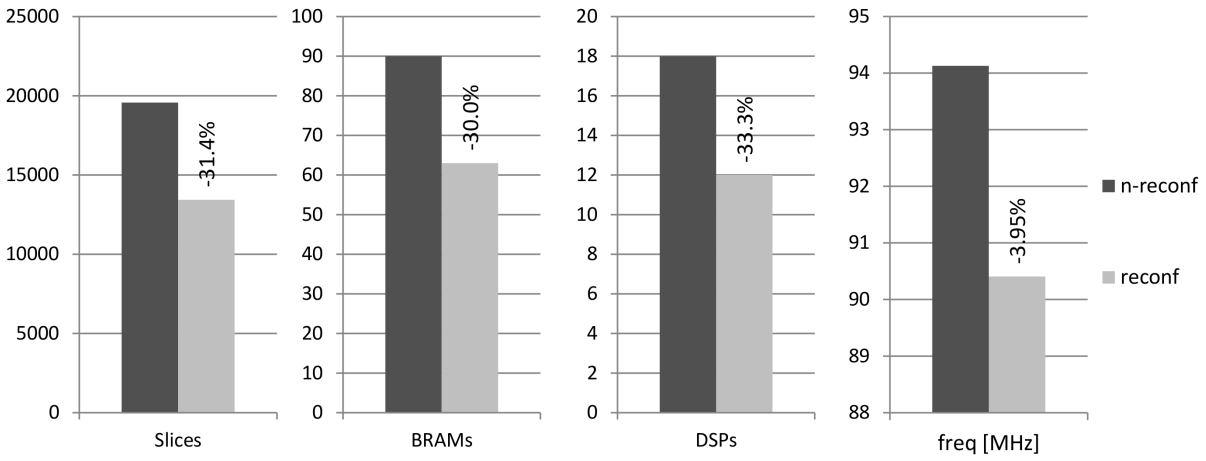


Figure 4.7: Resources occupancy of the JPEG working point designs.

Table 4.5: Comparison of the actors composition among the implemented solutions: the isolated working points designs with different quality factors Q (*wp0*, *wp1* and *wp2*), the non reconfigurable (*n-reconf*) and reconfigurable (*reconf*) ones, this latter obtained with the proposed approach.

	computational	**Sbox**	**overall**
wp0	9	0	9
wp1	12	0	12
wp2	22	0	22
n-reconf	43	0	43
reconf	29	18	47

resources. Dealing with the operating frequency, still reported by Figure 4.7, the reconfigurability cost is revealed: *reconf* support a maximum frequency that is 4% slower than the *n-reconf* one.

Table 4.6: Power consumption, in mW, of the implemented solutions: the baseline non reconfigurable one (*n-reconf*) and the one implemented leveraging on the proposed flow (*reconf*). Δ% defines the percentage increment/reduction between the *n-reconf* and the *reconf* design.

	n-reconf	**reconf**	Δ%
Quiescent	3294.1	3294.1	0.00
Dynamic	373	291	-21.83
Dynamic Q=50	368	290	-21.20
Dynamic Q=65	366	286	-21.86
Dynamic Q=80	384	298	-22.40
Total	3670.1	3585.1	-2.31

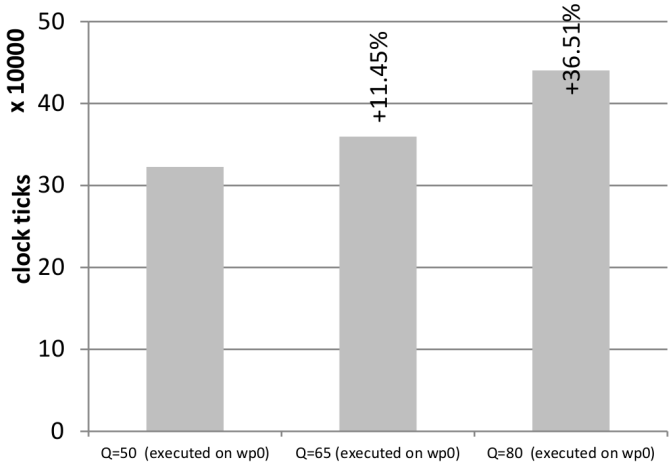


Figure 4.8: Performance of encoding with Q=50, Q=65 and Q=80 on the same architecture of *wp0*.

Table 4.6 depicts some FPGA power consumption numbers, retrieved through the Xilinx Power Analyzer tool, related to the two considered designs. The reported data for a given design is calculated as the mathematical average of the power consumed while the system is running on the three different working points. Apart the high quiescent term, that is very common on FPGAs and anyway equal for both the designs, the *reconf* solution achieves 20% dynamic power saving with respect to the *n-reoconf* one. The same dynamic power slowly grows going from a working point to one other with an increased quality factor Q. This behaviour can be expected, since the higher Q, the more resources are employed within the encoding pipeline. This suggests the possibility of adopting CG reconfiguration, besides for providing different quality factors (different hardware versions of the actors are required), also for enabling dynamic adaptation of the system to different power versus performance trade-offs. Due to the huge quiescent power dissipation of the FPGAs, the dynamic consumption differences among the working points are not visible on the overall power, but it could be appreciated in a prospective ASIC implementation, where the two power contributes are more balanced.

Prior to analyse the performance of the proposed designs, some considerations have to be done on the baseline encoding process. As shown in Figure 4.8, if different quality factors are adopted on the same encoding pipeline, the execution latency is directly proportional to Q. Thus the encoding of an image with Q=65 is longer than the encoding of the same image considering $Q = 50$, but it is shorter than selecting $Q = 80$. For this reason, *wp1* and *wp2* dataflow models have been modified so that, despite supporting higher quality factors than *wp0*, they keep more or less the same execution latency

The execution latency of the adopted working points when encoding the same input sample image is illustrated in Figure 4.9. *wp1* and *wp2* not only are able to provide the same frame rate of *wp0* but, actually, they additionally speed-up the processing. The execution time saving reaches 30% and 60% respectively for *wp1* ($Q = 65$) and *wp2* ($Q = 80$) if compared to *wp0* ($Q = 50$). The *wp1* and *wp2* designs are then able to provide an additional performance budget that could be exploited to support bigger quality factors or input images.

Going back to the non reconfigurable design, *n-reconf*, by analysing Figure 4.9 it is possible to appreciate how the performance degradation introduced by reconfigurability (due to

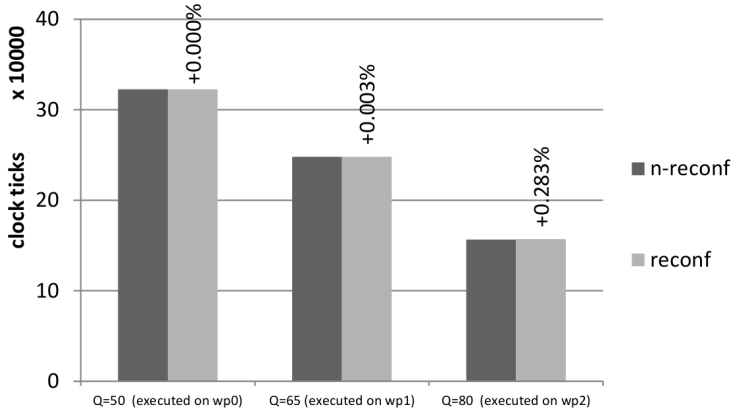


Figure 4.9: Performance of the JPEG working point designs.

the Sboxes insertion) is generally limited and always less than 5%.

It has to be highlighted that the presented use case is a generalizable application of the proposed design flow: starting from a baseline model of a given application, different working points can be obtained. Working points make the system flexible, providing several trade-off levels for the addressed metrics (e.g. latency, power, quality, etc.) without affecting the overall performance.

4.3 Chapter Remarks

The design of reconfigurable architectures is typically a complex and time consuming process. The baseline MDC functionality alone can not guarantee a fully automated instrument for this kind of architectures since it does not take care about some development aspects such as optimisation and hardware design. Dealing with these aspects by hand could threaten the benefits earned by the coupling between CG reconfigurability and dataflow models. By leveraging on MPEG-RVC framework, such benefits are not only achieved with very little designer effort, but it is also possible to obtain an efficient system implementation. From the developers point of view, the design flow is simplified. In fact, they just need to provide the starting DPN specifications of the functionalities to be implemented over the reconfigurable substrate, and they do not need to deal neither with manual HDL optimisation, nor with resource minimisation, nor with the system configuration to enable run-time reconfiguration. This happens as the optimal processing elements synthesis is provided by Turnus and Xronos and the multi-functional mapping and configuration are provided by the MDC tool. The assembled tool chain constitutes the unique flow in the MPEG-RVC scenario in charge of automatically mapping, optimising and designing in hardware reconfigurable systems.

To explain the benefits of each single phase of the proposed fully automated tool chain, a multi-dataflow decoder has been assembled embedding on the same CG reconfigurable design the RVC MPEG-4 SP Intra-only-decoder and the RVC MPEG-4 SP Inter-decoder. Synthesis results have been provided targeting a Xilinx Virtex 5 330 FPGA board. The CG reconfigurable paradigm, based on the adoption of switching modules in order to save resources, does not affect decoding functionality. Despite the presence of the Sboxes chains a sufficient frame rate is achieved. Then a real use case demonstrated the possibility of exploiting the

modularity of the dataflow models in combination with the CG hardware design paradigm, to create efficient multi-functional applications. In particular, a multi-quality MPEG-RVC JPEG encoder has been presented. This reconfigurable system is capable of providing, over the same platform, different working points, with increasing compression performance and no latency penalty. Therefore, the proposed approach demonstrated its capabilities in deploying flexible systems without affecting the overall performance.

List of Publications Related to the Chapter

Journal papers

- C. Sau, P. Meloni, L. Raffo, F. Palumbo, E. Bezati, S. Casale-Brunet and M. Mattavelli, *Automated Design Flow for Multi-Functional Dataflow-Based Platforms*, in Journal of Signal Processing Systems, (online August 2015), to appear.

Conference papers

- C. Sau, L. Raffo, F. Palumbo, E. Bezati, S. Casale-Brunet and M. Mattavelli, *Automated design flow for coarse-grained reconfigurable platforms: An RVC-CAL multi-standard decoder use-case*, in Proceedings of the 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), Samos Island (Greece), July 2014.

Chapter 5

Structural Profiler

In this chapter the structural profiling additional feature of the Multi-Dataflow Composer (MDC) will be presented. This feature constitutes one of the extensions of the tool, aiming at optimising the topology of the generated multi-dataflow network. Dealing with coarse grained (CG) reconfigurable platforms, resources mapping and dynamic reconfiguration can be very complex (see Chapter 2), so that finding out the optimal configuration of the system could be extremely time consuming. Within the described context, early stage strategies can aid the designer driving the development flow to the wanted project goals. In particular, when multiple constraints have to be met at the same time, such as for the modern embedded devices, early stage design space exploration (DSE) techniques can be very useful. DSE is an objective-driven process that discovers and evaluates all the possible design alternatives, called *design points*, during system development. Typically, DSE takes into consideration different metrics of interest (timing, resources usage, energy, etc.) and several design parameters (the number and type of cores, size and organisation of memories, interconnect topology, etc.). When performed at the early stages of the design flow, DSE normally deals with the system high level model.

The MDC structural profiler relies on an early stage DSE involving a Pareto based analysis for the solution of multi-objectives problems [50], conceived for CG multi-context architectures. The adopted DSE methodology is able to guide MDC in the definition of the most suitable and efficient architectural solutions, taking into consideration multiple performance/technological constraints to be met. In particular, exploiting low level back annotated information, the proposed methodology is able to perform a complete DSE evaluating the trade-off between resource usage, power consumption and operating frequency. The problem addressed does not aim at choosing among multiple specific design parameters; rather the presented work focuses on determining which parts of the hardware should be shared among different applications/kernels and which parts should be kept separated to achieve the best trade-off among the selected metrics of interest.

In Section 5.1 the flow of the developed structural profiler will be discussed in detail. Then, Section 5.2 will show the experimental results obtained in order to demonstrate the effectiveness of the profiling methodology. Lastly, Section 5.3 will conclude with some considerations and future improvements of the MDC structural profiler extension.

5.1 Profiling Aware Topology Definition

As deeply discussed in Chapter 3, the MDC mainly faces the problem of efficient resource mapping and management onto reconfigurable platforms. At this purpose it exploits the CG design paradigm in conjunction with dataflow based specifications to automatically handle complex multi-context executions providing, at the same time, efficient resource sharing for on-chip optimisation purposes and flexibility through run-time adaptivity. The tool combines together a set of input specifications provided as dataflow process networks (DPNs), producing as output the RTL description (in terms of HDL code) of the corresponding CG reconfigurable platform able to implement all the functionalities of the input specifications. MDC is designed to be connected to higher-level utilities by means of an adequate front-end, in charge of parsing the high level descriptions of the dataflows to be combined. In this way, relying on the chosen front-end, MDC in principle is able to process any type of dataflow model. At the moment MDC is coupled to Orcc [7].

At the base of the profiling aware topology definition strategy there is the MDC combination algorithm. In particular, topology optimisation for the resulting CG reconfigurable architectures is needed for the following reasons:

- in a multi-functional environment, merging all the DPNs together is not always the optimal solution. An excessive number of switching elements may overcome the benefits of sharing an actor, causing both area and static power loss.
- in the adopted iterative merging algorithm (see Chapter 3), two networks at a time are processed by MDC. This merging process generates chains of switching modules (Sboxes)[1]. This may negatively affect the operating frequency. Therefore, it would be more efficient to merge just a subset of the input DPNs.

It is fundamental to determine the (sub-)optimal design specification(s). The proposed profiling extension, whereby MDC has been supported, will be able to determine the design costs of different implementations, before prototyping, through the back annotation of low level information on the high level multi-dataflow specifications.

Figure 5.1 depicts an overview of the proposed early stage profiling chain, whose main phases are presented in the follow:

1. Sequences Extraction: the *Sequences Generator* defines all the possible network sequences the MDC tool will be fed with (Section 5.1.1).
2. MDC Merging Process: for each sequence the MDC tool extracts the multi-functional directed flow graphs (Section 5.1.2).
3. Profiling: each graph, representing a design point, is back annotated with synthesis information (currently extracted off line). Area, power dissipation and operating frequency are the estimated implementation costs (Section 5.1.3).
4. Pareto Analysis: the Pareto based analysis is carried out on the entire design space to determine the optimal system configuration(s), according to the selected design effort (Section 5.1.4).

It is important to notice that the exploration is carried out at a high level of abstraction, but the dataflow models are back annotated with a lower-level feedback. Users can choose,

[1] In the worst case scenario, where all the input DPNs share the same actor, the chain will have as many element as the iterations are.

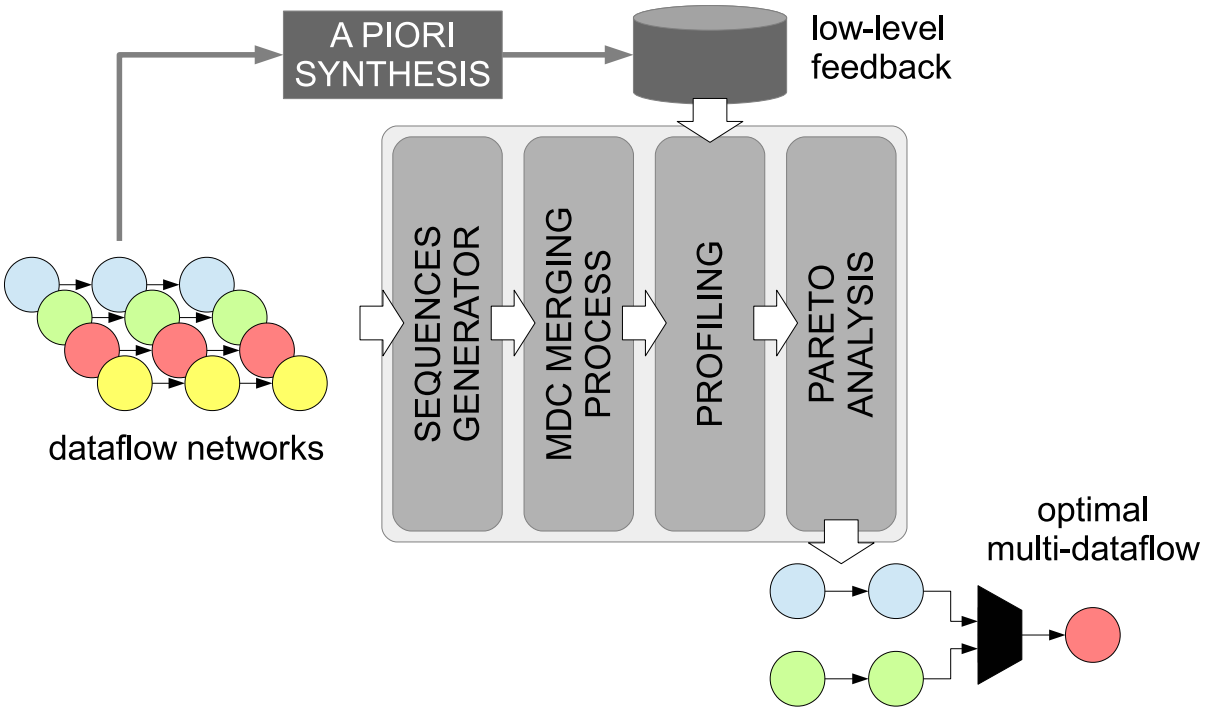


Figure 5.1: MDC structural profiler: an overview.

via the MDC graphical user interface, enabling or not the early stage profiling chain. Without it, MDC will merge the networks as provided. Design effort can be set to find the optimal topology in the area/power domain or in the frequency one. The rest of this section details the above listed phases and concludes with a step-by-step example (Section 5.1.5) of the defined methodology.

5.1.1 Sequences Extraction

As deeply explained in Chapter 3, given N input networks, the MDC implements an heuristic algorithm that merges incrementally two networks at a time. Keeping in mind that the merging process acts at the actor level, the feeding order of the N networks may change the output multi-dataflow topology in terms of cascade of Sboxes to access an FU when switching from one computation to another. During the sequences extraction phase, given a set of input DPNs, is in charge of generating all the possible feeding orders for MDC.

$$\begin{aligned} D &= D\_notMer + D\_Mer + D\_partMer \\ D &= 1 + N! + \sum_{k=1}^{N-2} C_{N,k} * (N-k)! \\ D &= 1 + N! + \sum_{k=1}^{N-2} \frac{N!}{(N-k)! * k!} * (N-k)! \\ D &= 1 + N! + \sum_{k=1}^{N-2} \frac{N!}{k!} \end{aligned} \tag{5.1}$$

Equation 5.1 determines the number and the type of multi-functional specifications that the sequences extraction phase is able to compute. Please notice that it processes the directed flow graphs (DFGs) of the inputs networks, already acquired by Orcc. Defining as D

the number of different feeding sequences and N the one of the input networks, the three terms in Eq. 5.1 are:

- *D_notMer* - The "static", not merged, composition of the *N* networks in parallel;
- *D_Mer* - MDC shares as much resources as possible among the *N* given networks, by merging all the common actors. "All-merged" solutions are provided. The number of possible sequences is equal to the number of all the possible permutations of the input networks;
- *D_partMer* - MDC will not maximise resource sharing, generating "partially-merged" solutions. All the possible combinations[2] that can be extracted from the set of input networks are placed in parallel with all the merged permutations of the other *N-k* networks, where k is the number of the objects of the current combination.

where *D_notMer* is the *static*, not merged, composition of the *N* DPNs in parallel; *D_Mer* is the *all merged* term that, maximising resource sharing, is given by all the possible permutations of the DPNs merged into a unique one; *D_partMer* is the *partially-merged* term that, not following the resource maximisation principle, provides all the sequences composed of the combinations[3] that can be extracted from a subset of *k* input DPNs placed in parallel with all the permutations of the other *N-k* networks merged together.

5.1.2 MDC Merging Process

For each sequence provided by the sequences extraction phase, the MDC tool extracts the multi-dataflow DFG. This is the step that actually generates the design points that will be evaluated in the following phases. The main steps required to generate the HDL description of a multi-functional reconfigurable architecture, starting from the DPN models of the functionalities to be implemented, are three:

1. Orcc parses the Input DPNs (along with their actors) and translates each of them into an Intermediate Representation (IR), which is basically a DFG.
2. The MDC front-end leverages on such flattened IRs to assemble a single multi-functional specification. MDC front-end also keeps trace of the system programmability through the *Configuration Table* (*C_TAB*).
3. The MDC back-end then creates the respective HDL CG reconfigurable hardware, mapping each actor on a different FU. FUs are passed as input to MDC within a proper components library that can be manually or automatically created. FUs granularity/-functionality does not affect the described flow.

