IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 15, 2020, accepted November 3, 2020, date of publication November 6, 2020, date of current version November 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3036541

An Integrated Approach and Tool Support for the Design of FPGA-Based Multi-Grain Reconfigurable Systems

RAFAEL ZAMACOLA, ANDRÉS OTERO, (Member, IEEE), ALBERTO GARCÍA, AND EDUARDO DE LA TORRE
Centro de Electrónica Industrial, Universidad Politécnica de Madrid, 28006 Madrid, Spain

Corresponding author: Rafael Zamacola (rafael.zamacola@upm.es)

This work was supported in part by the EU Horizon 2020 Research and Innovation Programme under Grant 732105(CERBERO Project), and in part by the Spanish Ministry of Economy and Competitiveness under Project PLATINO under Grant TEC2017-86722-C4-2-R.

ABSTRACT Dynamic partial reconfiguration technique can be used to modify regions of an FPGA as large as the whole reconfigurable fabric or as small as individual logic elements. However, FPGA manufacturers have focused their efforts on designing tools that support the design of monolithic reconfigurable accelerators spanning large regions of the device. Nevertheless, in some applications, it is enough to fine-tune the accelerators' behavior instead of changing them entirely. In these cases, rather than allocating new accelerators, it is possible to reconfigure individual logic elements of the circuit, such as look-up tables or flip-flops. There is also an intermediate approach that targets the reconfigurability of accelerators composed of several tightly interconnected modules, such as overlays. In those architectures, it is possible to reconfigure only the modules that differ between the existing accelerator versions, thus reducing the reconfigurable footprint granularity. This article proposes a classification of the approaches above, categorizing them as coarse, fine, and medium grain, respectively. There are neither commercial nor academic tools supporting multi-grain reconfiguration to take advantage of each granularity strength on commercial FPGAs. Differently, this article proposes a tool called IMPRESS, that provides design-time and run-time support for multi-grain reconfiguration in Xilinx 7 Series FPGAs. Specific criteria are provided to combine the different granularity levels, trading off the benefits in terms of flexibility and performance, with different design and run-time costs. Two use cases in the image processing and neural network domains have been implemented to show how IMPRESS can build multi-grain reconfigurable systems.

INDEX TERMS FPGA, dynamic partial reconfiguration, multi-grain reconfiguration, IMPRESS.

I. INTRODUCTION

Reconfigurable computing has emerged as a paradigm standing between Application-Specific Integrated Circuits (ASICs) and software programmable architectures, such as Central Processing Units (CPUs) and Graphic Processing Units (GPUs) [1], [2]. ASICs implement a custom circuit tailored to solve a specific algorithm, while CPUs and GPUs have a fixed data-path programmable via an instruction set architecture. Differently, the underlying architecture in reconfigurable computing consists of a reconfigurable fabric, typically a Field-Programmable Gate Array (FPGA), where the parallel sections of the applications can be implemented as spatially distributed custom data-paths. FPGAs can be coupled with CPUs where sequential, and control-intensive application partitions can run more efficiently.

FPGAs may be conceived as dual-layered devices, with a pool of reconfigurable logic resources distributed as a 2D array, and a memory that controls the configuration of the resources in the first layer. This memory holds the configuration bitstream that describes every particular circuit implementation. For those cases when it is not possible to allocate all the logic simultaneously in a single FPGA, a cost-effective alternative to multi-FPGA systems is to swap different

The associate editor coordinating the review of this manuscript and approving it for publication was Li Minn Ang.

configurations at run-time into the same FPGA. This process is called dynamic reconfiguration and requires downloading a new bitstream each time a new configuration is demanded. Besides enabling logic reuse across applications, dynamic reconfiguration constitutes a key technology for autonomous self-adaptive hardware systems [3].

The main obstacle for adopting dynamic reconfiguration of the entire FPGA is the latency overhead when switching contexts. Latency can be reduced using Dynamic Partial Reconfiguration (DPR), which involves reconfiguring a subset of the design instead of the whole FPGA [4]. Compared to full device reconfiguration, DPR reduces both the reconfiguration time and the memory footprint used to store partial configuration files. The reconfigurable footprint granularity can vary from large regions to individual logic elements of the FPGA. Footprint granularity is an important property of a reconfigurable system as it affects overhead, flexibility, and architectural cost [5]. Given its importance, in this article, we propose a classification for DPR systems based on the reconfigurable footprint granularity with three different categories: *coarse, medium, and fine granularities*. It must be highlighted that our proposal differs from the traditional classification of reconfiguration based on the underlying reconfigurable architecture's granularity. Reconfigurable architectures are classified as coarse grain when they have interconnections and reconfigurable processing elements (PEs) that implement word-level operations. In contrast, fine grain architectures consist of PEs and interconnections configured at bit level [6], typically using commercial FPGAs. The work presented in this article addresses FPGA-based reconfigurable systems.

DPR is mostly used to swap monolithic accelerators in and out of the FPGA. We propose to classify this as coarse grain reconfiguration as the accelerators usually span large reconfigurable fabric regions. However, reconfiguration time and memory footprint overhead can be improved by reducing the elements' granularity under reconfiguration. We classify as fine grain reconfiguration to only changing the configuration of specific logic elements of the FPGA fabric or routing resources. There is another approach that uses DPR with an intermediate reconfiguration granularity aiming at reconfiguring highly modular and regular hardware accelerators. This happens in many architectures that can be overlaid on top of the FPGA fabric, such as systolic arrays [7], wavefront arrays [8], general-purpose overlays [9], or block-based neural networks [10], among others. All these architectures are generated by replicating building modules from a preexisting library. Therefore, it is possible to modify their functionality or performance by replacing just a subset of these modules. Moreover, some of these architectures can benefit from scaling its size dynamically, adding or removing some of the building blocks depending on the changing application requirements. We propose to classify this technique as medium grain reconfiguration.