For the purposes of this profiling flow, only the first two steps will be executed for each sequence generated by the sequences extraction phase. It has to be noticed that not always different feeding orders of the input DPNs lead to different multi-dataflow specification. In general more than one input sequence, once processed by MDC, gives the same multi-dataflow. This means that, in the successive profiling phase, when the design points are evaluated, it is possible to have multiple design points with the same metric values.

[2] A combination is a selection of all or part of a set of objects, regardless to the order in which they are selected. Given A, B and C, the complete list of possible selections would be: AB, AC, and BC (BA is not a combination since it contains the same object as AB).

[3] A combination is a selection of all or part of a set of objects, regardless to the order in which they are selected. Given A, B and C, the complete list of possible selections of two items would be: AB, AC, and BC.

Basing on the estimated metric values associated to the design points, the structural profiler will choose optimal solutions according to the selected design goal. If the optimal design point refers to more than one input sequence, the last evaluated input sequence will be considered in order to generate the final CG reconfigurable hardware platform.

5.1.3 Profiling

For each multi-dataflow DFG provided by MDC and corresponding to a different feeding order of the input DPNs, implementation costs are estimated by the *MDC Profiler*. In particular it considers three metrics, typically extremely important for the embedded system development: area occupancy, static power dissipation and maximum achievable operating frequency. The metrics are evaluated by exploiting a low level information[4] back annotating the isolated input dataflow implementations and the hardware processing elements (the HDL components library) corresponding to the involved actors. At this purpose, let's consider the mathematical formalism, introduced in Chapter 3, where a DPN is described by means of the corresponding DFG $G\langle V, E\rangle$, where V is the set of vertices of the graph and E is the set of edges (an ordered pair of vertices), respectively representing the processing elements (to be instantiated on the final multi-context system) and the connections among them.

Area and Power Analysis

In CG reconfigurable architectures, where resource redundancy reduction is one of the primary goals, the area estimation should necessarily be one of the metrics of interest to be taken into account while exploring the design space. Resource redundancy reduction implies also static power reduction. In fact, generally speaking, the less are the FUs involved, the less will be the static power consumption. The dynamic dissipation is a bit more complex metric, since it is related to the system switching activity, and its estimation for a future integration within the MDC structural profiler is still under scrutiny.

Having already back annotated the HDL components library with one value of estimated area and static power consumption per functional unit, the MDC profiler retrieves $\forall v_i \in V$ the correspondent back annotated values, a_i and p_i. Given as M the size of the V set, the following equations are considered:

$$Area(G) = \sum_{i=1}^{M} a_i \tag{5.2}$$

$$Power(G) = \sum_{i=1}^{M} p_i \tag{5.3}$$

to determine area and static power consumption of the considered design point. Since they are calculated basically in the same way, area and power metrics are expected to have the same trend, thus their minimisation is considered a unique design goal by the MDC structural profiler.

Timing Analysis

The operating frequency estimation is less straightforward. The chosen CG reconfigurable approach is based on the introduction, within the final system, of dedicated switching elements capable of correctly route data, to preserve computational correctness, while accessing shared resources. Merging together different input networks may require cascading

[4] Synthesis trials with the RTL Compiler of Cadence SoC Encounter. 90 nm CMOS technology.

several different Sboxes. These latter are logically simple and do not provide, at the moment, any type of pipelining; therefore, a cascade of Sboxes may negatively impact on the system critical path (CP). Dealing with simple computational actors (e.g. adders, subtractors, multipliers) involved in the input networks, it may be highly probable that the cascade of Sboxes would determine the CP. Whereas when the actors are responsible of complex operations (e.g. an FFT) that won't be the case.

For timing analysis purposes, SoC Encounter has been used to extract the CP associated to the *N* different input networks synthesised stand-alone. The MDC profiler $\forall G_i \in InN$ (being InN the set of input networks) retrieves the correspondent back annotated CP, CP_i, and defines $CPstatic = max(CP_i)$ as the CP of the multi-context system considering its "static" composition. Then it estimates the longest cascade of Sboxes ($seqSB$) within the considered DFG.

Given N_S as the number of Sboxes composing $seqSB$ and b as the number of bits of Sboxes data, the CP due to the cascade of Sboxes is given by the following empirical equation[5]:

$$CP\_seqSB = f(b) * ln(N_S) + g(b) \tag{5.4}$$

Coefficients f(b) and g(b) change depending on the type of Sbox (Sbox_1x2 or Sbox_2x1).

- Sbox1x2: $f(b) = 0.268 * b + 59.85,\ g(b) = -0.294 * b + 406.3$
- Sbox1x2: $f(b) = 0.114 * b + 87.70,\ g(b) = 0.185 * b + 393.1$

The MDC structural profiler finally compares $CPstatic$ and CP_seqSB: the maximum of these two values determines the operating frequency of the given design point:

$$CP(G) = max(CPstatic, CP\_seqSB) \tag{5.5}$$

5.1.4 Topology Selection

The final phase of the proposed profiling aware topology definition strategy is the Pareto analysis and it is driven by the design effort selected by the user. In particular, it is possible to choose between two different design efforts: area/power and frequency. The former involves both area and power since the estimation related to this two metrics is calculated in the same way. The obtained values are directly proportional and minimising one of them implies also minimising the other.

Note that the definition of two different design efforts is required. Indeed, if the effort is minimising area and power dissipation, it will be extremely important to determine the least occupying and consuming configuration, but minimising area and power not necessarily implies having also the best operating frequency. Therefore, depending on the given effort, the final stage provides as outputs the area/power sub-optimal multi-functional design point (*TOP.p*) or the frequency (*TOP.f*) one.

5.1.5 Step-by-Step Example

The proposed structural profiler flow considers as input the five DFGs $\alpha.java$, $\beta.java$, $\gamma.java$, $\delta.java$ and $\varepsilon.java$ acquired and parsed by Orcc from a set of original RVC-CAL dataflows.

[5]Coefficients in Equation 6.3 are technology dependent and have to be modelled for each target technology node by interpolating a training set of experimental results. This form derives by experiments carried out by means of the RTL Compiler (Cadence SoC Encounter), using ASIC CMOS 90 nm technology as reference, varying the number of Sboxes (from 1 to 100) and the number of bits of Sboxes data (1, 8, 16, 32, 64).

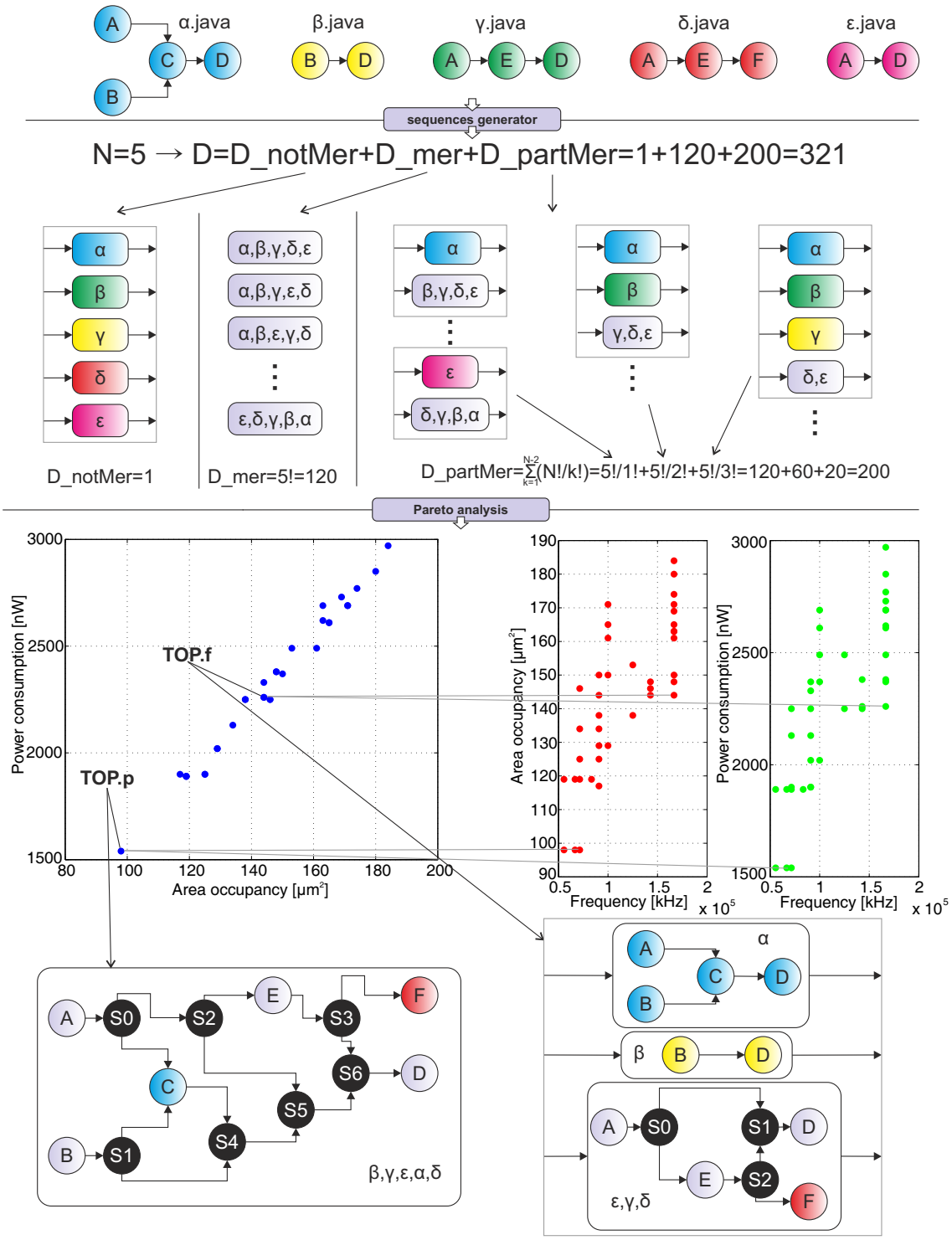


Figure 5.2: MDC structural profiler: a step-by step example.

During the sequences extraction phase, according to Eq. 5.1 and given 5 input networks, 321 different feeding sequences are extracted. Therefore, 321 points will constitute the design space of the possible system specifications. Figure 5.2 shows also how the "static", the "all-merged" and the "partially-merged" multi-functional output graphs will look like.

The profiling phase, leveraging on Eq. 5.2, Eq. 6.2 and Eq. 5.5, determines the implementation costs of each design point. By the Pareto analysis, according to the estimated costs, identifies *TOP.p*, the power-optimal system specification, and *TOP.f*, the frequency-optimal one. *TOP.p*, in this example, corresponds to an "all-merged" solution, where sharing is maximised and the feeding sequence is β-γ-ε-α-δ. *TOP.f*, instead, corresponds to a "partially-merged" solution, where the not-merged *combination* α-β is placed in parallel with the merged ε-γ-δ *permutation*.

5.2 Profiling Assessments

This section is intended to validate the proposed structural profiling flow for the MDC tool in the real use case scenarios, described in Section 5.2.1. Section 5.2.2 describes the trials setup chosen for the analysis and reports the experimental data that will validate the MDC profiling extension. Whereas Section5.2.3 will provide some considerations about the performed DSE.

5.2.1 Application Scenarios

In order to measure the effectiveness of the MDC structural profiler, three application scenarios for image processing (they are a subset of the ones adopted in Chapter 3 for the evaluation of the new version of MDC) are considered:

- Spatial Anti-Aliasing (UC1), to correct the distortion effects of an image downsampled violating the Nyquist constraint;
- Zoom (UC2), to scale an image by a given zooming factor interlacing pixels of the original image with other pixels coming from adaptive interpolations of the neighbouring ones;
- Deblocking Filter (UC3), to remove image distortions, like blocking and ringing, coming from compression processes.

The applications have been profiled to identify the most computationally intensive code segments (*computational kernels*). We have identified six, seven and eight kernels respectively for UC1, UC2 and UC3. These kernels have been modelled as DPNs according to the RVC-CAL dataflow model. The final intent is accelerating in hardware these applications by mapping their respective kernels over a reconfigurable multi-functional architecture, assembled with MDC.

Table 5.1 depicts an overview of the kernels composition, functionalities and occurrences within the three applications. The number of actors is a raw index of the kernel size, which is important to understand the kernel estimated metrics. A more precise insight of the size of each kernel is provided in Table 5.2 that reports both the overall kernel area (along with its percentage with respect to the corresponding use case) and the area shared with other kernels (along with its percentage with respect to the overall kernel area).

Back to Table 5.1, the data size is referred to the number of tokens that the kernel is able to process for every execution. This metric gives an idea of the kernel complexity and of

Table 5.1: Computational kernels of the adopted use cases.

app	kernel	# actors	# occ	data size	functionality
UC1	*sorter*	2	18659	25	vector sorting
	min_max	1	14864	2	maximum/minimum finding
	rgb2ycc	19	256	3x4x4	RGB to YCrCb colour conversion
	ycc2rgb	18	256	3x4x4	YCrCb to RGB colour conversion
	abs	1	124366	1	absolute value calculation
	corr	10	2321	25	vector correlation
UC2	*min_max*	1	15142	2	maximum/minimum finding
	abs	1	70045	1	absolute value calculation
	sbwlabel	17	966	16	edge block checking
	chgb	7	3072	4	bilevel/grayscale block checking
	cubic_conv	6	341	16	cubic filter convolution
	median	9	1253	4	median calculation
	cubic	10	1496	4	linear combination calculation
UC3	*min_max*	1	32806	2	maximum/minimum finding
	filter	13	2235	10	vector filtering
	clip	2	346	2	vector comparison
	inner	9	1108	4	vector weighting and sum
	mdiv	7	346	4	vector biasing
	sign	1	346	1	sign calculation
	rgb2yuv	19	256	3x4x4	RGB to YUV colour conversion
	yuv2rgb	18	256	3x4x4	YUV to RGB colour conversion

its execution latency. The occurrences of a given kernel (*#occ* in Table 5.1) are intended as the number of times the kernel is executed while running its corresponding application. If a kernel is executed several times, its activation frequency is high and, in turn, its switching activity largely impacts on the dynamic power consumption.

5.2.2 Experimental Data

With the purpose of validating the proposed methodology, different co-processor architectures, accelerating in hardware the applications presented in the previous section have been assessed. Each co-processor has been practically assembled with MDC. In any case, the ASIC synthesis results, to be discussed in the following section, have been performed using RTL Compiler (from the Cadence SoC Encounter commercial release). With respect to power consumption, all the power estimations have been extracted with RTL Compiler taking into account the real switching activity, collected during post-synthesis hardware simulations (performed with Cadence SimVision Debug) within VCD files.

The DPNs described above have been passed as input to the profiling aware chain to perform the DSE and identify the optimal design points for both the possible design efforts: area/power and frequency. UC1 is composed of 1951 design points, UC2 of 13693 and UC3 of 109493 points. These explorations are still feasible, UC3 analysis required just a couple of minutes. Therefore, an exhaustive exploration was performed and it was not necessary to apply any pruning on the design space size.

Table 5.2: Area occupancy of the kernels within the adopted use cases targeting a 90 nm ASIC technology. [* Percentages with respect to the UC total area; ** Percentages with respect to the kernel total area.]

app	kernel	area [μm^2]	% *	shared area [μm^2]	% *	% **
UC1	*sorter*	12010	12.90	3946	4.24	32.86
	min_max	3946	4.24	3946	4.24	100.00
	rgb2ycc	37084	39.82	13955	14.98	37.63
	ycc2rgb	39862	42.80	13955	14.98	35.01
	abs	2089	2.24	2089	2.24	100.00
	corr	31956	34.31	10820	11.62	33.86
UC2	*min_max*	3919	3.33	3919	3.33	100.00
	abs	2089	1.78	2089	1.78	100.00
	sbwlabel	52943	45.01	10831	9.21	20.46
	chgb	19077	16.22	12579	10.69	65.94
	cubic_conv	20613	17.52	12444	10.58	60.37
	median	23782	20.22	16160	13.74	67.95
	cubic	21728	18.47	15478	13.16	71.24
UC3	*min_max*	3943	2.72	3943	2.72	100.00
	filter	55921	38.60	15083	10.41	26.97
	clip	6988	4.82	3943	2.72	56.43
	inner	23302	16.09	15911	10.98	68.28
	mdiv	19346	13.35	11279	7.79	58.30
	sign	842	0.58	0	0.00	0.00
	rgb2yuv	51195	35.34	20816	14.37	40.66
	yuv2rgb	51696	35.69	33815	23.34	65.41

For all the considered use cases the MDC structural profiler does not detect any degradation of the reconfigurable system maximum operating frequency. This means that, according to Eq. 6.3, the Sboxes insertion caused combinatorial paths (see Section 5.1.3) shorter than those of the isolated DPNs: resource sharing preserved the performance of the isolated applications. Since a single data is representative of all the design space, for every considered scenario the frequency graph is not presented. Area and power estimations reported in the Pareto graphs have been derived according to Eq. 5.2 and Eq. 6.2.

UC1 area versus power Pareto graph is shown in Figure 5.3. It is possible to identify 3 different clusters: *a1*, where the DPNs kept in parallel, if present, are always the smallest ones (*abs*, *min_max*, *sorter*); *a2*, where the largest DPN (*ycc2rgb*) is always merged and, sometimes, the second largest kernel (*rgb2ycc*) is kept in parallel; and, *a3* that involves the remaining design space points, including the non reconfigurable one (which is the worst point for this UC). In frequency, the estimated maximum value due to the longest Sboxes chain is 1.56 GHz, which is larger than the one fixed by the isolated input DPNs (202.6 MHz). Therefore, *TOP.p* and *TOP.f* are coincident, corresponding to an *all-merged* solution that belongs to *a1*:*TOP.p/f* is 94513 μm^2 large and consumes 49.1 μW. With respect to the synthesised design the estimation error is 1.5% in area and 2.3% in power.

UC2 area versus power Pareto graph is depicted in Figure 5.4. There are 4 identified clusters: *z1* keeps (again) in parallel the smallest DPNs (*abs*, *min_max*); *z2* and *z3* respectively

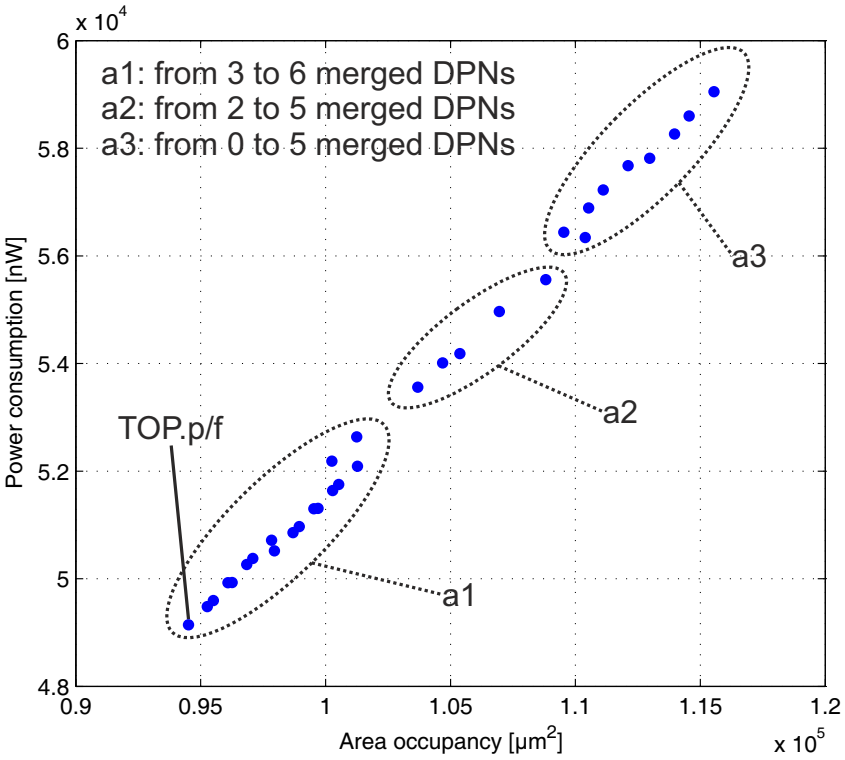


Figure 5.3: UC1 area vs power Pareto graph.

keep in parallel always one and only one or two and only two of the largest kernels (*sbwlabel*, *chgb*, *cubic*, *median*, *cubic_conv*); *z4* contains the remaining design space points, including the non reconfigurable one (which is not the worst). In terms of frequency the Sboxes chain maximum value is 1.51 GHz against the 384.6 MHz of the isolated DPNs. Again, *TOP.p* and *TOP.f* are coincident and correspond to an *all-merged* solution that belong to *z1* with 116619 μm^2 in area and 58.3 μW in power. The estimation error is now 0.9% for the area and 1.3% for the power.

Figure 5.5 illustrates the area versus power Pareto graph of UC3. Cluster *d1* involves design points that keep in parallel the smallest DPNs (*clip*, *min_max*, *sign*), while *d2* groups the combinations where also the medium size kernels (*inner* and *mdiv*), but not the biggest ones (*rgb2yuv*, *yuv2rgb*, *filter*), are kept in parallel, one at a time. In cluster *d3* also *rgb2yuv* is among the kernels that may be kept in parallel and in *d4 yuv2rgb* is added to the list. The last cluster, *d5*, contains the remaining design space points, including the non reconfigurable one (which is not the worst). In frequency, the maximum achievable value is 202.1 MHz, fixed by the input DPNs. The frequency associated to the longest Sboxes chain is estimated in 1.52 GHz. Again, *TOP.p* and *TOP.f* are coincident and correspond to an *all-merged* solution in *d1* with 147045 μm^2 in area and 76.1 μW in power. The estimation error for this UC is 1.5% for the area and 2.9% for the power.

Summarising, the proposed profiling aware methodology aiding the MDC baseline functionality allowed the automatic identification of the best topology configurations within a large set of design points. The MDC merging process, leading to the generation of the multi-functional platform, is not decreasing the performance of the isolated input DPN applications, guaranteeing the same maximum achievable frequency. The area and power estimations provided by the profiling phase, thanks to the dataflow modularity, reach very high accuracy

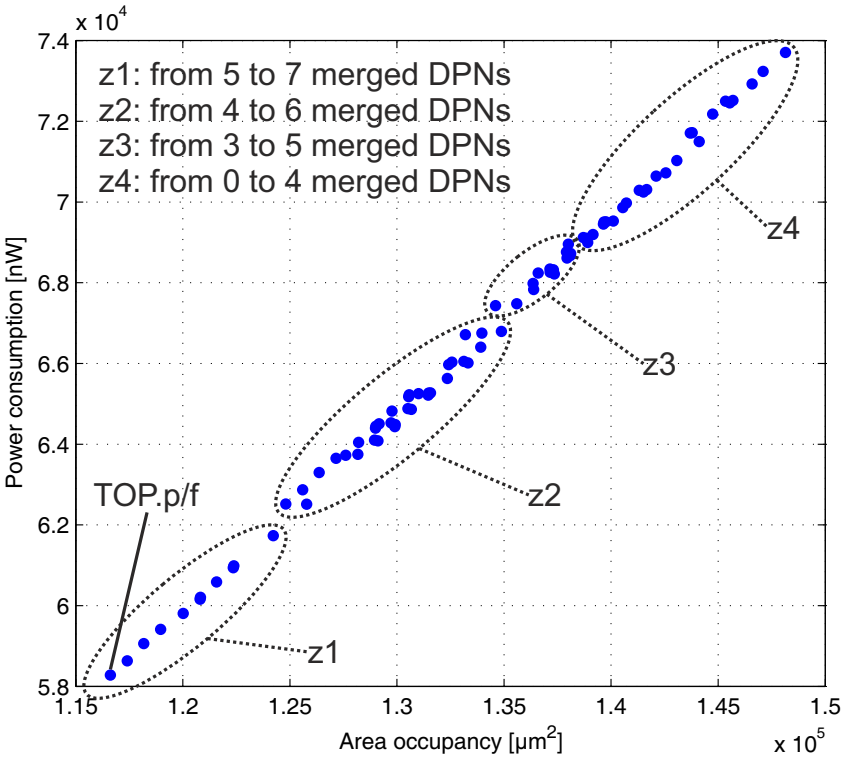


Figure 5.4: UC2 area vs power Pareto graph.

rates, with errors always below 3%.

5.2.3 DSE Considerations

Dealing with DSE problems, exhaustive search approaches, where all the design points within the design space are explored and characterised in terms of objective functions, are generally not feasible. Design space cardinality, and consequently the exploration time, may explode. The explorations proposed in this paper required at maximum to evaluate 13693 points, which is still feasible since it took a couple of minutes. Nevertheless, as soon as the number of input networks grows, it may not be doable anymore. Figure 5.6 shows the $3 * N!$ relationship that holds between the design space size and the number of input networks N.

Focusing on the MDC design space, the related exploration aims at identifying *TOP.f* and *TOP.p*. These considerations are valid:

- TOP.p normally is an "all-merged" solution. By construction, the smallest is the number of actors, the smallest would be the values retrieved by Eq. 5.2 and Eq. 6.2.
- TOP.f may be a "partially-merged" solution. The fewer networks you merge, the shorter should be the worst critical path.

Therefore, it would be possible to exploit these assumptions to analyse just those parts of the design space that are relevant for the given problem, basing on the effort set by the user. Still, on the considered portions, you will end up doing an exhaustive exploration.

To avoid exhaustive explorations, heuristic algorithms can be successfully used to reduce the overall exploration time by computing an approximated Pareto set of configurations, as in [69] and in [70]. In our case we may use similar approaches to reduce the cardinality of

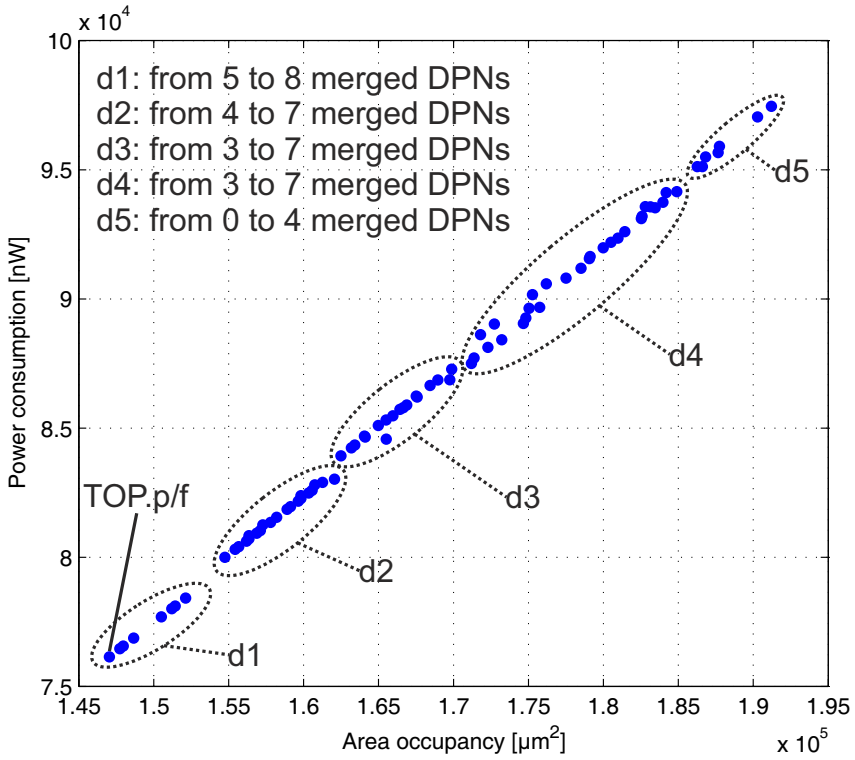


Figure 5.5: UC3 area vs power Pareto graph.

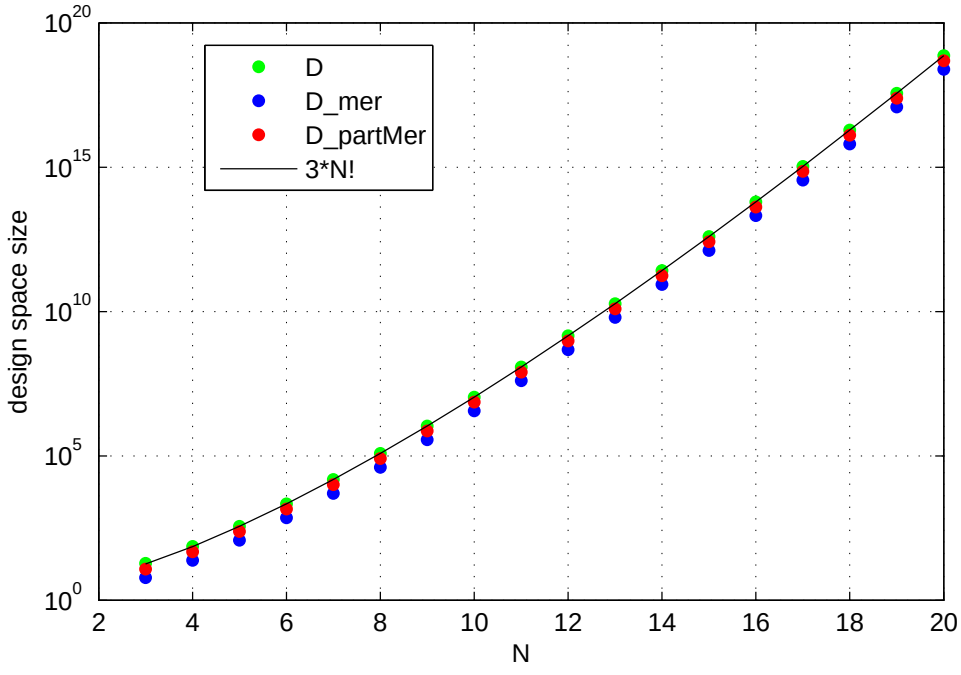


Figure 5.6: Design space size trend with respect to the number of input networks.

our design space. Moreover, automated strategies based on high level information can be proposed. In particular, with respect to the exploration of the "partially-merged" part of the

design space, two meaningful metrics could be considered:

- the probability of each actor to impact on the system operating frequency, which is proportional to the number of network accessing it. If the network(s) containing the actor(s) with the highest probability are placed in parallel, rather than merged, higher operating frequency values are achievable, since smaller Sbox chains will be present. A smart selection of the *combinations* in the third term of Eq. 5.1 can be implemented.
- the power overhead introduced when a network is placed in parallel, rather than merged.

With respect to the exploration of the "all-merged" part of the design space, instead, statistical studies may lead to the definition of a proper heuristic method for the exploration of the design points that have different frequencies, at the minimum area and power values.

As a conclusion, clearly the design space size explosion, in some cases, may be an issue. Different properties/characteristics/metrics of the design space can be exploited to define a robust and scalable heuristic algorithm performing approximated Pareto analysis. The actual implementation of those heuristics will constitute one of the future improvements of the proposed work.

5.3 Chapter Remarks

A structural profiling optimisation methodology for the MDC baseline tool has been presented in this Chapter. The proposed methodology is based on the DSE of the MDC design space and exploits some target specific feedback information to guide designers targeting multi-functional reconfigurable architectures. The final goal is to provide an efficient support to determine which parts of the given high level specifications should share the same hardware resources (if possible) and which parts should be better kept separated to meet/improve the expected system performance. Back annotated low level information are exploited at the dataflow models level to estimate the different metrics of interest necessary to carry out a complete Pareto analysis. Coupling this exploration methodology with automatic architectural synthesis flows could be extremely beneficial to help in managing the complexity of modern multi-functional application scenarios by solving multi-objective characterisation problems.

Given different image processing application scenarios, the proposed structural profiler has been adopted to assemble the reconfigurable computing core of different co-processing units. It has been demonstrated how topology may affect the area and the static power dissipation of the obtained CG architectures. The implemented profiling and exploration strategy has been capable of finding the optimal system topology through very accurate estimations.

List of Publications Related to the Chapter

Journal papers

- F. Palumbo, C. Sau and L. Raffo, *Coarse-grained reconfiguration: dataflow-based power management*, in IET Computers and Digital Techniques, Volume 9, Issue 1, January

2015, pp 36-48.

- F. Palumbo, T. Fanni, C. Sau and P. Meloni, *Power-Awarness in Coarse-Grained Reconfigurable Multi-Functional Architectures: a Dataflow Based Strategy*, in Journal of Signal Processing Systems, (online February 2016), to appear.

Conference papers

- F. Palumbo, C. Sau and L. Raffo, *DSE and Profiling of Multi-Context Coarse-Grained Reconfigurable Systems*, in Proceedings of the 2013 8th International Symposium on Image and Signal Processing and Analysis (ISPA), Trieste (Italy), September 2013.
- F. Palumbo, C. Sau and L. Raffo, *Power-awarness in coarse-grained reconfigurable designs: A dataflow based strategy*, in Proceedings of the 2014 IEEE Workshop on Signal Processing Systems (SiPS), Belfast (Northen Ireland), October 2014.

Chapter 6

Dynamic Power Manager

This chapter is intended to discuss the low power extension whereby the Multi-Dataflow Composer (MDC) has been provided. MDC natively addresses area and, in turn, power minimisation by sharing hardware processing elements (PEs) among different application specific datapaths, provided as dataflow networks. However, power reduction in modern embedded systems design has become an extremely important issue, pushed by the high demand of handheld devices, requiring efficient execution of multiple applications while complying with battery-life limits. The power issue, in the considered context of coarse grained (CG) reconfigurability, is exacerbated by the complexity and heterogeneity of such a kind of architectures. Therefore, the MDC baseline functionality can not guarantee the successful addressing of specific scenarios where the power requirements are extremely demanding.

Reducing the power consumption of an hardware design at the register level by hand is tedious and no longer practical for larger systems. Different design tools (such as the Synopsys Power Compiler or the Cadence Soc Encounter) offer automatic support for power management, normally at a very low level (e.g. implementing fine grain clock gating, providing multi-threshold and/or multi-supply physical libraries, etc). Nevertheless, this turned out to be not sufficient and higher level model based strategies are needed (see Chapter 2). The definition of automated methodologies for power management is of primary importance to aid designers, while reducing debugging and deployment costs. At this purpose, dealing with signal processing applications, very often algorithms can be effectively described exploiting the dataflow based formalism. Dataflow models of computation are characterised by distinctive features, such as modularity and parallelism highlighting as described in Chapter 2, that could aid in challenging these issues and cut-down time to market. MPEG group has defined a complete design framework (Reconfigurable Video Coding, RVC) enabling decoders specification by selecting components from a given library, leveraging on the dataflow formalism. MDC has been aligned with MPEG-RVC framework (it supports the related dataflow formalism), allowing an easy management and composition of dynamically reconfigurable systems in such a context. Nevertheless, despite RVC offers several different tools, in its reference design framework power management is still an open issue.

To make some steps forward towards filling this gap, during this thesis work an early stage methodology, where dataflow high level decisions are reflected at the hardware level, has been defined around the baseline MDC functionality. The developed methodology addresses power management exclusively for CG reconfigurable systems. These architectures typically involve redundant resources since not all the available PEs are used to implement every supported functionality. Thus, during the execution of a given context, some resources

may be in an idle state, uselessly dissipating precious energy. Starting from different high level specifications, the proposed methodology identifies directly on the high level models disjointed homogeneous logic regions, where the platform resources can be enabled/disabled together without affecting the overall system performance. Resources activation can be managed with standard state of the art power saving strategies, such as clock gating (for further details please refer to 2).

Clock gating acts at the clock net level responsible for more than the 40% of the dissipation [103]. To operate on clock (for power saving purposes) there are two strategies. The fine grain one acts at the registers level (to cut down the power consumed for refreshing the flip-flops and also that of the combinatorial logic driven by their values). The coarse grain one acts at a higher level: the clock is shut down at the unit level and this allows reducing the power consumption related to the clock tree. The major complexity for coarse grain strategies is the identification of the resources that have to be mapped on a specific logic domain. The dynamic power manager proposed to extend the baseline MDC feature addresses this particular aspect for RVC-CAL compliant systems. It is able to automatic partitioning the high level multi-functional specification of the system into homogeneous logic regions. Therefore, in the case of clock gating, the different clock domains are early identified in the design flow.

The rest of the chapter is organised as follow: Section 6.1 will describe in detail the proposed dynamic power manager; Section 6.2 will discuss the experimental trials conducted in order to evaluate the effectiveness of the approach; lastly, Section 6.3 will conclude with some final remarks.

6.1 Low Power CG Reconfigurable Systems Design

In this section the dynamic power management strategy built around the MDC baseline functionality will be described. MDC was conceived for the automatic creation and management of multi-dataflow systems. It was meant to address the difficulty of mapping different applications onto a CG reconfigurable architecture. Its final goal is automating such a mapping process while minimising hardware resources, with consequent area/energy saving. MDC works at a high level of abstraction. This means that, as soon as the different specifications are acquired and transformed into directed graphs, it does not matter the starting dataflow model. At the moment MDC is directly coupled to Orcc [7], so that dataflow process networks, expressed as XDF[1] files, are processed.

In the current flow, the MDC front-end leverages the Intermediate Representation (IR), in java code, generated by the Orcc front-end, to assemble a single multi-dataflow specification. The MDC back-end, then, creates the respective HDL CG reconfigurable hardware, mapping each actor on a different functional unit. These latter are passed as input to the MDC within the HDL components library and can be manually or automatically created. MDC is capable of assembly any type of reconfigurable platform, without any knowledge of the units granularity/functionality. The maximisation of resource sharing among the given input specifications is ensured as well as a quick, single-cycle, reconfiguration based on low overhead switching blocks (Sboxes) placed at the crossroads between the different paths of data. Sboxes are simply responsible of data routing, without any computational overhead.

The power management strategy that has been embedded within the additional features of MDC, so that the disjointed logic regions can be identified yet on the multi-dataflow IR specification. The logic regions correspond to groups of actors that are always active/inactive

[1] The XML Dataflow Format (XDF) is an XML dialect, used to describe dataflow graphs and standardised by MPEG

together during the system execution. Information related to this system partitioning is then exploited during the hardware platform generation in order to physically switch off them when unused.

The proposed dynamic power management approach enables early stage partitioning of the design onto disjointed logic regions. This process leads to the identification, from the directed graphs of the input networks, of the actors that are active/inactive together and groups them within homogeneous logic regions. On the basis of the found partitioning, at the hardware level clock gating techniques are applied. Users are just required to specify, on the MDC graphical user interface, the final target device, since different targets require different physical solutions. At the moment, both AND gates[2] cells for ASIC designs and BUFG cells for FPGA implementations on Xilinx boards are featured.

The proposed early stage power management strategy is mainly composed of two phases:

1. identification of the minimal set of logic regions;
2. merging process of the logic regions (where necessary).

The rest of this section provides a mathematical treatment of the logic regions identification along with a description of the optional merging process. The details about the algorithm implemented in the practice by the dynamic power manager extension of MDC are attached in Appendix B. The last part of the section will propose a step-by-step example with the intention of clarifying the defined methodology.

6.1.1 Logic Regions Identification Process

Logic regions identification is performed regardless the chosen hardware target, to minimise on-chip redundancy. As introduced in Chapter 3, any dataflow process network can be seen as a directed graph, $G\langle V, E\rangle$, where:

- V is the set of its actors v;
- $E \subseteq V \times V$ is the set of the edges e among its actors.

During the merging process, MDC maps each set of actors V_i belonging the i-th input network to a set of actors $V_i' \subseteq V$, where $G\langle V, E\rangle$ corresponds to the output multi-dataflow specification. Considering S as the complete set of logic regions, this step implies to minimise the number of elements, N, within S:

$$minimise\ N, \quad S = \{S_1, S_2, ...S_N\} \tag{6.1}$$

where S_i is the i-th logic region.

The minimum N solving this problem is the number of input networks M, when they do not share any actor. When shared actors are present, the partition S, composed of M sets, constitutes the starting point to determine N. Normally, S is composed of two different types of sets:

- S_{comp} contains those sets corresponding to a specific input network, involving non shared actors;
- S_{shared} contains those sets corresponding to more than one network, involving shared actors.

[2]Glitches are a false problem in the RVC domain: clock enables do not have a high switching activity.

Two dedicated map structures, namely the *Complementary map* (*Cmap*) and the *Intersection map* (*Imap*), are necessary to keep trace of the identified logic regions.

Given $A = \{\alpha_1, \alpha_2, ..., \alpha_M\}$ as the set of the input networks, let's define the *Complementary map* as:

$$Cmap = \begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_j \\ \vdots \\ c_M \end{array} \begin{array}{c} \begin{array}{cccccc} \alpha_1 & \alpha_2 & \cdots & \alpha_j & \cdots & \alpha_M \end{array} \\ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1j} & \cdots & w_{1M} \\ w_{21} & w_{22} & \cdots & w_{2j} & \cdots & w_{2M} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j1} & w_{j2} & \cdots & w_{jj} & \cdots & w_{jM} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{Mj} & \cdots & w_{MM} \end{pmatrix} \end{array}$$

where

$$c_j = \alpha_j \setminus \bigcup_{\substack{i \neq j \\ i=1}}^{M} \alpha_i$$

and

$$w_{ij} = \begin{cases} 0 \ if \ i \neq j \\ 0 \ if \ i = j \ and \ c_j = \emptyset \\ 1 \ if \ i = j \ and \ c_j \neq \emptyset \end{cases}$$

Having $w_{jj} \neq 0$ means that:

1. the input network α_j has, at least, one actor that is not shared with the other networks;
2. c_j is a logic region containing the non shared actor(s) of α_j.