There are different works in the literature targeting the design of dynamically reconfigurable systems. However, to the best of authors' knowledge, none of them addresses multiple reconfiguration granularities in a unified manner. This article provides an integrated vision of multi-grain reconfiguration, showing how coarse, medium, and fine grain reconfiguration can be combined in a given design to leverage each granularity strength. Moreover, we have extended the IMPRESS design automation tool, initially proposed in [11] and [12], with support for design and run-time management of medium grain and fine grain reconfiguration. With IMPRESS, it is now possible to combine all the three granularities in the same system.

The main contributions of this article are:

- A classification of DPR based on their reconfiguration granularity, including a discussion of each category's requirements.
- An integrated approach of multi-grain reconfiguration that provides criteria for combining coarse, medium, and fine granularity levels.
- An integrated design tool that offers design and run-time support for multi-grain reconfiguration in Xilinx 7 Series FPGAs. Among its capabilities, it stands out the support to build run-time scalable 2D overlays, and to manage them using specialized Reconfiguration Engines (REs).

The rest of the paper is organized as follows. Background concepts on reconfiguration are explained in section II. Section III details a classification of DPR based on the granularity level. Related work found in the literature is described in Section IV. Section V presents IMPRESS, our proposal for using multi-grain reconfiguration. Besides including design-time support, IMPRESS also offers run-time capabilities, described in Section VI, for working with multiple granularities. A theoretical discussion on integrating multiple granularities in a system is presented in Section VII. Section VIII presents experimental results to characterize the performance of IMPRESS for each granularity level. Section IX shows two different use cases, a streaming filter pipeline and a scalable BbNN, that show how different granularities can be effectively combined in a design. Finally, section X presents the conclusions and future work.

II. BACKGROUND CONCEPTS

This section introduces important concepts related to the design and implementation of dynamically reconfigurable systems, particularized to the Xilinx 7 Series FPGAs, which are the target of the IMPRESS tool proposed in this work. In order to improve the readability of the paper, we have grouped some of the abbreviations used in this article in Table 1.

A. INTERNAL STRUCTURE OF XILINX FPGAs

FPGAs are composed of multiple reconfigurable resources such as Configurable Logic Blocks (CLBs), BRAMs, and DSPs. Other specialized resources, such as PLLs and GTX/GTH receivers, are also available on these devices. Resources of the same type are grouped in columns, which are

TABLE 1. Main background reconfiguration concepts.

Concept	Definition
RM (Reconfigurable Module)	Accelerator that can be reconfigured into the FPGA.
RR (Reconfigurable Region)	Region of the FPGA that can allocate one or several RMs.
VA (Virtual Architecture)	Set of all the RRs in the FPGA and their interconnections.
PBS (Partial BitStream)	File containing a partial FPGA configuration.
RE (Reconfiguration Engine)	System component that downloads a PBS into the FPGA memory.

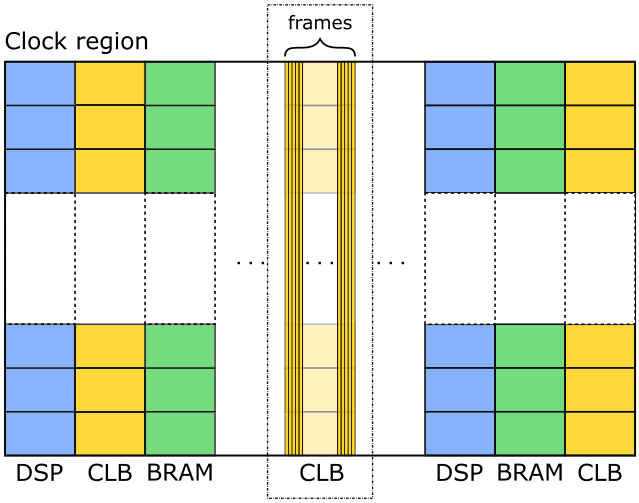


FIGURE 1. A clock region is divided into columns of logic elements. In turn, the configuration memory divides each column into several frames, as shown for the middle CLB column.

interleaved to compose clock regions. Clock region rows are almost identical to each other, making the FPGA resources highly regular along its vertical axis. The structure of the device configuration memory is similar to the disposition of the configurable elements in the logic layer. The minimum addressable unit in the configuration memory is called a frame [13]. Each frame contains part of the configuration of the logic resources placed in a clock region column. Depending on the resource type of the column, a variable number of frames is needed. Figure 1 depicts a clock region in the FPGA. The clock region is divided into several columns, which in turn are divided into several frames in the configuration memory.

Since reconfiguring the device ultimately involves writing to this configuration memory, it is necessary to know its addressing scheme, that is, the relationship between each frame and the column which configuration stores. Moreover, for the specific case of fine grain reconfiguration, it is also necessary to understand how the configuration memory content can be modified to alter particular logic primitives. This information is partially included in the device's configuration guides, while further details have been released by academic works, such as the SymbiFlow [14] open source tool.

Dynamic reconfiguration requires using device ports that provide access to the configuration memory. Apart from the external ports used for the device configuration during system booting, internal ports grant access from the device's logic layer, so enabling self-adaptive systems. In Xilinx 7 Series devices, the ICAP is the configuration port available from

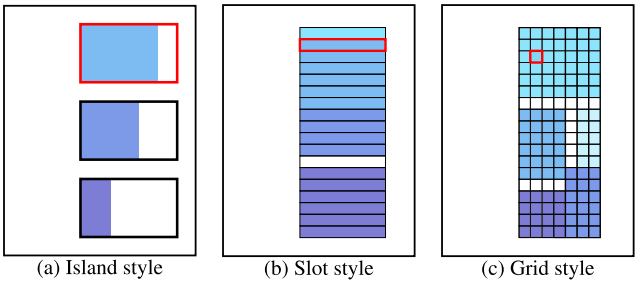


FIGURE 2. Different VA floorplanning styles. A RR is marked in red in each VA style. Blue rectangles represent RMs allocated inside one or multiple RRs. Interfaces with the static system are not shown for clarity.

the programmable logic. In the particular case of the Xilinx Zynq-7000 SoC devices, which includes a hard-core ARM processor integrated with the 7 Series FPGA in the same chip, another configuration port called PCAP is incorporated. Both configuration ports are multiplexed and, therefore, they cannot be used simultaneously.

B. STATIC SYSTEM AND RECONFIGURABLE MODULES

Reconfigurable systems are divided into two parts: the static and the reconfigurable subsystems. The former is composed of all the circuits essential for the system's correct behavior and therefore remains unaltered during its whole lifetime. In turn, the reconfigurable subsystems contain the logic elements that can be modified at run-time. In coarse and medium grain reconfiguration, reconfigurable logic resources are grouped in predefined areas of the device, called Reconfigurable Regions (RRs). The modules that can be configured in each RR are called Reconfigurable Modules (RMs). The RM and RR concepts are only applicable in coarse and medium granularities where modules are exchanged in the device configuration memory. In fine grain reconfiguration, the reconfigurable subsystem consists of individual device primitives spread throughout the FPGA and interleaved with the static logic. However, to speed-up the reconfiguration process, fine grain components are usually stacked in clock-region columns.

C. FLOORPLANNING

Efficient resource management in reconfigurable systems is not possible without the virtualization of the reconfigurable resources of the device. In this article, we will use the term Virtual Architecture (VA) [15] to refer to the set of RRs and their interconnections. The VA floorplanning can follow different configuration styles, as shown in Figure 2: island, slot, and grid [16].

The island-based style consists of several RRs isolated from each other. Each RR can allocate only one RM at a time. One of the main problems of this VA style is that it suffers internal fragmentation (i.e., waste of resources) when RMs of diverse sizes are implemented in the same RR. Reconfigurable resources can be used more efficiently by defining flexible VAs. Reconfigurable systems with flexible VAs formally have several contiguous RRs that can be

grouped to allocate one or more RMs. Depending on how the RRs are defined, flexible VAs can be classified as slot- and grid-based.

The slot-based VA, shown in Figure 2b, entails a region of the FPGA divided into multiple contiguous RRs, called slots. If the defined slots are small enough, internal fragmentation is avoided. However, this VA style can still lead to unused resources due to external fragmentation, which appears when there are enough free reconfigurable resources to allocate a given RM, but these free resources are not contiguous, and so the module can not be configured. External fragmentation can be minimized with efficient run-time allocation algorithms. The grid-based VA can be considered a 2D generalization of the slot-based style. In this case, the RR is divided into smaller RRs arranged in a grid pattern. Each RM can span several adjacent RRs of the grid in both dimensions. 2D grids further minimize internal fragmentation.

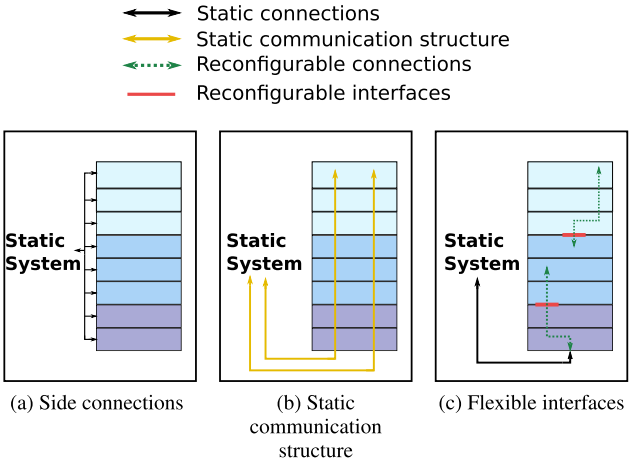


FIGURE 3. Different methods to communicate slot-based virtual architectures.

D. COMMUNICATION INTERFACES

The communication interfaces define how the RMs are connected to the static system and neighboring RMs. The most efficient way to implement these interfaces is by using one routing node with one endpoint in the static region and the opposite endpoint in the reconfigurable region. This type of interfaces has been designated as partition pins [17], virtual interfaces [15] or direct wire binding [18] in the literature. Other interface styles use logic resources (i.e., Lookup tables (LUTs)) at one (e.g., proxy logic) or both ends (e.g., bus macros) [16], introducing overhead both in terms of resources and delay.

In the island-based VA, each region has its interface with the static system. In the slot- and grid-based VAs, it is necessary to define the connection with the static system along with the interfaces between RMs. Three different approaches can be used to interface these VAs. These approaches are explained below for slot-based VAs, but the same concepts apply to grid-based VAs:

- The simplest solution is to directly connect each slot to the static system through one of the shared borders, as shown in Figure 3a. This scheme does not allow RM to RM direct communication.
- The second solution, shown in Figure 3b, relies on placing a communication infrastructure inside the RRs [19]. Every RM contains the communication infrastructure in the same location, and thus, despite being reconfigured each time, the communication infrastructure can be considered part of the static system. RMs can use this infrastructure to communicate with the static system or with other RMs.
- In the last option, shown in Figure 3c, internal modules use nets that belong to the RMs to connect to other adjacent RMs, and from there to the static system. We will refer to these as flexible interfaces, since they are not defined in the static system, and so they can be changed at run-time.