Therefore, once *Cmap* is completely determined, S_{comp} can be derived as all the $c_j \neq \emptyset$. S_{comp} sets can be M as maximum (if all the M input networks are composed, at least, of one non shared actor).

In order to define the *Intersection map* you need to compute the set of all the non shared actors (Γ) as:

$$\Gamma = \bigcup_{i=1}^{M} c_i$$

and the complementary set of all the shared actors (Δ) as:

$$\Delta = \bigcup_{i=1}^{M} \alpha_i \setminus \Gamma$$

the *Intersection map* then is given by:

$$Imap = \begin{array}{c} \\ \delta_1 \\ \delta_2 \\ \vdots \\ \delta_j \\ \vdots \\ \delta_K \end{array} \begin{array}{c} \begin{array}{cccccc} \alpha_1 & \alpha_2 & \cdots & \alpha_j & \cdots & \alpha_M \end{array} \\ \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1j} & \cdots & p_{1M} \\ p_{21} & p_{22} & \cdots & p_{2j} & \cdots & p_{2M} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{j1} & p_{j2} & \cdots & p_{jj} & \cdots & p_{jM} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{K1} & p_{K2} & \cdots & p_{Kj} & \cdots & p_{KM} \end{pmatrix} \end{array}$$

where K is the cardinality of Δ, $\delta_j \in \Delta$ and

$$p_{ij} = \begin{cases} 0 \; if \; \delta_j \cap \alpha_i = \emptyset \\ 1 \; if \; \delta_j \cap \alpha_i \neq \emptyset \end{cases}$$

Having $p_{ij} \neq 0$ means that the shared actor δ_j belongs to the input network α_i. *Imap* may contain replicated rows: δ_x and δ_y associated both to α_m and α_n need to be grouped within a unique logic region, since they will be switched on and switched off together. To eliminate possibly replicated rows, consider *Imap* as a block matrix:

$$Imap = \begin{pmatrix} Imap_1 \\ Imap_2 \\ ... \\ Imap_j \\ ... \\ Imap_K \end{pmatrix}$$

and process it as follow:

$$\forall \; i > j \; if \; Imap_i = Imap_j \implies Imap_i = \emptyset \; and \; \delta_j = \delta_j \cup \delta_i \; and \; \delta_i = \emptyset$$

At this point S_{shared} can be derived as all the $\delta_j \neq \emptyset$. S_{shared} sets can be K as maximum (if no replicated or null rows are present on *Imap*).

Summarising, S_{comp} and S_{shared} represent all the logic regions in the system, which are $M+K$ at maximum. These regions are driven accordingly to *Cmap* and *Imap* that, together, represent the *Association map* of the system, relating each logic region to the original input networks triggering it. Given the input network α_x to determine its associated logic regions you need to:

1. check on the *Cmap* which row has a 1 in the α_x column:
   - none: α_x is composed of shared actors only;
   - the q-th row: α_x triggers the c_q logic region, composed of non shared actors.
2. check on the *Imap* which row(s) has a 1 in the α_x column:
   - none: α_x is composed of non shared actors only;
   - the q-th row(s): α_x triggers the δ_q logic region(s), composed of shared actors.

In order to uniquely define which clock signal should drive the different actors $\in G$, S must be a partition of the set of actors V: meaning that S is a division of V as a union of non overlapping and non empty subsets. Moreover, S must also contain a partition of the set of actors V_i' of each input network. In particular, this would mean that, when an input network is executed, exclusively its actors will be effectively active; whereas, all the other actors will be switched off to avoid useless power consumption. All these constraints and conditions are summarised by the Equation 6.2 and Equation 6.3:

$$S = P\{V\} \tag{6.2}$$

$$\forall i \in [1, M], \; \exists S' \subset S \; : \; S' = P\left\{V_i'\right\} \tag{6.3}$$

where $P\{V\}$ indicates the partition of V and M is the number of input dataflow process networks.

6.1.2 Logic Region Merging Process

The second phase of the proposed early stage power management strategy is the logic regions merging, performed to reduce their amount. FPGAs have a limited number of BUFG cells available. Sub-optimal regions are identified: there will be switching activity also in the unused functional units. Reduction can be done, according to two cost functions based on:

- minimising the contribution of useless activity due to the number of functional units per logic region - Given c_i as the cardinality of the i-th logic region and P as the number of networks activating it, we can define wN_i:

$$wN_i = c_i * P \tag{6.4}$$

  as the *weight* of considered region.

- minimising the power amount due to the functional units per logic region - Given p_i as the estimated power consumption (see Eq. 6.2) of the i-th logic region and P as the number of networks activating it, we can define wP_i:

$$wP_i = p_i * P \tag{6.5}$$

  as the *weight* of considered region.

The dynamic power manager, in both cases, while merging the different logic regions will aim at keeping all the weights as minimal as possible.

6.1.3 Step-by-Step Example

To clarify this second part of the proposed dynamic power management methodology let's consider the same five input networks of the step-by-step example provided in Chapter 5: $\alpha = \{A, B, C, D\}$, $\beta = \{B, D\}$, $\gamma = \{A, E, D\}$, $\delta = \{A, E, F\}$, $\varepsilon = \{A, D\}$.

The *Complementary map* is the following:

$$Cmap = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{array} \begin{array}{c} \begin{array}{ccccc} \alpha & \beta & \gamma & \delta & \varepsilon \end{array} \\ \begin{pmatrix} w_{1\alpha} & 0 & 0 & 0 & 0 \\ 0 & w_{2\beta} & 0 & 0 & 0 \\ 0 & 0 & w_{3\gamma} & 0 & 0 \\ 0 & 0 & 0 & w_{4\delta} & 0 \\ 0 & 0 & 0 & 0 & w_{5\varepsilon} \end{pmatrix} \end{array}$$

where

$c_1 = \alpha \setminus (\beta \cup \gamma \cup \delta \cup \varepsilon) = \{A, B, C, D\} \setminus (\{B, D\} \cup \{A, E, D\} \cup \{A, E, F\} \cup \{A, D\}) = \{A, B, C, D\} \setminus \{B, D, A, E, F\} = \{C\} \implies w_{1\alpha} = 1$

$c_2 = \beta \setminus (\alpha \cup \gamma \cup \delta \cup \varepsilon) = \{B, D\} \setminus (\{A, B, C, D\} \cup \{A, E, D\} \cup \{A, E, F\} \cup \{A, D\}) = \{B, D\} \setminus \{A, B, C, D, E, F\} = \{\emptyset\} \implies w_{2\beta} = 0$

$c_3 = \gamma \setminus (\alpha \cup \beta \cup \delta \cup \varepsilon) = \{\emptyset\} \implies w_{3\gamma} = 0$

$c_4 = \delta \setminus (\alpha \cup \beta \cup \gamma \cup \varepsilon) = \{F\} \implies w_{4\delta} = 1$

$c_5 = \varepsilon \setminus (\alpha \cup \beta \cup \gamma \cup \delta) = \{\emptyset\} \implies w_{5\varepsilon} = 0$

According to these values, the *Complementary map* is:

$$Cmap = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{array} \begin{array}{c} \begin{array}{ccccc} \alpha & \beta & \gamma & \delta & \varepsilon \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

and $S_{comp} = \{c_1, c_4\}$, with $c_1 = \{C\}$ and $c_4 = \{F\}$.
The set of all the non shared actors, Γ, is:

$$\Gamma = \bigcup_{i=1}^{M} c_i = \{C\} \cup \{F\} = \{C, F\}$$

and its complementary Δ is:

$$\Delta = (\alpha \cup \beta \cup \gamma \cup \delta \cup \varepsilon) \setminus \Gamma = \{A, B, D, E\}$$

The *Intersection map* is given by:

$$Imap = \begin{array}{c} \\ \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{array} \begin{array}{c} \begin{array}{ccccc} \alpha & \beta & \gamma & \delta & \varepsilon \end{array} \\ \begin{pmatrix} p_{1\alpha} & p_{1\beta} & p_{1\gamma} & p_{1\delta} & p_{1\varepsilon} \\ p_{2\alpha} & p_{2\beta} & p_{2\gamma} & p_{2\delta} & p_{2\varepsilon} \\ p_{3\alpha} & p_{3\beta} & p_{3\gamma} & p_{3\delta} & p_{3\varepsilon} \\ p_{4\alpha} & p_{4\beta} & p_{4\gamma} & p_{4\delta} & p_{4\varepsilon} \end{pmatrix} \end{array}$$

where:

$$\delta_1 = \{A\} = \begin{cases} \delta_1 \cap \alpha = \{A\} \cap \{A, B, C, D\} \neq \emptyset \implies p_{1\alpha} = 1 \\ \delta_1 \cap \beta = \{A\} \cap \{B, D\} = \emptyset \implies p_{1\beta} = 0 \\ \delta_1 \cap \gamma = \{A\} \cap \{A, E, D\} \neq \emptyset \implies p_{1\gamma} = 1 \\ \delta_1 \cap \delta = \{A\} \cap \{A, E, F\} \neq \emptyset \implies p_{1\delta} = 1 \\ \delta_1 \cap \varepsilon = \{A\} \cap \{A, D\} \neq \emptyset \implies p_{1\varepsilon} = 1 \end{cases}$$

$$\delta_2 = \{B\} = \begin{cases} \delta_2 \cap \alpha \neq \emptyset \implies p_{2\alpha} = 1 \\ \delta_2 \cap \beta \neq \emptyset \implies p_{2\beta} = 1 \\ \delta_2 \cap \gamma = \emptyset \implies p_{2\gamma} = 0 \\ \delta_2 \cap \delta = \emptyset \implies p_{2\delta} = 0 \\ \delta_2 \cap \varepsilon = \emptyset \implies p_{2\varepsilon} = 0 \end{cases}$$

$$\delta_3 = \{D\} = \begin{cases} \delta_3 \cap \alpha \neq \emptyset \implies p_{3\alpha} = 1 \\ \delta_3 \cap \beta \neq \emptyset \implies p_{3\beta} = 1 \\ \delta_3 \cap \gamma \neq \emptyset \implies p_{3\gamma} = 1 \\ \delta_3 \cap \delta = \emptyset \implies p_{3\delta} = 0 \\ \delta_3 \cap \varepsilon \neq \emptyset \implies p_{3\varepsilon} = 1 \end{cases}$$

$$\delta_4 = \{E\} = \begin{cases} \delta_4 \cap \alpha = \emptyset \implies p_{4\alpha} = 0 \\ \delta_4 \cap \beta = \emptyset \implies p_{4\beta} = 0 \\ \delta_4 \cap \gamma \neq \emptyset \implies p_{4\gamma} = 1 \\ \delta_4 \cap \delta \neq \emptyset \implies p_{4\delta} = 1 \\ \delta_4 \cap \varepsilon = \emptyset \implies p_{4\varepsilon} = 0 \end{cases}$$

According to these values, the *Intersection map* is:

$$Imap = \begin{array}{c} \\ \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{array} \begin{array}{c} \begin{array}{ccccc} \alpha & \beta & \gamma & \delta & \varepsilon \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

There are no replicated rows, so that $S_{shared} = \{\delta_1, \delta_2, \delta_3, \delta_4\}$.

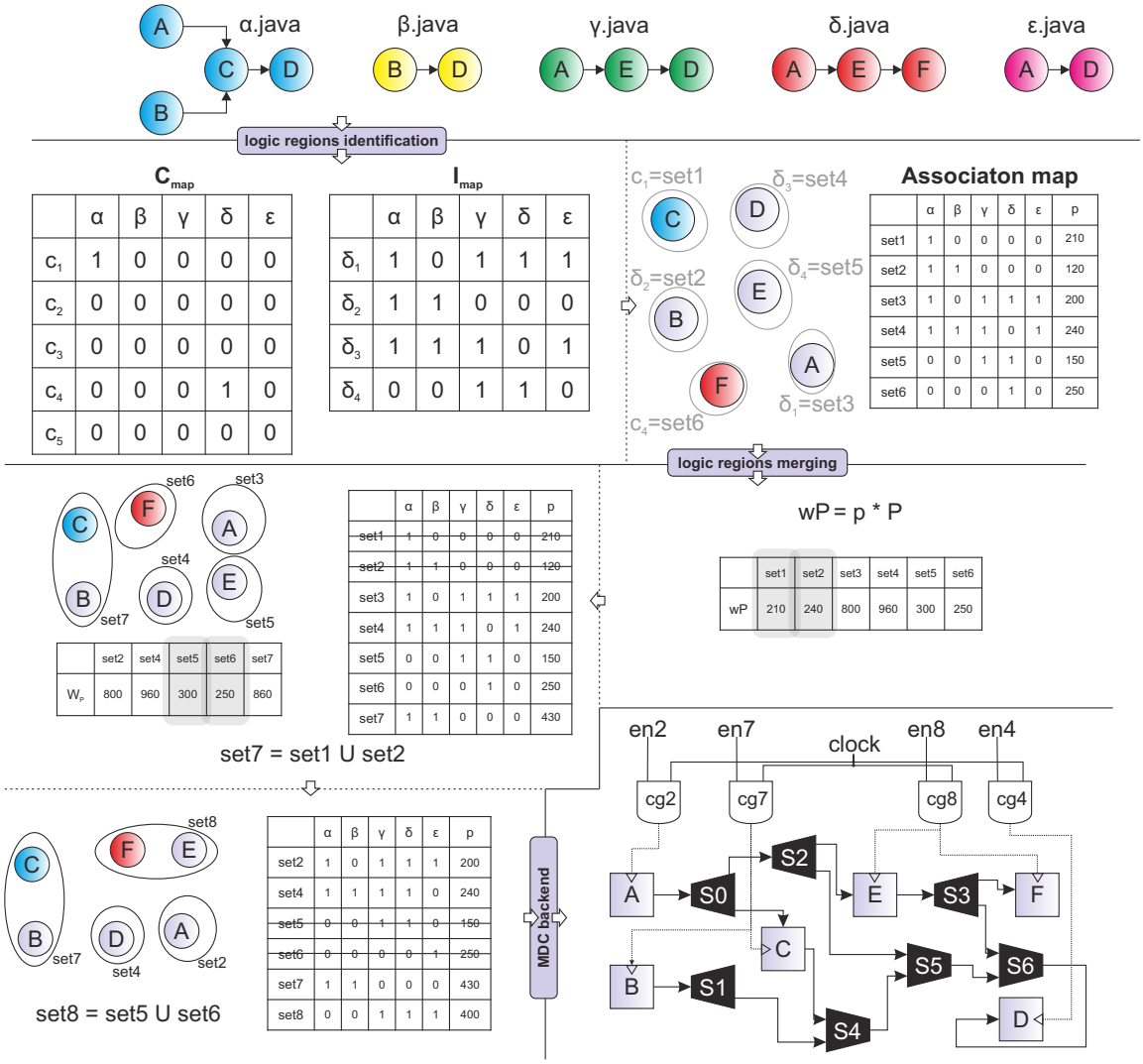


Figure 6.1: Dynamic Level Power Management: a step-by step example.

Figure 6.1 depicts the identified 6 logic regions and their respective *Association map*, as formally derived above. For completeness, the resulting *Association map* is also reported in Table 6.1.

Generally speaking, having so few logic regions their merging would not be required. Nevertheless, for the sake of completeness, let's assume the need of reducing their amount from 6 to 4. In this example, to maximise power reduction, we adopted Eq. 6.5: *set1* and *set3* present the smallest *wP* values so that they are the perfect candidates for merging. In the *Association map set1* and *set3* are removed and *set7=set1 U set3* is inserted. Exploiting again Eq. 6.5, it is possible to identify the next two candidates: *set5* and *set6* are removed and *set8=set5 U set6* is defined. Table 6.2 depicts the logic regions *Association map* after the merging process.

6.2 Approach assessment

This section is meant to assess the results achievable with the proposed dynamic power management methodology. At first the targeted scenarios will be briefly described (Section

Set	α	β	γ	δ	ϵ
set1	1	0	0	0	0
set2	1	1	0	0	0
set3	1	0	1	1	1
set4	1	1	1	0	1
set5	0	0	1	1	0
set6	0	0	0	1	0

Table 6.1: *Association map* of the found logic regions before their merging.

Set	α	β	γ	δ	ϵ
set2	1	1	0	0	0
set4	1	1	1	0	1
set7	1	1	0	0	0
set8	0	0	1	1	1

Table 6.2: *Association map* of the found logic regions after their merging.

6.2.1) and then the benefits of adopting the proposed logic regions identifications process analyse along with the effectiveness of implementing a standard coarse grained clock gating technique basing on the identified regions (Section 6.2.2).

6.2.1 Application scenarios and designs under test

Generally speaking the targeted scenario is the same adopted in [71] and already presented in Chapter 3. The main computing core of a reconfigurable hardware accelerator for image processing has been assembled, implementing in hardware several computing kernels (one at a time). These kernels belong to four different applications in the mobile multi-media domain:

- Spatial Anti-Aliasing (UC1), to correct the distortion effects of an image downsampled violating the Nyquist constraint;
- Zoom (UC2), to scale an image by a given zooming factor interlacing pixels of the original image with other pixels coming from adaptive interpolations of the neighbouring ones;
- Motion Estimation (UC3), to detect movements between two successive frames of a video sequence;
- Deblocking Filter (UC4), to remove image distortions coming from compression processes, like blocking and ringing.

Five different scenarios are going to be assessed: the four above-mentioned applications stand-alone (i.e. the co-processor will switch among the kernels of just one specific application, UC1-UC4) and a multi-application case (i.e. the co-processor will switch among the ensemble of kernels of all the provided applications, UC5). Please note that, as described in detail in Chapter 3 (see Figure 3.14), the kernels have some overlapping actors but involve also different non shared resources that should be turned off when the corresponding kernel is not under execution.

Different designs are going to be assembled and compared:

- *static* - the co-processor is assembled without the adoption of the MDC approach.
- *MDC 2.0* - the co-processor is assembled with MDC 2.0 disabling the dynamic power management extension.
- *MDC 2.0_LP* - the co-processor is assembled with MDC 2.0 enabling the dynamic power management extension.

6.2.2 Results discussion

This section is intended to assess the designs under test described in the previous section. In particular, synthesising the reconfigurable computing core on ASIC the benefits of the MDC low power extension will be highlighted. Furthermore some preliminary numbers derived prototyping the system on FPGA will be provided.

ASIC low power extension analysis

This Section focuses on the benefits that the logic regions identification methodology may lead to. The comparison between power aware systems (*MDC_LP*) and not power aware ones (*MDC*) is presented to show that just a little overhead in terms of hardware resources is required to achieve, on the other hand, severe power savings. Moreover, *MDC_LP* platforms result extremely performing also with respect to those systems where automatic power management methodologies implemented by the adopted commercial synthesiser are applied.

Using the presented logic regions identification methodology, MDC dynamic power manager found, respectively, 9 different logic regions for UC1, 13 for UC2, 2 for UC3, 15 for UC4 and 36 for UC5. The results are obtained by synthesis trials with a fixed operating frequency of 200 MHz. Table 6.3 shows the marginal area overhead that is present on ASIC, due to the introduction of the AND based clock gating cells, in all the *MDC_LP* designs.

App.	MDC [mm^2]	MDC_LP [mm^2]	%
UC1	0.1378	0.1381	+0.24
UC2	0.1194	0.1197	+0.33
UC3	0.0205	0.0206	+0.33
UC4	0.1858	0.1862	+0.25
UC5	0.3259	0.3270	+0.33

Table 6.3: MDC versus MDC_LP: area occupancy and overhead percentage due to the low power extension.

With respect to the dynamic power consumption, for a more accurate evaluation, we decided to take into account the real switching activity of the system. Table 6.4 to Table 6.8 report the achieved results executing each single kernel over different platforms:

- the *MDC* platform (see column *MDC* in Table 6.4 to Table 6.8);
- the *MDC* platform synthesised using the clock gating low power design feature provided by SoC Encounter (see column *MDC_auto* in Table 6.4 to Table 6.8);
- the *MDC_LP* platform (see column *MDC_LP* in Table 6.4 to Table 6.8).

Kernel	MDC	MDC_auto	MDC_LP	% LP vs. 2.0	% LP vs. auto
sorter	6.44	5.72	0.76	-88.19	-86.70
min_max	6.36	5.64	0.34	-94.62	-93.93
rgb2ycc	7.08	5.95	3.62	-48.92	-39.18
ycc2rgb	7.31	5.95	3.63	-50.32	-42.22
k_abs	6.36	5.54	0.26	-96.00	-95.40
corr	6.58	5.55	2.05	-68.83	-63.06

Table 6.4: MDC versus MDC_LP: UC1 - per kernel dynamic power consumption [mW].

Kernel	MDC	MDC_auto	MDC_LP	% LP vs. 2.0	% LP vs. auto
min_max	6.04	5.09	0.34	-93.99	-92.68
k_abs	5.63	4.59	0.25	-95.49	-95.22
sbwlabel	5.85	4.72	3.30	-43.53	-29.95
chgb	5.74	4.59	1.29	-77.49	-71.81
cubic_conv	6.05	4.99	1.30	-78.45	-73.90
median	5.65	4.59	1.46	-74.24	-68.32
cubic	5.65	4.62	1.36	-75.85	-70.46

Table 6.5: MDC versus MDC_LP: UC2 - per kernel dynamic power consumption [mW].

Kernel	MDC	MDC_auto	MDC_LP	% LP vs. 2.0	% LP vs. auto
k_nbit	1.03	1.07	0.23	-78.08	-79.00
smear_kernel	1.09	0.97	0.88	-19.04	-8.67

Table 6.6: MDC versus MDC_LP: UC3 - per kernel dynamic power consumption [mW].

No matter of the reference scenario, the proposed methodology provides superior performance with respect to the non clock gated one. This result (see column *% LP vs baseline* in Table 6.4 to Table 6.8) was absolutely expected and it is in line with the state of the art. The results, presented in the tables, show also that the MDC dynamic power manager is always capable of offering extremely better results with respect to the automatic clock gating implemented by the synthesiser (see column *% LP vs auto* in Table 6.4 to Table 6.8). Our methodology, acting at the logic regions level (on an ensemble of resources together) rather than at the registers level, is capable of saving a huge amount of clock tree power.

Table 6.9, considering UC1-UC4, provides the activation frequencies of each single kernel[3] along with the number of actors per kernel. Performing dynamic power consumption analysis, taking into consideration the switching activity coupled to the kernel activation frequency, the impressive power saving capabilities that the *MDC_LP* is able to provide are quite understandable. Let's consider for example UC1: it is composed of 6 different kernels and each of them involves a totally different amount of actors. The *min_max* kernel is composed of one single actor. Clearly each time this kernel is executed the rest of the design (the 97% of the UC1 total area) is completely inactive, eventually dissipating useless clock net related

[3]Numbers have been extracted while profiling the applications stand-alone, processing a 64 by 64 RGB picture for UC1 and UC4, a 32 by 32 BW picture for UC2 and 18 by 14 BW frames for UC3.

Kernel	MDC	MDC_auto	MDC_LP	% LP vs. 2.0	% LP vs. auto
min_max	8.84	7.83	0.39	-95.60	-95.03
filter	8.92	7.57	3.25	-63.54	-57.06
clip	8.84	7.82	0.55	-93.83	-93.03
inner_kernel	8.85	7.70	1.57	-82.27	-79.61
mdiv_kernel	8.85	7.88	1.14	-87.16	-85.57
k_sign	8.84	7.88	0.25	-97.17	-96.82
rgb2yuv	9.58	8.16	3.99	-58.35	-51.09
yuv2yrgb	9.87	8.28	4.03	-59.23	-51.38

Table 6.7: MDC versus MDC_LP: UC4 - per kernel dynamic power consumption [mW].

Kernel	MDC	MDC_auto	MDC_LP	% LP vs. 2.0	% LP vs. auto
sorter	14.25	11.51	0.96	-93.26	-91.65
min_max	14.19	11.65	0.59	-95.82	-94.91
rgb2ycc	14.94	11.99	3.96	-73.49	-66.96
ycc2rgb	15.22	12.30	4.03	-73.51	-67.21
k_abs	14.20	11.11	0.49	-96.53	-95.56
corr	14.37	11.39	2.24	-84.43	-80.36
sbwlabel	14.41	11.77	3.55	-75.36	-69.85
chgb	14.28	11.52	1.50	-89.49	-86.97
cubic_conv	14.60	11.99	1.55	-89.38	-87.07
median	14.21	11.67	1.70	-88.01	-85.40
cubic	14.23	11.59	1.61	-88.72	-86.15
filter	14.21	11.42	3.37	-76.28	-70.48
clip	14.18	11.58	0.74	-94.77	-93.60
inner_kernel	14.20	11.54	1.76	-87.60	-84.73
mdiv_kernel	14.20	11.78	1.33	-90.64	-88.72
k_nbit	14.18	11.68	0.58	-95.94	-95.07
k_sign	14.18	11.68	0.44	-96.90	-96.23
smear_kernel	14.14	11.45	1.08	-92.36	-90.56
rgb2yuv	14.95	11.95	4.23	-71.72	-64.60
yuv2yrgb	15.21	12.34	4.27	-71.90	-65.36

Table 6.8: MDC versus MDC_LP: UC5 - per kernel dynamic power consumption [mW].

power. There comes the fact that it is possible to save more than the 90% of power consumption disabling all the design but the *min_max* kernel (when this latter is executed on the *MDC_LP* platform). Similar considerations can be applied to the other use-cases. In particular, the worst saving percentage achieved by the *MDC 2.0_LP* is related to UC3 when the *smear_kernel* is executed (see Table 6.7). This kernel occupies the 80% of the UC3 design area, so that switching off the remaining of the design is not sufficient to guarantee large benefits in terms of saving percentages, which reaches just the 8.67%.

App.	kernel	actors	occurrences	%UC
UC1	sorter	2	18659	8.9
	min_ max	1	14864	3.0
	rgb2ycc	19	256	45.2
	ycc2rgb	18	225	37.2
	k_ abs	1	124366	1.6
	corr	10	2321	22.5
UC2	min_ max	1	15142	3.3
	k_ abs	1	70045	1.7
	sbwlabel	17	966	43.3
	chgb	7	3072	15.8
	cubic_ conv	6	341	17.0
	median	9	1253	19.6
	cubic	10	1496	17.8
UC3	k_nbit	1	8	17.5
	smear_ kernel	5	4	79.7
UC4	min_max	1	32806	2.1
	filter	13	2235	29.2
	clip	2	346	3.1
	inner_ kernel	9	1108	12.3
	mdiv_ kernel	7	346	9.3
	k_ sign	1	346	0.5
	rgb2yuv	22	256	34.4
	yuv2yrgb	21	256	28.5

Table 6.9: Actor count and activation per kernel.

Besides the considerations provided per kernel, also the dynamic power consumption per executed application can be reported. Estimations can be determined as the sum of the single kernel dissipation values weighted for their occurrences within the execution of each single application. They depend on the composition and the behaviour of the different use-cases. Table 6.10 presents the saving percentages achievable for the considered use-cases. UC3, being composed of just two kernels highly unbalanced (*smear_kernel* occupies the 80% of the area and is executed half of the time the *k_nbit* is) presents the worst saving percentages. In all the other scenarios the proposed methodology allows reaching more than the 90% of saving.

For the sake of completeness, please notice that Table 6.10 does not provide any information about UC5. Without a real execution trace, reporting in details how many times each single application is executed in the multi-application scenario, accurate considerations cannot be provided.

Kernel	MDC	MDC_auto	MDC_LP	% LP vs. 2.0	% LP vs. auto
UC1	6.41	5.65	0.56	-91.21	-90.02
UC2	5.64	4.60	0.37	-93.40	-91.90
UC3	8.86	7.74	0.65	-57.66	-57.21
UC4	1.05	1.04	0.44	-92.67	-91.61

Table 6.10: MDC versus MDC_LP: per use-case dynamic power consumption [mW].

FPGA preliminary data

The processor/co-processor environment has also been implemented on a Virtex5 330 device by Xilinx. The preliminary results we have achieved are in line with [71], but there are some differences with respect to the above-mentioned power considerations derived for the ASIC implementations.

The resources occupancy of the UC5 design is reported in Table 6.11. As for the other considered cases, the MDC approach, by exploiting resource sharing through the introduction of Sboxes, leads to a saving in terms of all the considered resources of the target board. Also the considerations about the clock gating overhead done for ASIC are reflected in FPGA, since *MDC 2.0* and *MDC 2.0 LP* requires basically the same amount of resources.

	Ratio [%]	Slices [%]	LUTs [%]	RAM [%]
static	16	7	15	19
MDC 2.0	9	3	7	6
MDC 2.0 LP	9	3	7	6

Table 6.11: FPGA implementation: UC5 - Resource occupancy on the Virtex5 330.

Table 6.12 depicts the dynamic power consumption related to the execution of each single kernel on the multi-application scenario (UC5). The advantages of the proposed methodology are still appreciable for most of the kernels. However, power saving percentages are strongly reduced with respect to the ASIC implementation (with a peak of 25.94% for the *k_sign* kernel). Generally speaking, the smaller is the kernel, the greater is the saved power. In some cases (the *ycc2rgb*, *filter* and *rgb2yuv* kernels), the *MDC LP* implementation shows the worst results if compared to the *MDC*.

Further investigations will be required but the cause could be related to mapping problems. In any FPGA device, the given system has to fit within the form factor of the targeted device. The utilisation of the BUFG modules, which have fixed positions around the board, forces the design to spread within the chip. This may lead to longer wires and, as a consequence, to additional power consumption. For these reasons, the proposed approach has to be improved to take into account also target specific considerations through dedicated technology aware models.

Please note that, MDC 2.0 identified 36 logic regions for UC5 exceeding the available BUFCGE cells provided for the Virtex5 330. So that, the second phase of the logic regions identification process has been necessary. The above-mentioned results adopt *CF1* (minimisation of the number of PEs per logic region) to select the candidate logic regions to be merged. If the MDC structural profiler is enabled and low level target specific power feedback is provided, *CF2* may lead to better results.

Kernel	MDC	MDC_LP	% LP vs. 2.0
sorter	219.23	171.90	-21.59
min_max	233.54	177.47	-24.01
rgb2ycc	279.27	260.60	-6.69
ycc2rgb	280.79	299.63	+6.71
k_abs	219.25	163.00	-25.66
corr	225.21	219.87	-2.37
sbwlabel	232.99	230.07	-1.25
chgb	225.21	184.07	-18.27
cubic_conv	228.43	191.71	-16.07
median	221.69	205.48	-7.31
cubic	228.67	203.23	-11.13
filter	219.49	226.62	+3.25
clip	219.29	165.08	-24.72
inner_kernel	220.80	182.22	-17.47
mdiv_kernel	220.25	180.60	-18.00
k_nbit	218.61	162.83	-25.51
k_sign	219.65	162.67	-25.94
smear_kernel	234.26	204.83	-12.53
rgb2yuv	280.25	304.36	+8.60
yuv2yrgb	278.90	265.33	-4.87

Table 6.12: MDC versus MDC_LP: UC5 - per kernel dynamic power consumption in mW. Implementation on the Virtex5 LX 330.

6.3 Chapter Remarks

In this chapter a dynamic power management methodology that extends the baseline functionalities of MDC has been presented. In particular, the proposed methodology is able to automatically identify disjointed homogeneous logic regions yet on the high level models, starting from the dataflow functionalities specifications to be implemented over the reconfigurable platform. These regions have been successfully adopted to implement a coarse grained clock gating based power saving technique.

The potentials of the dynamic power manager extension have been proven on a processor/co-processor environment, where the reconfigurable computing core of the co-processing unit has been assembled to speed-up four different applications in the image/video processing domain. Results assessment, on ASIC, demonstrated that the dynamic power manager can lead to the 90% of power saving, in highly variable multi-functional scenarios without any area penalty.

MDC power saving capabilities showed to be extremely performing also with respect to the automatic power management methodologies implemented by a commercial synthesiser. This result is due to the fact that the proposed automatic mapping methodology allows to tackle the power management problem at a coarse grained level, rather than at a fine grained one, guaranteeing better percentages of saving. The FPGA prototype confirmed the applicability of the methodology also targeting such a kind of devices. Nevertheless, further investigations and models adjustments will be required to successfully tackle FPGAs in the practice.

Note that the dynamic power manager is fully compliant, besides with the MDC baseline functionality, also with all the additional features of the design suite. In particular, if coupled with the structural profiler set with the area/power effort, the dynamic power manager can provide a two level strategy allowing both structural and behavioural optimisation of the generated system under the power aspect.

List of Publications Related to the Chapter

Journal papers

- F. Palumbo, C. Sau and L. Raffo, *Coarse-grained reconfiguration: dataflow-based power management*, in IET Computers and Digital Techniques, Volume 9, Issue 1, January 2015, p. 36-48.
- C. Sau, N. Carta, L. Raffo and F. Palumbo, *Early Stage Automatic Strategy for Power-Aware Signal Processing Systems Design*, in Journal of Signal Processing Systems, Volume 82, Issue 3, March 2016, pp 311-329.
- F. Palumbo, T. Fanni, C. Sau and P. Meloni, *Power-Awarness in Coarse-Grained Reconfigurable Multi-Functional Architectures: a Dataflow Based Strategy*, in Journal of Signal Processing Systems, (online February 2016), to appear.

Conference papers

- F. Palumbo, C. Sau and L. Raffo, *Power-awarness in coarse-grained reconfigurable designs: A dataflow based strategy*, in Proceedings of the 2014 IEEE Workshop on Signal Processing Systems (SiPS), Belfast (Northen Ireland), October 2014.

Chapter 7

Co-processor Generator

Hardware accelerators/co-processors are integrated in heterogeneous multiprocessor system-on-chip platforms to provide speed-up and reduced power consumption. Co-processors can be used in parallel to execute computational intensive kernels, improving the overall system efficiency by means of highly specialized circuits, and can be triggered on demand to save power. Very often these requirements have to be coupled with flexibility, since different functionalities or several variants of a certain baseline behaviour are needed. Coarse grained (CG) reconfigurable co-processors can tackle these issues by supporting a set of different execution contexts while guaranteeing high performance and efficiency in terms of consumption (see Chapter 2).

The key problem with hardware co-processors, including CG reconfigurable ones, is normally the design time required to implement, debug and deploy them. Custom hardware designs imply to describe the micro-architecture at a register transfer level (RTL), such as in Verilog or VHDL languages. All the details about the applications/kernels to be accelerated have to be known and a lot of effort is spent in coding and debugging. Additionally, reconfigurability has the cost of introducing a further issue due to the applications mapping, as discussed in Chapter 2. Research efforts have been undertaken to automate the application mapping process [17]. Dimensioning the underlying structure and efficiently mapping multi-functional applications over it are key challenges.

Hardware-software co-design strategies allowed for the automatic generation of dedicated computational units from the software source code and/or high level specifications, possibly grouping and merging together several functionalities. Automatic synthesis tools allow users to be totally unaware of the underlying hardware but, to enable efficient use of the co-processors, these latter have to be provided with libraries/drivers for accessing/configuring them. During the thesis work, an automated design flow that makes the Multi-Dataflow Composer (MDC) able to design and managing whole CG reconfigurable co-processors has been developed. The co-processor generator extension provides both the hardware IP and the software drivers, featuring two different levels of coupling with the host processor.

In the rest of this chapter, initially it will be described in detail the automated flow capable of generating the co-processing layer around the reconfigurable datapath provided by the MDC baseline functionality (Section 7.1). Then, the co-processor generator will be evaluated through some experimental trials, revealing the effectiveness of the proposed flow (Section 7.2). The chapter will be concluded with final considerations and future improvements discussion (Section 7.3).

7.1 Co-processors Synthesis Flow

Starting from a set of high level specifications describing the applications/kernels to be accelerated, the aim of the proposed process is to relieve designers from the tedious RTL coding and from the complex mapping process in multi-functional environments, reducing debugging time and positively affecting time to market. Input specifications are provided according to the dataflow formalism of the MPEG Reconfigurable Video Coding standard (MPEG-RVC) [5].

Despite MPEG-RVC has been standardised only a few years ago, several research groups have been active on this topic and many tools have been developed. The Open RVC-CAL Compiler (Orcc) is the most important one [7]. It generates, from the given high level specification, the related source code for different platforms: hardware, software or mixed. Moreover, it transforms the input models into Java directed graphs that can be fed to other tools, such as Xronos [23] (capable of High Level Synthesis for Xilinx FPGAs). Orcc, Xronos and MDC (as it will be detailed in Section 7.1.1) are part of the proposed co-processor synthesis flow. Before the work presented in this thesis, in the MPEG-RVC domain the automatic characterization of hardware acceleration units has never been addressed. Moreover, even though this work originated within MPEG-RVC studies, the applicability of the proposed approach is not limited to such a target scenario. The reconfigurable computing core inside the synthesised co-processors, practically implementing the computation tasks, is obtained using the MDC tool. Therefore, it is compliant with the communication protocol adopted by the MPEG-RVC actors. However, being such a protocol a very simple FIFO based scheme, only minor adjustments or simple wrappers would be required to enable support for different communication mechanisms.

The proposed flow is meant to automatically compose, synthesise and deploy runtime reconfigurable co-processors. Figure 7.1 provides an overview of the design flow, where different steps are identified as described hereafter.

- *High Level Specification Composition* - This step (Section 7.1.1) is meant to process the input high level dataflow specifications, provided as XDF networks[1]. Such specifications are combined in a unique multi-dataflow one, integrating all the functionalities to be supported by the reconfigurable co-processor.

- *Reconfigurable Computing Core Definition* - This step (Section 7.1.2) is responsible of creating the reconfigurable computing core to be inserted within the co-processor. Starting from the multi-dataflow specification, its actors are synthesized to be included within an HDL component library. Then, this latter and the high level multi-dataflow specification are used to create the HDL description of the CG reconfigurable computing core, in charge of executing the different kernels.

- *Co-processor Hw-Sw Specification* - This step (Section 7.1.3) is responsible of extracting from the multi-dataflow specification all the configuration information associated to the different input kernels in order to define the co-processor interface and its drivers. At this stage, designers are required to choose the processor/co-processor communication scheme to be supported.

- *Co-processor Deployment* - The final step (Section 7.1.4) is meant to finalize the IP, integrating the reconfigurable computing core with the co-processor interface. Moreover, all the files necessary to instantiate the co-processor as a standard Xilinx IP are derived.

[1] The XML Dataflow Format (XDF) is an XML dialect, used to describe the networks, standardized by MPEG.

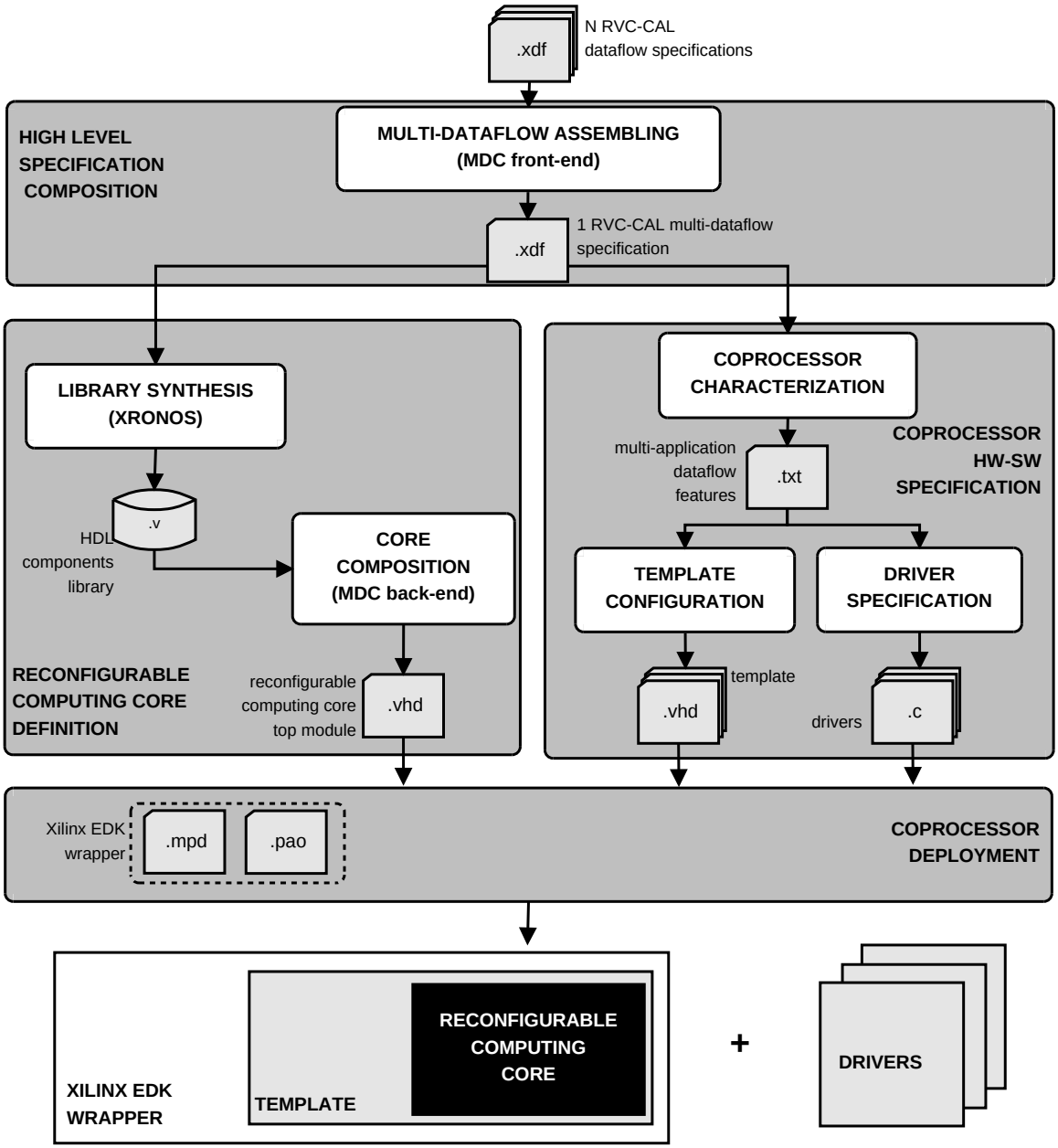


Figure 7.1: Co-processor generator design flow overview.