E. RELOCATION OF RECONFIGURABLE MODULES

The configuration of each RM is defined in a partial configuration file, known as Partial Bitstream (PBS). When using vendor tools for implementing reconfigurable systems, the generated PBS includes relevant commands to allocate the RM in the RR where it was originally implemented. This approach prevents using a single PBS to relocate an RM in an RR different from where it was generated. Therefore, if multiple copies of the same RM are to be configured in different RRs to increase flexibility at run-time, it would be necessary to generate multiple copies of the same PBS. Thus, if there were m RMs that can be allocated in n RRs, it would be necessary to generate $m*n$ partial bitstreams. This approach entails a big memory footprint when stored in the final system. However, it is possible to benefit from the FPGA fabric regularity using only one PBS for all the FPGA regions, whenever they have the same distribution of logic resources inside. This feature, called relocation, is of particular importance for slot- and grid-based VAs, where RMs can be allocated in compositions of multiple RRs. To enable relocation, VAs are usually floorplanned over the device layout with slot and grid cells as homogeneous as possible.

The requirements to implement relocatable RMs are:

- Logic primitives (CLBs, BRAMs, and DSPs) within a reconfigurable region must be distributed in the same manner, making the floorplanning compatible.
- Logic and routing resources of the RM need to be completely isolated within the area of a reconfigurable region. Therefore it is necessary to prevent the static system's routes entering the reconfigurable region.
- Physical interfaces in every relocatable region must be equal, making module interconnections compatible.

As detailed in subsection II-A, the distribution of the logical resources of the same type in columns makes Xilinx FPGAs more homogeneous in the vertical axis, which has to be taken into account when floorplanning the VA.

F. SUB-CLOCK REGION RECONFIGURATION

Sub-clock region reconfiguration involves reconfiguring regions that do not span complete clock region columns while leaving the rest of the logic in the column unaltered. This reconfiguration requires special treatment as the organization of the configuration memory in frames makes it mandatory to reconfigure entire clock region columns. The sub-clock region reconfiguration concept is relevant to all the reconfiguration granularities. In coarse and medium granularities, sub-clock reconfiguration must be taken into account when there are multiple RMs stacked in the same clock region. When using fine grain reconfiguration, it is necessary to reconfigure the whole column where the fine grain reconfigurable elements are placed, so it is especially important when there are fine grain components and other static components located in the same column.

Sub-clock region reconfiguration requires specialized run-time support, offered by a Reconfiguration Engine (RE). The RE is a logic module integrated as part of the static design devoted to downloading partial bitstreams in the FPGA configuration memory. A RE supporting sub-clock region reconfiguration has to perform a readback step to read all the affected frames from the configuration memory. Then, the RE has to combine the previous configuration with the new RM bitstream. It is possible to avoid the readback step by maintaining a description of the modules configured in each position of the FPGA. Then, during reconfiguration, the rest of the frame which must remain unmodified is gathered from external memory, according to this system description.

III. PROPOSED CLASSIFICATION FOR RECONFIGURABLE SYSTEMS

This section provides a discussion on how we propose to classify reconfigurable systems depending on the granularity of their reconfigurable elements. The analysis is focused on the most common design requirements in each case. Table 2 summarizes the distinguishing feature for each category, together with the typical features associated with each group.

A. COARSE GRANULARITY

Coarse grain reconfigurable systems include a monolithic accelerator as the reconfigurable unit. This granularity is usually applied with an island-based VA without relocation capabilities. RMs are generally implemented in large reconfigurable regions that can span whole clock regions. RMs are often isolated from other RMs so direct interconnections between RMs, without involving the static system, are typically not considered. This is the most common granularity used in reconfigurable designs.

B. MEDIUM GRANULARITY

This granularity level is introduced in the proposed classification as a technique specifically tailored to build highly regular and modular processing architectures, such as systolic arrays [7], wavefront arrays [8], general-purpose overlays [9] or block-based neural networks (BbNNs) [10]. All these architectural templates are composed of a set of functional units distributed as 1D arrays or 2D meshes. Swapping between variations of these architectures can be carried out by only reconfiguring the modules that differ, leaving the other common submodules unaltered. Variations may involve changing the size of the processing architecture at run-time, thus providing dynamic scalability [20]. It is essential to notice that the main difference between coarse and medium granularities is how the accelerator is composed, rather than the size of the RM. While coarse-grain is used to reconfigure monolithic accelerators, medium-grain reconfiguration is used to reconfigure only specific accelerator sub-modules.

The most effective VA configuration styles are slot- or grid-based VAs, depending on the reconfigurable architecture pattern (i.e., 1D or 2D). In these VA RMs can be connected to each other using direct RM to RM interfaces. Relocation and sub-clock reconfiguration are also highly desirable features to use with medium grain reconfiguration. Sub-clock reconfiguration allows the user to reconfigure multiple relatively small PEs in one clock region. In turn, relocation helps to allocate one PE in multiple different locations of the VA using only one PBS. While it is possible to use medium grain reconfiguration with island-based VAs when the reconfigurable architecture to implement is composed of a few modules, it is not recommended. The reason is that, for a given application, the RRs need a fixed interconnection between them, which makes it difficult to reuse the VA to implement other accelerators that may require different interconnections.