7.1.1 High Level Specification Composition

In this section we describe how the high level specification of the multi-functional system is derived. The RVC-CAL dataflow specifications of the kernels to be accelerated are assembled by the front-end of the MDC tool in a unique multi-dataflow specification. MDC was conceived for the automatic creation of stand-alone multi-dataflow CG reconfigurable platforms. Its final goal is minimizing the hardware resources among datapaths implementing different specifications. It exploits Orcc to acquire the input specifications and merges them together, two at a time, into a unique multi-dataflow XDF, where common actors are shared.

The Java intermediate representations produced by Orcc are simple directed graphs, so that the datapath merging process performed by MDC can be easily adjusted to deal with different dataflow models. Indeed, allowing the portability of the proposed methodology to

different domains.

7.1.2 Reconfigurable Computing Core Definition

In this section the definition of the CG reconfigurable computing core, to be inserted within the co-processor, is described. The starting point is the MPEG-RVC multi-dataflow specification integrating all the kernels to be accelerated. The output is the HDL description of the reconfigurable computing core. Two different phases compose this step: the *Library Synthesis* and the *Core Composition*.

Library Synthesis

This phase produces the HDL components library. Xronos [23] synthesises the functional units to be included within the reconfigurable computing core of the co-processors. It is a framework for generating RTL descriptions from dataflow or sequential applications. It basically targets FPGA platforms. Xronos supports the MPEG-RVC standard; therefore, if a different scenario is targeted, a different synthesizer would be required.

Core Composition

Final composition and configuration of the computing core of the co-processors is performed by exploiting the MDC back-end, which takes the HDL components library and the high level multi-dataflow specification to create the HDL description of the reconfigurable computing core. The MDC back-end used the CG approach to map actors to components. Actors belonging to different kernels and presenting the same functionality are mapped to the same component. Single-cycle reconfiguration of the CG substrate is guaranteed by the adoption of low overhead switching elements placed at the crossroads between the different paths of data. They are simple combinatorial multiplexers able to differentiate the flows of data around the shared components. These switching elements are programmed by means of configuration Look-Up Tables that keep trace of the correct data routing to preserve kernels correctness. Their content is determined by the MDC front-end during the composition phase.

Changing the reference scenario would not imply any change in the composition process itself, but switching elements with a different external interface would be necessary. Fortunately, the MDC back-end is fed with the specification of the actors communication protocol and generates accordingly the switching elements.

7.1.3 Co-processor Hw-Sw Specification

This step is meant to configure both the co-processing layer and its drivers according to the characteristics of the kernels to be accelerated. The starting point is the MPEG-RVC multi-dataflow specification; whereas, the outputs are the specialized HDL description (template) of the front-end and the back-end of the co-processor and its configuration and management drivers. Three different phases compose this step.

Co-processor Characterization

The footprint of the reconfigurable computing core is extracted from its high level specification to properly configure the co-processing layer. The footprint, in this context, is intended

Table 7.1: Influence of the extracted features on the template and drivers generation.

feature	template	drivers
# of I/O	modules instantiation	-
I/O size	modules dimensioning	-
I/O pattern	memory dimensioning	-
app ID	-	specify the issued computation
app I/O	-	specify which configuration registers have to be loaded

as the features exposed by the multi-dataflow specification to the external world. The reconfigurable computing core, once assembled, is treated as a black box with a well-defined I/O interface characterized by:

1. the number of I/O ports;
2. the depth of the data channel of each I/O port;
3. the token pattern of each I/O port.

In addition, since we are dealing with a multi-functional scenario, to properly characterize the co-processor also some configuration information have to be retrieved, as:

1. the ID whereby each input kernel is encoded,
2. the I/O ports required by the different input kernels.

I/O footprint and configuration information are exploited to specify the co-processor and its drivers. They are all stored within a .txt file, which content is summarized in Table 7.1. This table reports also how these parameters influence the co-processor template and drivers configuration.

Template Configuration

Once collected the features of the generated multi-dataflow specification, the co-processing layer can be configured. This latter, together with the reconfigurable computing core, constitutes the coprocessor itself. The co-processing layer is mainly composed of a configurable template, hereafter called Template Interface Layer (TIL). The TIL integrates a bank of configuration registers, to store the desired configuration, one (or more) front-end(s), to load data into the reconfigurable computing core, and one (or more) back-end(s), to read the computed data from the reconfigurable computing core.

As a matter of fact this structure would not change even if a different actors communication scheme is adopted. Moreover, since the reconfigurable computing core obeys to the MPEG-RVC FIFO based communication protocol, which relies on a very simple handshake scheme, the possibility of adopting the proposed TIL in different target scenarios is quite feasible. It would require fairly limited adjustments on the load/store mechanisms or the adoption of simple wrappers around the reconfigurable computing core to translate its communication scheme and its interface into those supported by the MPEG-RVC standard.

Generally speaking, co-processing units can have different degrees of coupling with the host processor. A loosely coupled co-processor is far from the processor, it is typically accessible through the system bus and it is affected by medium/high communication latencies for both control and data transfers. A tightly coupled co-processor is closed to the processor,

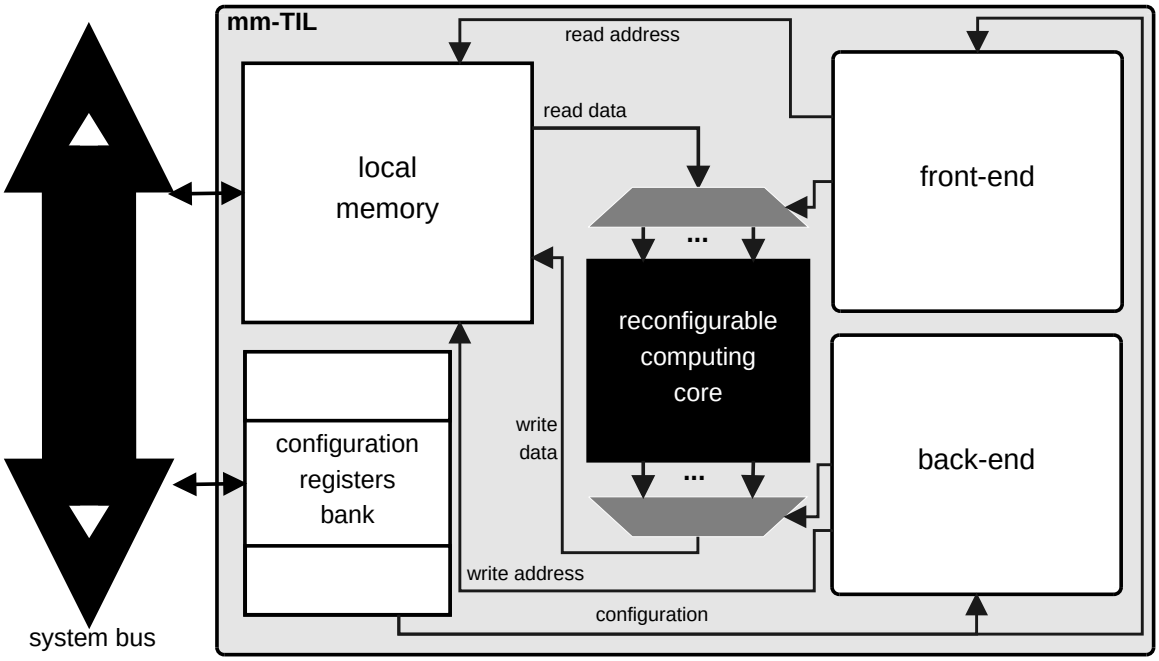


Figure 7.2: Architecture of the memory-mapped Template Interface Layer (mm-TIL).

it has a dedicated full-duplex link and it often shares with the processor high level memories. A loosely coupled co-processor can be easily adopted in different contexts, since it is connected to a generic system bus. On the contrary, it is hard to extend the adoption of a tightly coupled co-processor to different systems, since it has dedicated links and memory accesses.

In order to provide different levels of applicability and coupling, we have developed two TILs:

- a memory-mapped loosely coupled one, accessible through the system bus as a memory-mapped IP;
- a stream-based tightly coupled one, accessible through different full duplex links, one for each I/O port.

Both TILs are fully supported by the proposed tool chain. The user has only to specify the preferred option (memory-mapped or stream-based) and the corresponding ready-to-use co-processing unit is automatically generated.

Memory-Mapped TIL The memory-mapped TIL (mm-TIL) is the easiest adaptable version of the automatically generated co-processing layer. In general, it is possible to attach the mm-TIL to all the kinds of memory-mapped system bus, if provided with a simple wrapper.

The architecture of the mm-TIL is shown in Figure 7.2. Its main blocks are: one local memory, the configuration registers bank, one front-end and one back-end. The local memory contains all the data to be processed by the co-processor and the computed results. It has to be fully written by the processor before the co-processor execution phase and it has to be fully read once the co-processor has completed the task. A dedicated address range of the processor is reserved to the local memory. The configuration registers bank is the entity in charge of storing the configuration of the co-processor. The configuration includes,

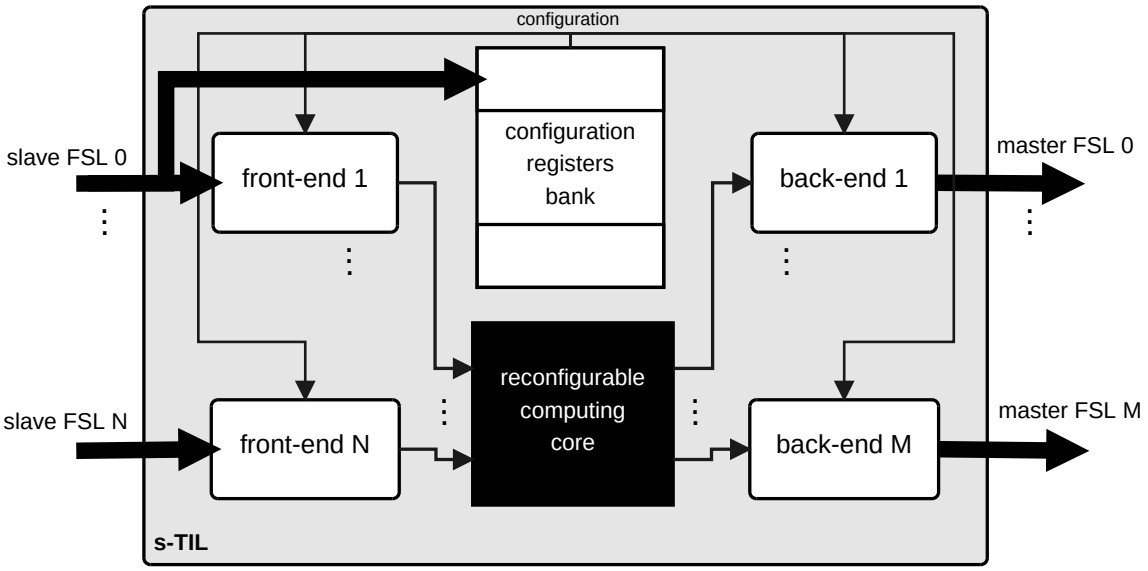


Figure 7.3: Architecture of the stream-based Template Interface Layer (s-TIL).

a part the ID of the kernel to be executed, the base address, the data number for each I/O port and, only for the inputs, the burst size. The base address is the first address of the local memory where the data have to be read/written. The data number is the amount of data to read/write from/to the local memory. The burst size is the number of data that have to be sent together to the reconfigurable computing core. The configuration registers bank holds an address range separated from the local memory one.

The front-end is responsible for the data transfer from the local memory to the reconfigurable computing core. Its execution flow can be divided in three different phases. At the beginning, an input port to load the data is chosen by means of a round robin priority policy. Then, the address of the local memory, where the related data is stored, is generated. Finally, a burst of data from the memory to the reconfigurable computing core is transferred. The front-end iterates on the described cycle until all the data have been transferred to the reconfigurable computing core.

The transfer of the processed data from the reconfigurable computing core to the local memory is performed by the back-end. Basically, the back-end architecture and execution flow are the same of the front-end ones, managing the output ports and dealing with memory writes instead of reads.

Stream-Based TIL The stream-based TIL (s-TIL) is adopted to realize tightly coupled co-processing units. It is based on a Xilinx proprietary point-to-point connection protocol called Fast Simplex Link (FSL). The FSL is a very fast communication channel provided with FIFO memories, typically used within the Xilinx environment to connect host processors with hardware co-processors.

The s-TIL requires a different FSL channel for each I/O port. With respect to the mm-TIL the s-TIL can boost the co-processor performance, not only leveraging on a faster communication channel, but also accessing in parallel different ports of the reconfigurable computing core. This parallelism can be fully exploited only if the host processor is able to read and write different FSL channels at the same time or if the co-processing unit is connected to different host processors.

Figure 7.3 depicts the s-TIL architecture. The main blocks are: the configuration registers bank, one front-end per input port and one back-end per output port. The configuration

registers bank, as in the mm-TIL, saves the co-processor configuration. It is not mapped in a specific address range of the host processor, but it is supplied by one of the FSL links. On this latter, the control data transfers are differentiated from the processing ones by means of the control bit of the FSL protocol. When the control bit is high the data on the FSL are stored into the registers bank; whereas, when it is low the data are forwarded to the reconfigurable computing core. The base address configuration is no longer necessary: data come from (and have to be written) directly into a FSL channel instead of a local memory.

In the s-TIL the front-end and back-end aims are the same as in the mm-TIL: the front-end transfers data from an input FSL bus to the reconfigurable computing core and the back-end transfers data from this latter to an output FSL bus. In the s-TIL each I/O port is served by a different front-end or back-end, thus managing a point-to-point transfer of data from/to one of the FSL buses to/from one of the reconfigurable computing core ports (the port selection phase is no longer needed). No local memory is available; therefore, there is no address generation phase and the loading/storing phase of the different I/O ports is parallel.

Driver Specification

The proposed flow, beyond providing the RTL code of the co-processing unit, gives also the software drivers to ease its use. In particular the features related to the configuration, retrieved during the characterization phase (see Section7.1.3), are used to specify the drivers. The driver suite is split in two different levels:

- low level drivers (LLDs): manage the processor/co-processor communication;
- high level drivers (HLDs): deal with the application issues, masking the system configuration complexity.

Listing 7.1: Low level co-processor driver example.

```
...
// memory-mapped config writing
# define mmCOPR_write_config_<port_name>(int value) \
        Xil_Out32(COPR_REGS_BASE_ADDR
        +<port_name>_OFFSET, value);

// stream config writing
# define sCOPR_write_config_<port_name>(int value) \
        cputfsl(value, FSL_ID_<port_name>);
...
```

LLDs encapsulate the system macros for writing/reading memory locations and for putting/getting data to/from the FSL buses into generic I/O functions (see Listing 7.1). The HLDs, starting from these generic I/O functions, manage the co-processor configuration and data transfer. For each I/O port of the reconfigurable computing core, a configuration word is written into the proper co-processor register. Then, for each input port involved in the current computation, a specific HLD primitive is used to send the data to be computed from the host processor to the co-processor. At last, as the processor receives an interrupt from the co-processing unit, a specific HLD primitive is adopted to read back the results into the processor from the output ports (see Listing 7.2).

Listing 7.2: High level memory-mapped co-processor driver example

```
...
// configuration
```

```
int config_<port_name> =
        (size_burst_<port_name><<(SIZE_ADDR+SIZE_CNT))
        | (size_<port_name><<SIZE_ADDR)
        | base_addr_<port_name>;
mmCOPR_write_config_<port_name>(config_<port_name>);

// data sending
for(int i=0;i<size_<port_name>;i++)
        mmCOPR_write_mem(base_addr_<port_name>
                + i*4, data_<port_name>[i]);

// data receiving
for(int i=0;i<size_<port_name>;i++)
        data_<port_name>[i]=
                mmCOPR_read_mem(base_addr_<port_name>
                + i*4);
...
```

No changes would be required if a different actors communication scheme is adopted. Indeed, it does not affect at all the processor/co-processor communication scheme. Therefore, no changes at the drivers generation level are necessary to target a different scenario.

7.1.4 Co-processor Deployment

In this final step the peripheral is integrated and deployed as a standard Xilinx IP. The inputs are the HDL description of reconfigurable computing core, its front-end and its back-end; whereas, the output is the resulting Xilinx IP comprehensive of software drivers. The co-processor is encapsulated through a Xilinx EDK wrapper to be included within the Xilinx IP catalog. The wrapper is mainly composed of two files: the Microprocessor Peripheral Definition (MPD), used to define the peripheral interface within the system, and the Peripheral Analyze Order (PAO), needed to associate and properly order all the peripheral source files.

As well as in the driver case, no changes would be required if a different actors communication scheme is adopted. The final system integration phase is not affected by the internal actors communication scheme within the reconfigurable computing core, since the TIL template masks it.

7.2 Experimental Results

In this section all the trials performed to evaluate the effectiveness of the proposed tool chain are presented. In Section 7.2.1 a JPEG codec use case will be presented while in Section 7.2.2 the related experimental data will be shown and discussed.

7.2.1 JPEG Codec Use Case

To evaluate the co-processor generator, a real use case involving a JPEG codec application has been adopted. The codec is based on the simple profile ITU-T.IS 1091 standard. The JPEG encoder and decoder have been modelled as MPEG-RVC dataflow networks[2]. The encoder is compliant with the YCrCb 4:2:0 format, it requires two chrominance macroblocks (8x8 pixels

[2]The reference models have been retrieved from Orc-apps, a library of open-source applications described in a dynamic dataflow programming way, using the MPEG-RVC standard. Available at https://github.com/orcc/orc-apps

Table 7.2: *mm-sys* vs. *s-sys*: drivers memory footprint and configuration time.

system	*mm-sys*	*s-sys*
drivers size [B]	1904	1348
config time [ms]	1.600	0.632

blocks as defined by the standard), one for Cr and one for Cb, every four luminance ones. The coding process can be divided in 3 phases: the 2-dimensional Forward DCT (FDCT), which shows the frequency footprint of the input image, the quantization, which emphasises the lowest frequency components and cuts the highest ones, and the Huffman entropy encoding, which compresses the image through a variable length coding algorithm. The decoding process is specular to the encoding one: firstly the Huffman decoding algorithm is applied, then the de-quantization flats the frequency components of the decoded image and lastly the Inverse DCT (IDCT) brings the image back to the spatial domain.

Since the FDCT and the IDCT show common processing steps, like the multiplication of a given macroblock by a coefficient matrix, their MPEG-RVC models involve some common actors. The proposed design flow has been exploited to automatically develop two co-processors, a loosely coupled one and a tightly coupled one, able to perform both the JPEG codec operations.

7.2.2 Experimental Data

The MPEG-RVC networks of the JPEG codec constitute the starting point of the proposed design flow. They are given as inputs to the MDC front-end in order to generate the related multi-dataflow specification. This latter has 4 inputs and 3 outputs whose data channel depths vary from 8 to 32 bits and whose token patterns are always less than or equal to 64. The involved input encoder employs 3 input ports and 1 output port, while the decoder has 1 input and 2 outputs. The proposed flow automatically assembles the ready-to-use JPEG codec co-processor, suitable for Xilinx environments. The assessed systems are composed as follow:

- the memory-mapped system (*mm-sys*) involves the memory-mapped co-processor, a host processor and a system bus;
- the stream-based system (*s-sys*) involves the stream-based co-processor, a host processor, a system bus and as many FSL buses as needed by the use case (7 in the considered JPEG application).

The FSL buses, exploited by the stream-based co-processor, are an overhead that have to be evaluated. In both cases, the host processor is the Xilinx Microblaze soft-core and the system bus is the Xilinx Processor Local Bus. Both systems have been implemented targeting a Xilinx Virtex-5 330 FPGA board.

To highlight the differences between the co-processors drivers, Table 7.2 shows the memory occupancy and configuration time of the *mm-sys* and the *s-sys*. This latter seems to be more efficient: its tightly coupling behaviour allows skipping completely the address configuration phase, which in turn means a smaller memory footprint and a shorter configuration time (less information have to be accessed and managed).

Table 7.3 depicts the resources usage of the two systems. In terms of Slice Regs they have similar occupancies, while dealing with Slice LUTs the *mm-sys* needs 23% less resources than the *s-sys*. Considering the memory blocks (BRAMs) available on the board, the *s-sys* has a

Table 7.3: *mm-sys* vs. *s-sys*: resources occupancy and maximum achievable frequency.

resource (available)	*mm-sys* **(%)**	*s-sys* **(%)**
Slice Regs (207360)	10380(5)	9511(4)
Slice LUTs (207360)	36563(17)	47607(22)
BUFGs (32)	8(25)	8(25)
BRAMs (288)	112(25)	117(40)
frequency [MHz]	57.8	57.8

Table 7.4: *mm-sys* and *s-sys*: execution performance with respect to an off-the-shelf general purpose processor (*arm*).

encoding

system	freq [MHz]	cycles	time [ms]
mm-sys	57.8	350098	6.06
s-sys	57.8	169581	2.93
arm	666.67	47022699	70.53

decoding

system	freq [MHz]	cycles	time [ms]
mm-sys	57.8	504016	8.72
s-sys	57.8	262182	4.54
arm	666.67	67525734	101.29

higher demand (+4%) with respect to the *mm-sys*. The maximum achievable frequency of both the systems is the same, 57.8 MHz, since the critical path is determined by the computational units within the reconfigurable computing core. These latter have been automatically synthesized from the high level description of the actors and negatively affect the frequency numbers.

The higher resource occupancy of the *s-sys* is justified by the performance results, shown in Table 7.4, referring to the encoding and the decoding of a sample image. The *s-sys* execution latency numbers are halved, thanks to the parallel loading and storing of the I/O ports. In particular the *s-sys* saves the 52% of time during the encoding phase and the 48% during the decoding phase. Table 7.4 reports also the latencies of the adopted applications running on an ARM Cortex-A9 core (*arm*). Starting from the same MPEG-RVC networks of the JPEG codec, Xronos is able to generate the corresponding C++ code, which has been used in these trials. Despite a lower operating frequency, both the *mm-sys* and the *s-sys* are able to strongly speed-up the execution of the JPEG codec with respect to the *arm*.

The experimental results clearly showed the peculiarities of the two available kinds of co-processing unit: the memory-mapped one provides a lower resources footprint and less constraints on the system infrastructure, since it can be easily adapted to every system bus. On the contrary, the stream-based one is target dependent, but it guarantees higher performances.

7.3 Chapter Remarks

CG reconfigurable co-processors constitute a valid solution to challenge embedded application scenarios, where flexibility and high efficiency are required. However their development is complicated due to the huge platform design/debug effort and to the difficulty of mapping different applications over the same platform. In order to manage these issues MDC with the co-processor generator additional feature is able to automatically generate CG reconfigurable co-processors starting from the high level specifications of the embedded applications. A similar feature, before the present thesis work, was never presented within the MPEG-RVC scenario.

On the one hand, leveraging on Xronos and MDC, the developed tool chain composes a reconfigurable computing core able to implement all the input applications. On the other, it provides a co-processing layer that makes the core available as a co-processing unit in a processor/co-processor environment. The tool chain outcome is a ready-to-use co-processor IP, providing both the RTL code and its software drivers. In order to address different design contexts, two kinds of co-processing layers are provided: a loosely coupled memory-mapped one and a tightly coupled stream-based one. The former guarantees an easier adaptivity to different environments, while the latter provides better performance. At the state of the art it has not been found any approach able to derive reconfigurable co-processing units with different degree of coupling starting directly from the isolated descriptions of the wanted functionalities.

The proposed approach is portable also in different target scenarios. If the reconfigurable computing core is created according to a different dataflow model of computation, simple wrappers can be used to adapt the FIFO based load/store scheme supported by the proposed architectural templates. The flow is portable, with minor adjustments, even if the input high level specifications are changed. As we discussed, the datapath merging process, the computing core composition, the drivers definition and the overall processor/co-processor system integration will require practically no adjustments. The architectural template should be adjusted to the new actors communication scheme or wrappers would be required, but unfortunately Xronos will have to be substituted with a different actors synthesizer or the HDL components library will have to be manually created.

The tool chain has been evaluated on a JPEG codec application, by implementing both the memory-mapped and the stream-based configurations. It is important to note that the aim of the assessment was not determining which coupling better suits to accelerate a JPEG codec. Targeting a Xilinx FPGA technology, we intended to highlight the peculiarities of the two kinds of co-processors and their easy integration and use on the vendor environment.

List of Publications Related to the Chapter

Conference papers

- C. Sau and F. Palumbo, *Automatic Generation of Dataflow-Based Reconfigurable Co-processing Units*, in Proceedings of the 2014 Conference on the Design and Architectures for Signal and Image Processing (DASIP), Madrid (Spain), November 2014.

- C. Sau, L. Fanni, P. Meloni, L.Raffo and F. Palumbo, *Reconfigurable Coprocessors Synthesis in the MPEG-RVC Domain*, in Proceedings of the 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Mayan Riviera (Mexico), December 2015.

Chapter 8

Concluding remarks

Reconfigurable computing seems to be a good solution for the challenging world of portable multi-media embedded systems. It allows devices to achieve a trade-off between flexibility and performance. In particular, coarse grained reconfigurability guarantees quick functionality switches while limiting the supported flexibility, resulting very convenient in real-time application specific contexts. The main drawback common to all the reconfigurable architectures is related to the typically huge cost required by their design and management.

The objectives of this thesis work were mainly related to the development of a powerful methodology for aiding designers in the complex and time consuming process of developing reconfigurable architectures. To achieve these objectives, an automated tool for the design and management of coarse grained reconfigurable systems has been studied and developed. The tool, Multi-Dataflow Composer (MDC), starting from the specification of the desired functionalities (modelled as dataflow networks), can derive the RTL description of the corresponding coarse grained reconfigurable substrate able to implement all the input functionalities. The tool has been assembled from a previous exploration version. The new MDC is integrated within the MPEG-RVC framework in order to provide a fully automated tool chain where the only effort that the designers have to spend is on the input functionalities modelling. This tool chain, leveraging on other MPEG-RVC tools, such as Orcc, Turnus and Xronos, allowed the automation of the whole reconfigurable systems design flow, including resource mapping, optimisation and hardware design, as expected by one of the thesis secondary objectives.

The addressing of others secondary goals of the thesis led to the definition of a whole design suite around the baseline MDC features. In particular, MDC has been extended to support: structural profiling, dynamic power management and co-processor generation. Structural profiling has been provided to address the secondary goal related to some application specific contexts where strict design constraints required a smarter generation of the reconfigurable substrate. The developed profiling flow is in charge of optimising the system structure by exploiting low level information coming from a priori synthesis of the isolated input functionalities specifications. It has been able to assign a very accurate estimation, in terms of the main electronics design metrics, to the solutions belonging to the MDC design space.

One important secondary objective of the thesis is related to the power issue that in reconfigurable architectures is hardened by the intrinsic resource redundancy of the substrate. Indeed, in such a kind of systems, resources that are not involved in the current computation can easily lead to useless consumption. The developed dynamic power management strat-

egy is capable of identifying, yet on the system high level model, the logic regions, each one grouping resources that are always active or inactive together in the computation. Basing on the identified regions, power saving techniques can be applied for shutting down the unused resources. These techniques are applied at a coarse level, thus resulting extremely efficient, even more than lower level approaches, as it has been demonstrated with clock gating.

The last thesis secondary goal was related to the system prototyping and integration that had not to be a bottleneck for the proposed flow, limiting the possible benefits. Here comes the co-processor generator, an additional feature that makes the reconfigurable datapath, provided by the baseline MDC functionality, a real hardware accelerator ready-to-use for a rapid system prototyping. The co-processor generator supports two different levels of coupling with the host processor, a memory-mapped loosely coupling and a stream-based tightly one, letting the designer the choice of the best option basing on its specific requirements.

All the developed features of the proposed design suite have been validated and evaluated on real use cases belonging to the image and video processing application field. Nevertheless, the proposed approach demonstrated to be independent by the considered application field, resulting beneficial also in different contexts, such as neural signal processing for implantable devices. For each considered use case, several designs have been developed and implemented, in some cases also prototyped. The trials involved both FPGA and ASIC final targets, demonstrating the benefits of the proposed methodology in both cases.

8.1 Future Works

The MDC design suite that has been presented during this thesis work could be enhanced under different aspects. Dealing with the baseline MDC functionality, it could be improved by supporting also different algorithms to solve the datapath merging problem, besides the current empirical one, in order to provide a wider set of alternatives to the designer (Chapter 3).

The automated tool chain (Chapter 4) could be completed by integrate also additional entities or features belonging to the already existent tools, besides the MDC ones that are already compliant with the whole tool chain. For instance, the tool chain could integrate the low power features supported by Turnus, able to implement a multi-frequency technique on a given dataflow network, and by Xronos, capable of performing clock gating on the actors according to the related FIFOs state. Obviously, in this case further elaborations would be required in order to fit with the reconfigurability of the developed system, since Turnus and Xronos are basically mono-dataflow tools.

With respect to the structural profiler (Chapter 5), in future the additional dynamic dissipated power metric could be considered for the evaluation of the explored design space, in order to provide a more complete system optimisation, eventually allowing a third design effort choice besides area/power and frequency. Furthermore, heuristic methods exploiting useful information about the design space could be adopted making it possible the addressing of more complex scenarios involving a bigger number of considered input dataflows and actors.

Regarding the dynamic power manager (Chapter 6) future improvements, power gating support explorations (leveraging on the common power format) have already been done but, since the results are still under investigation, the corresponding discussion has not been included in this thesis. Moreover, the integration of MDC with Orcc and the other MPEG-RVC compliant tools could be exploited to: deploy automatic power aware co-processing units, create a set of model based strategies for early stage power estimation and management and, finally, define an automatic flow for power efficient multi-standard decoders.

Lastly, considering the co-processor generator (Chapter 7), the proposed approach could be improved by providing high level analysis methods to automatically identify the kernels to be accelerated and to decide, possibly for each of them separately, the proper level of coupling. This will lead to multi/hybrid accelerator environments. Another improvement to the co-processor generator could be the support for additional levels of coupling with the processor or for different target environments, besides the current Xilinx one.

Bibliography

[1] Compute Unified Device Architecture (CUDA). In *http://www.nvidia.com/object/cuda_home_new.html*. [cited at p. 6]

[2] H.261 : Video codec for audiovisual services at p x 64 kbit/s. In *http://www.itu.int/rec/T-REC-H.261*. [cited at p. 5]

[3] H.265 : High efficiency video coding. In *http://www.itu.int/rec/T-REC-H.265*. [cited at p. 5]

[4] Independent JPEG Group (IJG). In *http://www.ijg.org/*. [cited at p. 71]

[5] ISO/IEC 23001-4 (2011).MPEG video technologies—Part 4: Codec configuration representation. [cited at p. 29, 110]

[6] ISO/IEC 23002-4 (2010).MPEG video technologies—Part 4: Video tool library. [cited at p. 29]

[7] Open RVC-CAL compiler (Orcc). In *http://orcc.sourceforge.net/*. [cited at p. 30, 78, 94, 110]

[8] OpenCL - the open standard for parallel programming of heterogeneous systems. In *https://www.khronos.org/opencl/*. [cited at p. 6]

[9] A. Agarwal, S.K. Mathew, S.K. Hsu, M.A. Anders, H. Kaul, F. Sheikh, R. Ramanarayanan, S. Srinivasan, R. Krishnamurthy, and S. Borkar. A 320mv-to-1.2v on-die fine-grained reconfigurable fabric for dsp/media accelerators in 32nm cmos. volume 53, pages 328–329, 2010. cited By 19. [cited at p. 11, 16, 18, 20]

[10] Altera. *FPGA Coprocessing Evolution: Sustained Performance Approaches Peak Performance*, 2009. [cited at p. 10]

[11] Altera. *Nios II C2H Compiler User Guide*, 2009. [cited at p. 23]

[12] Altera. *Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs*, 2010. [cited at p. 9]

[13] Altera. *Nios II Classic Processor Reference Guide*, 2015. [cited at p. 19]

[14] Altera. *Stratix 10 Product Table*, 2015. [cited at p. 10, 19, 20]

[15] M. Amagasaki, R. Yamaguchi, M. Koga, M. Iida, and T. Sueyoshi. An embedded reconfigurable IP core with variable grain logic cell architecture. *Int. J. Reconfig. Comp.*, 2008:180216:1–180216:14, 2008. [cited at p. 14, 15, 20]

[16] G. Ansaloni, P. Bonzini, and L. Pozzi. Egra: A coarse grained reconfigurable architectural template. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(6):1062–1074, June 2011. [cited at p. 14, 18, 19, 20]

[17] G. Ansaloni, K. Tanimura, L. Pozzi, and N. Dutt. Integrated kernel partitioning and scheduling for coarse-grained reconfigurable arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(12):1803–1816, December 2012. [cited at p. 109]

[18] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem 1. *Algorithmica*, 29(4):610–637, 2001. [cited at p. 25]

[19] C. Beaumin, O. Sentieys, E. Casseau, and A. Carer. A coarse-grain reconfigurable hardware architecture for rvc-cal-based design. In *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*, pages 152–159, October 2010. [cited at p. 30]

[20] L. Benini and G. de Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. [cited at p. 34]

[21] N.W. Bergmann, S.K. Shukla, and J. Becker. Quku: A dual-layer reconfigurable architecture. *ACM Trans. Embed. Comput. Syst.*, 12(1s):63:1–63:26, March 2013. [cited at p. 13, 17, 19, 20]

[22] E. Bezati, S.C. Brunet, M. Mattavelli, and J.W. Janneck. Coarse grain clock gating of streaming applications in programmable logic implementations. In *Electronic System Level Synthesis Conference (ESLsyn), Proceedings of the 2014*, pages 1–6, May 2014. [cited at p. 34]

[23] E. Bezati, M. Mattavelli, and J.W. Janneck. High-level synthesis of dataflow programs for signal processing systems. In *Image and Signal Processing and Analysis (ISPA), 2013 8th International Symposium on*, pages 750–754, September 2013. [cited at p. 23, 30, 48, 110, 112]

[24] S.S. Bhattacharyya, E.F. Deprettere, and B.D. Theelen. Dynamic dataflow graphs. In *Handbook of Signal Processing Systems*, pages 905–944. Springer, 2013. [cited at p. 61]

[25] S.S. Bhattacharyya, J. Eker, J.W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet. Overview of the mpeg reconfigurable video coding framework. *J. Signal Process. Syst.*, 63(2):251–263, May 2011. [cited at p. 29]

[26] G. Bilsen, M. Engels, R. Lauwereins, and J.A. Peperstraete. Cyclo-static data flow. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 5, pages 3255–3258 vol.5, May 1995. [cited at p. 28]

[27] B. Bond, K. Hammil, L. Litchev, and S. Singh. Fpga circuit synthesis of accelerator data-parallel programs. In *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, pages 167–170, May 2010. [cited at p. 23]

[28] A. Cappelli, A. Lodi, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis, L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, M. Toma, and R. Guerrieri. Xisystem: a xirisc-based soc with a reconfigurable io module. In *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pages 196–593 Vol. 1, February 2005. [cited at p. 10]

[29] N. Carta, P. Meloni, G. Tuveri, D. Pani, and L. Raffo. A custom mpsoc architecture with integrated power management for real-time neural signal decoding. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 4(2):230–241, June 2014. [cited at p. 32]

[30] N. Carta, C. Sau, F. Palumbo, D. Pani, and L. Raffo. A coarse-grained reconfigurable wavelet denoiser exploiting the multi-dataflow composer tool. In *Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference on*, pages 141–148, October 2013. [cited at p. 39]

[31] N. Carta, C. Sau, D. Pani, F. Palumbo, and L. Raffo. A coarse-grained reconfigurable approach for low-power spike sorting architectures. In *Neural Engineering (NER), 2013 6th International IEEE/EMBS Conference on*, pages 439–442, November 2013. [cited at p. 39]

[32] S.M. Carta, D. Pani, and L. Raffo. Reconfigurable coprocessor for multimedia application domain. *Journal of VLSI signal processing systems for signal, image and video technology*, 44(1):135–152, 2006. [cited at p. 21, 39]

[33] S. Casale-Brunet, E. Bezati, C. Alberti, M. Mattavelli, E. Amaldi, and J.W. Janneck. Partitioning and optimization of high level stream applications for multi clock domain architectures. In *Signal Processing Systems (SiPS), 2013 IEEE Workshop on*, pages 177–182, October 2013. [cited at p. 34]

[34] S. Casale-Brunet, E. Bezati, M. Mattavelli, M. Canale, and J. W. Janneck. Execution trace graph analysis of dataflow programs: Bounded buffer scheduling and deadlock recovery using model predictive control. In *Design and Architectures for Signal and Image Processing (DASIP), 2014 Conference on*, pages 1–6, October 2014. [cited at p. 62]

[35] S. Casale-Brunet, A. Elguindy, E. Bezati, R. Thavot, G. Roquier, M. Mattavelli, and J.W. Janneck. Methods to explore design space for mpeg rmc codec specifications. *Image Commun.*, 28(10):1278–1294, November 2013. [cited at p. 61]

[36] S. Casale-Brunet, M. Mattavelli, and J.W. Janneck. Buffer optimization based on critical path analysis of a dataflow program design. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 1384–1387, May 2013. [cited at p. 61, 62]

[37] S. Casale-Brunet, M. Mattavelli, and J.W. Janneck. Turnus: A design exploration framework for dataflow system design. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 654–654, May 2013. [cited at p. 30, 61]

[38] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, June 2002. [cited at p. 8, 10, 18, 22]

[39] J. B. Dennis. First version of a data flow procedure language. In *Programming Symposium, Proceedings Colloque Sur La Programmation*, pages 362–376, London, UK, UK, 1974. Springer-Verlag. [cited at p. 26]

[40] K. Desnos and J. Heulot. Pisdf: Parameterized & interfaced synchronous dataflow for mpsocs runtime reconfiguration. In *1st Workshop on MEthods and TOols for Dataflow PrOgramming (METODO)*, 2014. [cited at p. 29]

[41] J. Eker and J. Janneck. Cal language report. Technical report, Tech. Rep. ERL Technical Memo UCB/ERL, 2003. [cited at p. 30]

[42] S. Fernando, M. Wijtvliet, C. Nugteren, A. Kumar, and H. Corporaal. (as)2: Accelerator synthesis using algorithmic skeletons for rapid design space exploration. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 305–308, San Jose, CA, USA, 2015. EDA Consortium. [cited at p. 23, 24]

[43] J. Fowers, J.Y. Kim, D. Burger, and S. Hauck. A scalable high-bandwidth architecture for lossless compression on fpgas. In *The 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines*. IEEE - Institute of Electrical and Electronics Engineers, May 2015. [cited at p. 10]

[44] F. Garzia, W. Hussain, and J. Nurmi. Crema: A coarse-grain reconfigurable array with mapping adaptiveness. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 708–712, August 2009. [cited at p. 13, 17, 18, 20]

[45] K. Gilles. The semantics of a simple language for parallel programming. *In Information Processing*, 74:471–475, 1974. [cited at p. 26]

[46] J. Gorin, M. Wipliez, F. Prêteux, and M. Raulet. Llvm-based and scalable mpeg-rvc decoder. *J. Real-Time Image Process.*, 6(1):59–70, March 2011. [cited at p. 31]

[47] J.L. Gross and J. Yellen. *Graph Theory and Its Applications, Second Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2005. [cited at p. 62]

[48] R.W. Hartenstein. Coarse grain reconfigurable architecture (embedded tutorial). In *Proceedings of ASP-DAC 2001, Asia and South Pacific Design Automation Conference 2001, January 30-February 2, 2001, Yokohama, Japan*, pages 564–570, 2001. [cited at p. 11, 14, 15, 20]

[49] P. Heysters, G. Smit, and E. Molenkamp. A flexible and energy-efficient coarse-grained reconfigurable architecture for mobile systems. *The Journal of Supercomputing*, 26(3):283–308, 2003. [cited at p. 12, 15, 16, 17, 18, 20]

[50] J.E. Hopcroft, J.D. Ullman, and A.V. Aho. *Data structures and algorithms*, volume 175. Addison-Wesley Boston, MA, USA:, 1983. [cited at p. 77]

[51] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu. Elastic cgras. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 171–180, New York, NY, USA, 2013. ACM. [cited at p. 16, 30]