C. FINE GRANULARITY

We refer to fine grain reconfiguration to change specific elements of an FPGA fabric (LUTs, FFs, or routing paths) as an alternative to swapping entire RMs. Therefore, fine grain reconfiguration, as opposed to the other two granularities, cannot be implemented with reconfiguration tools based on RMs. Fine grain reconfiguration allows tuning part of an accelerator logic by modifying a reduced amount of frames. Fine grain reconfiguration offers advantages in three different levels:

- *Reduce resource utilization*. Circuit specialization by reconfiguring particular LUTs is an alternative to circuits in which all possible functionalities are implemented together from the beginning.
- *Reduce reconfiguration time*. The time needed for circuit adaptation is reduced compared to coarse grain reconfiguration since fewer frames need to be reconfigured. To give some numbers, in Xilinx 7 series FPGAs there are 400 LUTs in a single column, which can be reconfigured individually. Each column can be reconfigured with only eight frames, a minimal number compared to the number of frames needed to configure an entire clock region (e.g., 2536 frames per clock region in a 7z020 FPGA).
- *Increase placement flexibility*. One of the main challenges when building grid-based reconfigurable systems

TABLE 2. Typical granularity configurations.

	Coarse Grain	Medium Grain	Fine Grain
Distinguishing feature	Monolithic accelerators	Submodules (usually in a regular architecture)	Individual components (e.g., LUTs)
RR floorplanning	Island	Grid or Slot	Components stacked in columns
Relocation	Not very important	Very important	Does not apply
RM size	Large (usually whole clock regions)	Sub-clock size	Few frames
RM communication	RM - static	RM - RM & RM-static	Does not apply

is how to connect RMs with the static system when they are not placed in the boundaries of the grid. Fine grain reconfiguration can be used to write and read the contents of internal flip-flops, memories or LUTs of the module from the static system, without using a direct physical link.

IV. STATE OF THE ART

This section describes the main contributions existing in the state-of-the-art related to the implementation of dynamically reconfigurable systems. Since there are not integrated support tools for multi-grain reconfiguration (i.e., tools that allow combining multiple granularities in a design), each granularity, together with relevant use cases, is tackled separately.

A. COARSE GRANULARITY

Most reconfigurable systems use coarse grain reconfiguration. Therefore, reconfiguration design flows provided by major FPGA vendors have focused their efforts on supporting this type of reconfiguration. However, Xilinx [17] and Intel [21] reconfiguration flows, only allow building reconfigurable systems following an island-based VA without relocation. This lack of flexibility has induced the development of several academic tools that tried to provide additional reconfiguration capabilities. In the case of Xilinx devices, the first tools appeared for the outdated ISE design environment, and they focused mainly on relocation, such as the works in [22] and [23]. Later on, more comprehensive tools appeared targeting flexible VAs, such as Recobus [19], and more efficient reconfigurable interfaces, such as Go Ahead [18] and Dreams [15].

The launch of Vivado, the newer Xilinx design suite, made the previous tools almost useless for new FPGA families. The reason is that all of them were based on the proprietary Xilinx Design Language (XDL) [24], which is no longer supported in Vivado, to modify the design at the physical level. Gradually, new tools have appeared for Vivado based on its TCL command interface, instead of XDL. Table 3 presents a comparison between Xilinx and Intel commercial reconfiguration flows and the main academic tools that can be used in Vivado to enhance Xilinx reconfiguration flow. Most academic tools enhance Vivado reconfiguration flow in only one facet such as relocation (e.g., RePaBit [25], Reloc [26] and the work of Oohmen *et al.* [27]), or implementing flexible VA configuration styles (e.g., Amorphous DPR [28]). However, some of the aforementioned tools use bus macros or proxy logic to implement the interfaces; Thus, adding resource and timing overhead to the interfaces. An updated version of Go Ahead [18], combined with the BITMAN tool [29] can implement overhead-less reconfigurable interfaces using flexible virtual architectures. An example of a slot-based virtual architecture with coarse grain reconfiguration using Go Ahead is shown in [30].

B. MEDIUM GRANULARITY

There are some reconfigurable designs in the literature that use DPR to implement modular processing architectures. A first example is a general-purpose overlay proposed in [31], which aims at implementing code written in a domain-specific language. Each module features a functional unit that can be reconfigured so that the overlay can map the functionality desired at any given time. A variation of this work was proposed in [32] for overlays running in cloud accelerators. Real-time video pipelines can also benefit from this type of reconfiguration, as shown in [33]. The aforementioned applications used a reconfiguration style that can be considered as medium grain, with only a few RRs connected to each other through the static system. Therefore, they could be implemented using commercial reconfiguration tool flows using an island-based VAs. However, as explained in subsection III-B, there are several features that are highly desirable when using medium grain reconfiguration with architectures with several processing elements, such as RM relocation, sub-clock reconfiguration, flexible VA configuration styles, and RM-to-RM direct communication. An example of a reconfigurable system using medium grain reconfiguration with relocatable RMs is [34], which implements a systolic array to accelerate multiple algorithms, such as the extended Kalman filter and the discrete wavelet transform. Moreover, there are also examples in the state of the art of 2D architectures that leverage grid-based VAs and flexible RM-to-RM communications to build a run-time scalable 2D systolic array [35] and an evolvable hardware architecture for video filtering [36]. Both applications were implemented with the reconfiguration tool Dreams [15].

As shown in Table 3, most commercial and current academic tools lack the features necessary to use medium grain reconfiguration effectively. The only tools that include all the features to implement medium grain reconfiguration are IMPRESS and the combination of Go Ahead [18] and BITMAN [29]. The main difference between IMPRESS and Go Ahead is the method used to connect RMs to each other and to the static system. While IMPRESS uses flexible interfaces (see Figure 3c), Go Ahead uses a static communication

TABLE 3. Reconfiguration tools available for Xilinx Vivado and Intel Quartus Prime for coarse & medium reconfiguration granularities.