[52] Z. Huang and S. Malik. Managing dynamic reconfiguration overhead in systems-on-a-chip design using reconfigurable datapaths and optimized interconnection networks. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '01, pages 735–, Piscataway, NJ, USA, 2001. IEEE Press. [cited at p. 25]

[53] W. Hussain, F. Garzia, T. Ahonen, and J. Nurmi. Designing fast fourier transform accelerators for orthogonal frequency-division multiplexing systems. *J. Signal Process. Syst.*, 69(2):161–171, November 2012. [cited at p. 13, 18, 20]

[54] Q. Inoue, K.and Zhao, Y. Okamoto, H. Yosho, M. Amagasaki, M. Iida, and T. Sueyoshi. A variable-grain logic cell and routing architecture for a reconfigurable ip core. *ACM Trans. Reconfigurable Technol. Syst.*, 4(1):5:1–5:24, December 2010. [cited at p. 15, 20]

[55] Chiu J.-C., Chou Y.-L., and Lin R.-B. The multi-context reconfigurable processing unit for fine-grain computing. *Journal of Information Science and Engineering*, 24(3):965–979, 2008. cited By 4. [cited at p. 11, 16, 18, 20]

[56] J.W. Janneck, I.D. Miller, and D.B. Parlour. Profiling dataflow programs. In *Multimedia and Expo, 2008 IEEE International Conference on*, pages 1065–1068, June 2008. [cited at p. 61]

[57] J.W. Janneck, I.D. Miller, D.B. Parlour, G. Roquier, M. Wipliez, and M. Raulet. Synthesizing hardware from dataflow programs: An mpeg-4 simple profile decoder case study. In *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 287–292, October 2008. [cited at p. 62]

[58] B. Jeff. Advances in big. little technology for power and energy savings. *ARM White Paper*, 2012. [cited at p. 33]

[59] C. Kohn. *Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices*. Xilinx, January 2013. [cited at p. 11]

[60] V.V. Kumar and J. Lach. Highly flexible multimode digital signal processing systems using adaptable components and controllers. *EURASIP Journal on Applied Signal Processing*, 2006:73–73, 2006. [cited at p. 23]

[61] C.Y. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, September 1961. [cited at p. 45]

[62] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987. [cited at p. 28]

[63] E.A. Lee and T.M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995. [cited at p. 26]

[64] P. Mishra and N. Dutt. Architecture description languages for programmable embedded systems. *IEE Proceedings - Computers and Digital Techniques*, 152(3):285–297, May 2005. [cited at p. 23]

[65] T. Miyoshi, H. Kawashima, Y. Terada, and T. Yoshinaga. A coarse grain reconfigurable processor architecture for stream processing engine. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 490–495, September 2011. [cited at p. 14, 16, 17, 20]

[66] N. Moreano, G. Araujo, Zhining Huang, and S. Malik. Datapath merging and interconnection sharing for reconfigurable architectures. In *System Synthesis, 2002. 15th International Symposium on*, pages 38–43, October 2002. [cited at p. 25, 26]

[67] S. Neuendorffer, T. Li, and D. Wang. *Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries*. Xilinx, June 2014. [cited at p. 24]

[68] A. Niedermeier, J. Kuper, and G. Smit. Dataflow-based reconfigurable architecture for streaming applications. In *System on Chip (SoC), 2012 International Symposium on*, pages 1–4, October 2012. [cited at p. 12, 16, 17, 18, 20, 30]

[69] G. Palermo, C. Silvano, and V. Zaccaria. Multi-objective design space exploration of embedded systems. *J. Embedded Comput.*, 1(3):305–316, August 2005. [cited at p. 88]

[70] M. Palesi and T. Givargis. Multi-objective design space exploration using genetic algorithms. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, CODES '02, pages 67–72, New York, NY, USA, 2002. ACM. [cited at p. 88]

[71] F. Palumbo, D. Carta, N.and Pani, P. Meloni, and L. Raffo. The multi-dataflow composer tool: generation of on-the-fly reconfigurable platforms. *Journal of real-time image processing*, 9(1):233–249, 2014. [cited at p. 38, 52, 53, 54, 101, 106]

[72] K. Paul, C. Dash, and M.S. Moghaddam. remorph: A runtime reconfigurable architecture. In *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 26–33, September 2012. [cited at p. 12, 16, 18, 20]

[73] M. Pedram. *Power Aware Design Methodologies*. Kluwer Academic Publishers, Norwell, MA, USA, 2002. [cited at p. 34]

[74] J. Piat, S.S. Bhattacharyya, and M. Raulet. Interface-based hierarchy for synchronous data-flow graphs. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pages 145–150, October 2009. [cited at p. 29]

[75] R. Puri, L. Stok, and S.S. Bhattacharya. Keeping hot chips cool. In *Design Automation Conference, 2005. Proceedings. 42nd*, pages 285–288, June 2005. [cited at p. 32]

[76] Z.E. Rakossy, A.A. Aponte, and A. Chattopadhyay. Exploiting architecture description language for diverse ip synthesis in heterogeneous mpsoc. In *Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on*, pages 1–6, December 2013. [cited at p. 23]

[77] Z.E. Rakossy, T. Naphade, and A. Chattopadhyay. Design and analysis of layered coarse-grained reconfigurable architecture. In *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, pages 1–6, December 2012. [cited at p. 13, 16, 18, 20]

[78] R. Ren, J. Wei, E. Juarez, M. Garrido, C. Sanz, and F. Pescador. A pmc-driven methodology for energy estimation in rvc-cal video codec specifications. *Image Commun.*, 28(10):1303–1314, November 2013. [cited at p. 34]

[79] M. Rutten, O.P. Gangwal, J. van Eijndhoven, E. Jaspers, and E.J. Pol. Application design trajectory towards reusable coprocessors - mpeg case study. In *Embedded Systems for Real-Time Multimedia, 2004. ESTImedia 2004. 2nd Workshop on*, pages 33–38, September 2004. [cited at p. 23]

[80] A.G. Schmidt, N. Steiner, M. French, and R. Sass. Hwpmi: An extensible performance monitoring infrastructure for improving hardware design and productivity on fpgas. *Int. J. Reconfig. Comput.*, 2012:2:2–2:2, January 2012. [cited at p. 35]

[81] R. Schreiber, S. Aditya, B. Ramakrishna Rau, V. Kathail, S. Mahlke, S. Abraham, and G. Snider. High-level synthesis of nonprogrammable hardware accelerators. In *Application-Specific Systems, Architectures, and Processors, 2000. Proceedings. IEEE International Conference on*, pages 113–124, 2000. [cited at p. 23]

[82] H. Singh, L. Ming-Hau, L. Guangming, F.J. Kurdahi, N. Bagherzadeh, and E.M. Chaves Filho. Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *Computers, IEEE Transactions on*, 49(5):465–481, May 2000. [cited at p. 18, 20]

[83] N. Siret, I. Sabry, J.F. Nezan, and M. Raulet. A codesign synthesis from an mpeg-4 decoder dataflow description. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 1995–1998, May 2010. [cited at p. 30]

[84] G. Smaragdos, D.A. Khan, I. Sourdis, C. Strydis, A. Malek, and S. Tzilis. A dependable coarse-grain reconfigurable multicore array. In *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 141–150, May 2014. [cited at p. 12, 19, 20]

[85] G.J.M. Smit, P.M. Heysters, M.A.J. Rosien, and B. Molenkamp. Lessons learned from designing the montium-a coarse-grained reconfigurable processing tile. 2004. [cited at p. 16, 20]

[86] I. Sourdis, C. Strydis, A. Armato, C.S. Bouganis, B. Falsafi, G.N. Gaydadjiev, S. Isaza, A. Malek, R. Mariani, D.N. Pnevmatikatos, D.K. Pradhan, G.K. Rauwerda, R.M. Seepers, R.A. Shafik, K. Sunesen, D. Theodoropoulos, S. Tzilis, and M. Vavouras. Desyre: On-demand system reliability. *Microprocessors and Microsystems - Embedded Hardware Design*, 37(8-C):981–1001, 2013. [cited at p. 14, 19, 20]

[87] Cid C. de Souza, Andre M. Lima, Guido Araujo, and Nahri B. Moreano. The datapath merging problem in reconfigurable systems: Complexity, dual bounds and heuristic evaluation. *J. Exp. Algorithmics*, 10, December 2005. [cited at p. 24, 26]

[88] P. Sundararajan. *High Performance Computing Using FPGAs*. Xilinx, September 2010. [cited at p. 10]

[89] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, March 2002. [cited at p. 19, 20]

[90] Tensilica. *Xtensa Architecture and Performance*, September 2002. [cited at p. 19]

[91] R. Tessier and W. Burleson. Reconfigurable computing for digital signal processing: A survey. *J. VLSI Signal Process. Syst.*, 28(1-2):7–27, May 2001. [cited at p. 8]

[92] F. Thoma, M. Kuhnle, P. Bonnot, E.M. Panainte, K. Bertels, S. Goller, A. Schneider, S. Guyetant, E. Schuler, K.D. Müller-Glaser, and J. Becker. Morpheus: Heterogeneous reconfigurable computing. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 409–414, August 2007. [cited at p. 17, 18, 20]

[93] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, and P.Y.K. Cheung. Reconfigurable computing: architectures and design methods. *IEE Proceedings-Computers and Digital Techniques*, 152(2):193–207, 2005. [cited at p. 9, 11, 19]

[94] M. Wipliez, N. Siret, N. Carta, F. Palumbo, and L. Raffo. Design ip faster: Introducing the chigh-level language. *Design & Reuse*, 2013. [cited at p. 23]

[95] Xilinx. *LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c)*, April 2010. [cited at p. 19]

[96] Xilinx. *Partial Reconfiguration User Guide*, April 2012. [cited at p. 9]

[97] Xilinx. *MicroBlaze Processor Reference Guide*, May 2014. [cited at p. 23, 24]

[98] Xilinx. *Vivado Design Suite User Guide - High-Level Synthesis*, April 2014. [cited at p. 19]

[99] Xilinx. *7 Series FPGAs Overview*, May 2015. [cited at p. 10, 20]

[100] Xilinx. *UltraScale Architecture and Product Overview*, December 2015. [cited at p. 19]

[101] M. Yan, Z. Yang, L. Liu, and S. Li. Prodfa: Accelerating domain applications with a coarse-grained runtime reconfigurable architecture. In *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pages 834–839, December 2012. [cited at p. 15, 30]

[102] A.K.W. Yeung and J.M. Rabaey. A reconfigurable data-driven multiprocessor architecture for rapid prototyping of high throughput dsp algorithms. In *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, volume i, pages 169–178 vol.1, January 1993. [cited at p. 14, 18, 20]

[103] Y. Zhang, J. Roivainen, and A. Mammela. Clock-gating in fpgas: A novel and comparative evaluation. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages ‘584–590, 2006. [cited at p. 34, 94]

[104] M. Zhu, L. Liu, Shouyi Y., Y. Wang, W. Wang, and S. Wei. A reconfigurable multiprocessor soc for media applications. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2011–2014, May 2010. [cited at p. 17, 20]

List of Publications Related to the Thesis

Published papers

Journal papers

- F. Palumbo, C. Sau and L. Raffo, *Coarse-grained reconfiguration: dataflow-based power management*, in IET Computers and Digital Techniques, Volume 9, Issue 1, January 2015, pp 36-48. (Relation to Chapters 5 and 6)
- D. Pani, C. Sau, F. Palumbo and L. Raffo, *Computing Swarms for Self-Adaptiveness and Self-Organization in Floating-Point Array Processing*, in ACM Transactions on Autonomous and Adaptive Systems (TAAS), Volume 10, Issue 3, October 2015, pp 16:1-16:34.
- C. Sau, N. Carta, L. Raffo and F. Palumbo, *Early Stage Automatic Strategy for Power-Aware Signal Processing Systems Design*, in Journal of Signal Processing Systems, Volume 82, Issue 3, March 2016, pp 311-329. (Relation to Chapters 3 and 6)
- C. Sau, P. Meloni, L. Raffo, F. Palumbo, E. Bezati, S. Casale-Brunet and M. Mattavelli, *Automated Design Flow for Multi-Functional Dataflow-Based Platforms*, in Journal of Signal Processing Systems, (online August 2015), to appear. (Relation to Chapter 3)
- F. Palumbo, T. Fanni, C. Sau and P. Meloni, *Power-Awarness in Coarse-Grained Reconfigurable Multi-Functional Architectures: a Dataflow Based Strategy*, in Journal of Signal Processing Systems, (online February 2016), to appear. (Relation to Chapters 5 and 6)

Conference papers

- F. Palumbo, C. Sau and L. Raffo, *DSE and Profiling of Multi-Context Coarse-Grained Reconfigurable Systems*, in Proceedings of the 2013 8th International Symposium on Image and Signal Processing and Analysis (ISPA), Trieste (Italy), September 2013. (Relation to Chapter 5)
- N. Carta, C. Sau, F. Palumbo, D. Pani and L. Raffo, *A coarse-grained reconfigurable wavelet denoiser exploiting the Multi-Dataflow Composer tool*, in Proceedings of the 2013 Conference on Design and Architectures for Signal and Image Processing (DASIP), Cagliari (Italy), October 2013.

- N. Carta, C. Sau, D. Pani, F. Palumbo and L. Raffo, *A Coarse-Grained Reconfigurable Approach for Low-Power Spike Sorting Architectures*, in Proceedings of the 2013 6th International IEEE/EMBS Conference on Neural Engineering (NER), San Diego (California, USA), November 2013.

- C. Sau, L. Raffo, F. Palumbo, E. Bezati, S. Casale-Brunet and M. Mattavelli, *Automated design flow for coarse-grained reconfigurable platforms: An RVC-CAL multi-standard decoder use-case*, in Proceedings of the 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), Samos Island (Greece), July 2014. (Relation to Chapter 3)

- C. Sau and F. Palumbo, *Automatic Generation of Dataflow-Based Reconfigurable Coprocessing Units*, in Proceedings of the 2014 Conference on the Design and Architectures for Signal and Image Processing (DASIP), Madrid (Spain), November 2014. (Relation to Chapter 7)

- F. Palumbo, C. Sau and L. Raffo, *Power-awarness in coarse-grained reconfigurable designs: A dataflow based strategy*, in Proceedings of the 2014 IEEE Workshop on Signal Processing Systems (SiPS), Belfast (Northen Ireland), October 2014.(Relation to Chapters 5 and 6)

- T. Fanni, C. Sau, L. Raffo and F. Palumbo, *Automated Power Gating Methodology for Dataflow-Based Reconfigurable Systems*, in the Proceedings of the 12th ACM International Conference on Computing Frontiers, Ischia (Italy), May 2015.

- C. Sau, L. Fanni, P. Meloni, L.Raffo and F. Palumbo, *Reconfigurable Coprocessors Synthesis in the MPEG-RVC Domain*, in Proceedings of the 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Mayan Riviera (Mexico), December 2015. (Relation to Chapter 7)

- T. Fanni, C. Sau, P. Meloni, L. Raffo and F. Palumbo, *Power modelling for saving strategies in coarse grained reconfigurable systems*, in Proceedings of the 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Mayan Riviera (Mexico), December 2015.

Other scientific papers

- C. Sau, F. Palumbo and L. Raffo, *Profiling of Dataflow-Based Coarse-Grained Reconfigurable Platforms*, at the 2013 International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES), Fiuggi (Italy) July 2013.

- F. Palumbo, C. Sau and L. Raffo, *Coarse-Grained Reconfiguration: high-level dataflow-based power management strategies*, at the 2014 Riunione Annuale del Gruppo Elettronica (GE), Cagliari (Italy) June 2014.

Submitted papers

- S. Banik, A. Bogdanov, T. Fanni, C. Sau, L. Raffo, F. Palumbo and F. Regazzoni, *Adaptable AES Implementation with Power-Gating Support*, submitted to a conference, May 2016.

- T. Fanni, C. Sau, P. Meloni, L. Raffo and F. Palumbo, *Power and Clock Gating Modelling in Coarse Grained Reconfigurable Systems*, submitted to a conference, May 2016.

- C. Sau, T. Fanni, L. Raffo, F. Regazzoni, S. Banik, A. Bogdanov and F. Palumbo, *Multi-Profile Energy-Aware Reconfigurable AES*, submitted to a conference, August 2016.

Appendices

Appendix A: MDC Merging Agorithm

ALGORITHM 1: Main loop of the empiric merging algorithm adopted by MDC.

```
foreach DPN_i in input DPNs do
    G_m = G_i;
    if G_r = 0 then
        G_r = G_m
    else
        /* merging of the input and output ports                    */
        mergePorts;
        /* merging of the actor instances                           */
        mergeActorInstances;
        /* merging of the connections                               */
        mergeConnections;
    end
end
```

ALGORITHM 2: Merging of the ports step (*mergePorts* in the main algorithm) of the MDC empiric merging algorithm.

```
foreach v_m in V_m do
    if v_m is a port then
        foreach v_r in V_r do
            if L(v_m) = L(v_r) then
                map v_m in v_r
            end
        end
        if v_m is not mapped in V_r then
            add new v_r in V_r with L(v_r) = L(v_m)
        end
    end
end
```

ALGORITHM 3: Merging of the actor instances step (*mergeActorInstances* in the main algorithm) of the MDC empiric merging algorithm.

```
foreach v_m in V_m do
    if v_m is an actor instance then
        foreach v_r in V_r do
            if L(v_m) = L(v_r) then
                add v_m to matchList
            end
        end
        if matchList is empty then
            add new v_r in V_r with L(v_r) = L(v_m)
        else
            mc_max = 0;
            foreach v_r in matchList do
                mc = 0;
                foreach e_r involving v_r do
                    foreach e_m involving v_m do
                        if L(e_r) = L(e_m) then
                            mc = mc + 1;
                        end
                    end
                end
                if mc ≥ mc_max then
                    mc_max = mc;
                    v_best = v_r;
                end
            end
            map v_m in v_best;
        end
    end
end
```

ALGORITHM 4: Merging of the connections step (*mergeConnections* in the main algorithm) of the MDC empiric merging algorithm.

```
put BSF ordering of V_m in bsfList;
foreach v_m in bsfList do
    put all v_m predecessors in predList;
    foreach v_mp in predList do
        /* candidate connections                                   */
        foreach e_m in E_m do
            if e_m involves both v_m and v_mp then
                create a mapping e_mr of e_m in G_r;
                add e_mr to candList;
            end
        end
        /* share candidates                                        */
        foreach e_mr in candList do
            foreach e_r in E_r do
                if e_mr = e_r then
                    remove e_mr from candList
                end
            end
        end
        /* place candidates                                        */
        foreach e_mr in candList do
            foreach e_r in E_r do
                if source of e_mr = source of e_r then
                    /* source collision                            */
                    add Sbox 1x2 v_sb to V_r;
                    add e_r from source of e_r to v_sb;
                    change source of e_r to v_sb;
                    change source of e_mr to v_sb;
                end
                if target of e_mr = target of e_r then
                    /* target collision                            */
                    add Sbox 2x1 v_sb to V_r;
                    add e_r from v_sb to target of e_r;
                    change target of e_r to v_sb;
                    change target of e_mr to v_sb;
                end
                add e_mr to E_r;
            end
        end
    end
end
```

Appendix B: Dynamic Power Manager Logic Regions Algorithms

ALGORITHM 5: Description of the logic regions identification algorithm adopted by the MDC dynamic power manager (*MAP* is the *Association map*).

```
/* Logic Regions identification                                    */
foreach DPN_i in input DPNs do
    V'_i =mapping of V_i in V;
    if isEmpty(MAP) then
        S_0 = V'_i;
        put key S_0 with value DPN_i in MAP;
    else
        foreach S_j in MAP keys do
            if V'_i = S_j then
                add DPN_i to value of key S_j in MAP;
                break;
            end
            if V'_i ∩ S_j ≠ ∅ then
                S_N = V'_i ∩ S_j to MAP;
                put key S_N with value DPN_i in MAP;
                S_j = S_j − S_N;
                V'_i = V'_i − S_N;
            end
        end
        if !isEmpty(V'_i) then
            S_N = V'_i;
            put key S_N with value DPN_i in MAP
        end
    end
end
```

ALGORITHM 6: Description of the and of the logic regions merging adopted by the MDC dynamic power manager (MAP is the *Association map* and $N = |MAP|$).

```
/* Logic Regions merging                                          */
while N >= TH do
    sort MAP basing on CF;
    S_lc =lowest cost S_j in MAP;
    S_slc =second lowest cost S_j in MAP;
    put key S_mrg = S_lc ∪ S_slc in MAP;
    add value of key S_lc to value of key S_mrg in MAP;
    add value of key S_slc to value of key S_mrg in MAP;
    remove keys S_lc and S_slc from MAP;
end
```