Tool	Coarse & Medium grain reconfiguration features				
	Relocation	Interface Type (LUTs per interface net)	VA conf. style	RM-to-RM communications	sub-clock reconfiguration
IMPRESS	Yes	Partition Pin (0)	All	Yes (Flexible)	Yes
Xinlinx reconf. flow [17]	No	Partition Pin (0)	Island	No	No
Intel reconf. flow [21]	No	Wire LUT (1)	Island	No	No
Oohmen et al. [27]	Yes	Bus Macro (2)	Island	No	No
RePaBit [25]	Yes	Bus Macro (2)	Island	No	No
Reloc [26]	Yes	Partition Pin (0)	Island	No	No
Go Ahead [18] + BITMAN [29]	Yes	Direct Wire (0)	All	Yes (static infrastructure)	Yes
Amorphous DPR [28]	No	Proxy logic (1)	All	No	No

infrastructure (see Figure 3b). Placing a communication infrastructure inside the reconfigurable regions forces to decide the size of the grid at design-time. The designer has to reach a trade-off between the grid granularity and the number of resources utilized by the communication infrastructure (i.e., the finer the granularity, the more resources are needed). Moreover, the communication links among RMs offered by this type of infrastructure is limited, which hinders the implementation of highly interconnected arrays which are the most common applications that can be build using medium grain reconfiguration. For these reasons, using flexible interfaces is more efficient to build 2D architectures.

C. FINE GRANULARITY

Fine grain reconfiguration was partially supported in the past by Xilinx with difference-based partial reconfiguration [37]. Difference-based partial reconfiguration was the first approach that Xilinx introduced to use partial reconfiguration in their devices. This approach computed the difference between two designs and generated a partial bitstream containing the changing configuration bits. The main problem is that this approach is not scalable to systems that need multiple different configurations as it is not practical to compute the difference between all the possible combinations of different configurations. Thus, Xilinx discontinued difference-based partial reconfiguration in favor of module-based reconfiguration flows that contained a static system with one or more RRs [5].

Different application-specific fine grain reconfiguration schemes have been proposed by the academic community, to adapt both routing and functional units. In [38], a reconfigurable crossbar switch made up of reconfigurable LUTs is proposed, in which by reconfiguring the truth table of the LUTs, the crossbar configuration is changed. Authors claim an area-saving of up to 84% compared to other solutions since the crossbar does not require control logic. In [39], a crossbar using a specific reconfigurable architecture made up of reconfigurable LUTs is implemented.

Going further, some authors have proposed the implementation of crossbars without using any logic element but leveraging the low-level routing resources of the FPGA. This is the case of [40], which introduces the concept of tunable connections (TCONs), implemented in the routing resources of a theoretical device architecture. A different approach is the reconfigurable crossbar called ρ-P2P proposed in [41]. In this work, the authors proposed saving all the possible routing configurations of the crossbar as PBS, in such a way that routing can be modified by reconfiguring only the frames containing routing resources, reducing the configuration time. Authors in [42] propose a different approach, consisting of changing the routing of arbitrarily designed netlists by rerouting a given path at run-time without using logic resources.

Fine grain reconfiguration can also be used to implement LUT-based functional units. The functional unit's behavior depends on the LUTs' configuration, which can be modified using fine grain reconfiguration. In this way, the same functional unit can implement multiple functionalities. This approach has been used in [43] to implement an evolvable hardware system for video filtering, which can evaluate up to 139,000 candidate circuits per second. Fine grain reconfigurability can also be used to reduce resource usage as done in [44], [45]. In these works, authors reduce the circuit size by embedding some of the input values that do not often change, as parameters inside special reconfigurable LUTs, called TLUTs. In this way, when one of those input changes, TLUTs are reconfigured to adapt the circuit to the new functionality. Authors show a great reduction in LUTs up to 39% in adaptive FIR filters and a 66% in ternary content-addressable memories.

V. IMPRESS: A TOOL TO SUPPORT MULTI-GRAIN RECONFIGURABLE SYSTEMS DESIGN

IMPRESS (IMplementation of Partial REconfigurable SystemS) is an open-source tool for implementing multi-grain reconfigurable systems [46]. It was proposed initially in [11] for designing reconfigurable systems using flexible VAs and relocatable reconfigurable modules. In this work, IMPRESS has been extended to support multi-grain reconfiguration. To that end, both the design-time and run-time support of IMPRESS have been updated. The design-time support has been extended to include different fine grain reconfiguration components that can be instantiated in a design. In turn, the run-time support has been extended to (1) reconfigure fine-grain elements effectively using a specialized RE, (2) use slot and grid VAs easily, and (3) to mix different

reconfiguration granularities in one design. In this way, IMPRESS is the first tool (to the best of the authors' knowledge) that unifies the implementation of reconfigurable systems with the three granularity levels. IMPRESS currently works with series 7 FPGAs. Making IMPRESS compatible with other Xilinx devices as the Ultrascale+ FPGAs is planned as future work. Moreover, as IMPRESS relies on Vivado TCL commands and low-level bitstream configuration details, it cannot be compatible with FPGAs of other manufacturers (e.g., Intel).

IMPRESS tries to simplify the design of complex reconfigurable systems, hiding as much as possible the low-level device-dependent details. To accomplish a reconfigurable design with IMPRESS, the user only has to provide three files with the system's specification. With the information in these files, IMPRESS automatically carries out all the subsequent steps to provide the full and partial bitstreams. In the first file, the user must provide the design sources' path and the selected FPGA. The design sources corresponding to the static system and the reconfigurable modules can be provided as HDL files or design checkpoints. In the second file, the user has to define the RRs' coordinates within the FPGA layout. In the last file, the user has to specify the interface of the RRs. This means, to assign which RR border has to be used for each RM port. Further details on the physical interfaces are defined below.

In the case of fine grain reconfiguration, the user only has to instantiate in their HDL designs specific HDL components from a predefined library, as discussed in the corresponding subsection.

A. COARSE GRAIN AND MEDIUM GRAIN RECONFIGURATION WITH IMPRESS

IMPRESS enables the implementation of each RM and the static system in fully decoupled runs. This way, new RMs can be added to the system without generating all the modules from scratch. Since Vivado requires having a static partition and at least one RM in the project, IMPRESS automatically introduces dummy logic modules. Thus, during the implementation of the static part, IMPRESS generates and instantiates one dummy RM for each RR, while in the case of the RM's implementation, it uses a dummy static system.

To ensure compatibility among the static system and the RMs, both must share the same physical interface. IMPRESS leverages the partition pins available in the Xilinx reconfiguration flow to ensure this compliance. Partition pins are nodes that have one endpoint in the static region and the other one inside the reconfigurable part. Differently to Vivado, IMPRESS limits partition pins to one-hop nodes located at bordering interconnection blocks (see Figure 4a). This approach increases compatibility for module relocation. Moreover, the user can specify which borders are used for each of the interface nets. It is also possible to define interfaces that only use some interconnection blocks of the edge (i.e., south_0:3 specifies that a net can only use the first four south interconnection blocks, see Figure 4b). In coarse grain reconfigurable systems, the interface is generated automatically by IMPRESS during the implementation of the static system. A description of the reconfigurable interface, including which partition pins are used for each crossing net, can be saved and applied during the implementation of compatible reconfigurable modules.

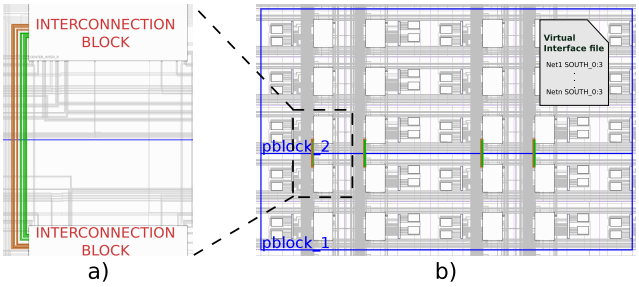


FIGURE 4. (a) One-hop nodes that can be used as valid interfaces (b) example of IMPRESS virtual interface file.

The decoupling of the static system and the RMs, combined with the definition of one-hop interfaces, allows IMPRESS to build efficient VAs. Instead of defining the VA layout at design-time, it is possible to define a single flexible RR that can allocate either one RM using an island-style or multiple RMs following a slot or grid VA style. The RR is connected to the static system using a fixed set of nodes placed at the RR border. In contrast, internal connections between RMs are not fixed and are defined by the interfaces of the RMs. Whenever two RMs share a border and have a compatible interface, they are effectively connected. In this way, the RMs can be connected to each other using any VA style (e.g., slot or grid). Moreover, as the RR does not contain internal static communication resources, the size and number of the grid/slots are not predefined and can be modified at run-time, adapting them to the application requirements.

Figure 5 shows how an empty RR can adopt different VA styles at run-time by allocating different RMs. On the left side of Figure 5, the RR allocates two coarse RMs in a slot-based fashion. In contrast, on the right side the RR allocates six medium grain RMs in a grid-based style using direct RM-to-RM interfaces.

IMPRESS provides support for module relocation. As stated in Section II, relocatable designs must comply with three requirements. The first one is that regions must have the same distribution of logic resources. IMPRESS run-time software library includes a function to check if two regions are compatible. The second requirement is to ensure that the static routing cannot enter the RR. As Vivado does not have any directive to guarantee this, IMPRESS generates a blocker net, such as the one proposed in [18], that is used as a fence during the routing phase to ensure that the static system does not enter inside the RR. The third requirement is to apply the same physical interface to each region. IMPRESS automatically applies these interfaces in island-based VAs. However, in slot/grid VAs, where RMs can

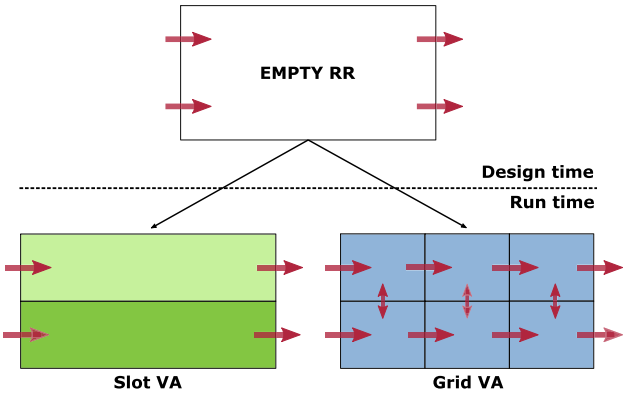


FIGURE 5. Example of how IMPRESS can generate an empty RR that can adopt different VA styles at run-time.

be arranged in different positions and interconnections are highly flexible, it is the user's responsibility to ensure that an RMs is compatible with other RMs and the static system, as it would be computationally expensive to check at run-time that interfaces are compatible. The downside of using relocation is that due to the extra design constraints (e.g., same interfaces and blocker net) the maximum frequency of the system can diminish, as already shown in [11].

B. FINE GRAIN COMPONENTS WITH IMPRESS

The IMPRESS fine grain reconfiguration proposal is based on the definition of a set of parameterized HDL components that users can instantiate either in the static system or within a RM (i.e., mixing fine grain reconfiguration with coarse/medium reconfiguration). These fine grain components have built-in features that make them reconfigurable at run-time.

Three different types of fine grain reconfigurable components are proposed: parameters, multiplexers, and Functional Units (FU). The user can take advantage of reconfigurable parameters to tune hardwired settings of an accelerator without the need to connect them to the static system. Similarly, multiplexers can change the data-path of a module without using selection lines. In turn, functional units can implement different logic and arithmetic operations that can be reconfigured by changing their truth tables through the reconfiguration interface. All the fine grain components that have been proposed are based on the reconfiguration of LUT equations. Fine grain reconfiguration depends heavily on certain device particularities. In Xilinx 7 series devices, a device LUT primitive is comprised of two 5-input LUTs, resulting in a 6-input LUT with two outputs (Figure 6a). Configurable Logic Blocks (CLB) contain two slices with four LUTs each. The truth tables of the four LUTs that compose the slice are distributed over four consecutive frames [29] in such a way that even if a single LUT needs to be reconfigured, all the four frames must be changed. However, if inputs A1 and A4 are not used, it is possible to reconfigure the four LUTs with a single frame, thus achieving a fourfold decrease in reconfiguration time.

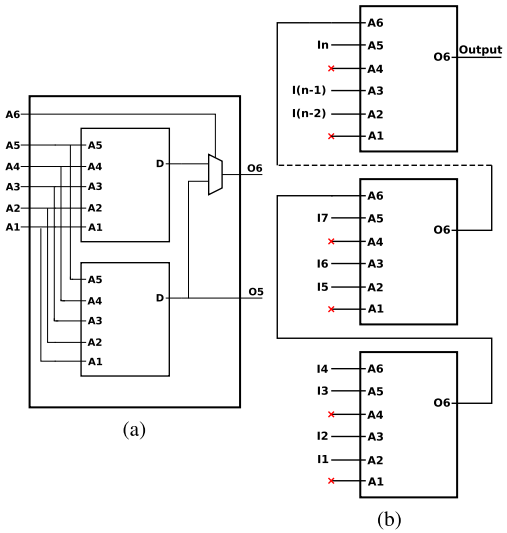


FIGURE 6. a) 6-input look-up table b) n-input 1-bit multiplexer.

The design of optimized fine grain components is as follows. LUT-based parameters leverage the two 5-input LUT configuration of a 6-input LUT (Figure 6a) to use the two available outputs (O5 and O6). This way, for an n-bit parameter, only $\lceil n/2 \rceil$ LUTs are needed. In turn, multiplexers are defined by two parameters: the number of inputs and maximum input width. To allow an arbitrary number of inputs, they are chained as shown in Figure 6b. To allow m-bits width input multiplexers, the chain in Figure 6b is replicated m times. It is worth noting that the multiplexer in Figure 6b does not contain selector lines as they are embedded in the LUT's truth table. The selected input can be changed by modifying the LUT equations using DPR. The design of both elements avoids inputs A1 and A4 so that they can be reconfigured with only one frame, hence speeding up reconfiguration. However, in resource-constrained designs, the multiplexer can be adapted to use their six inputs, but increasing reconfiguration time.

The last component provided is a fine grain reconfigurable functional unit, extended here from the original proposal targeting image processing in [43]. Figure 7 shows the FU structure for one bit. The FU has two stages, the first one is made up of the LUT in the left and the carry operator, while the LUT in the right forms the second stage. The first stage can perform an addition/subtraction operation or compare the two inputs or one of the inputs and any arbitrary constant. The second stage is used as a multiplexer to select any of its inputs using the carry out from the first stage as the input selector. It is also possible to skip step one and perform a logical operation involving the two inputs and an arbitrary constant in the second stage. Table 4 shows some of the arithmetic and logic functions that can be performed by modifying both LUT configuration bitstreams. The FU does not use the A4 input, and therefore it can be reconfigured with only two frames. The presented FU was initially conceived to implement a set of functions used in the image processing

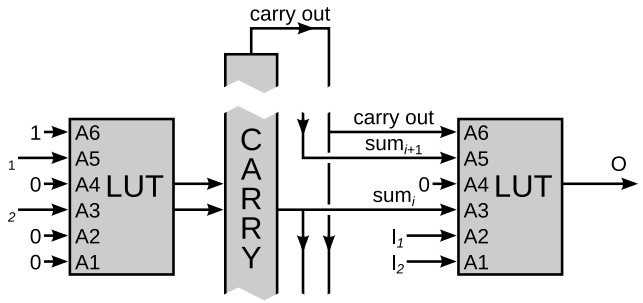


FIGURE 7. Fine grain Reconfigurable Functional Unit structure.

TABLE 4. Functions implemented by the fine grain reconfigurable functional unit.

Functions		
$(I_1 + I_2) \mod 2^n$	$\max(I_1, I_2)$	I_1 nand I_2
$(2 \times I_x) \mod 2^n$	$\max(I_1 - I_2, 0)$	I_1 nor I_2
$\min(I_1 + I_2, 2^n - 1)$	$\max(I_2 - I_1, 0)$	I_1 xnor I_2
$\min(2 \times I_x, 2^n - 1)$	not I_x	$(I_2 > I_1)$? 255 : 0
$\lfloor (I_1 + I_2)/2 \rfloor$	I_1 and I_2	$(I_1 > I_2)$? 255 : 0
$I_x/2$	I_1 or I_2	$(I_1 - I_2) \mod 2^n$
$\min(I_1, I_2)$	I_1 xor I_2	$(I_2 - I_1) \mod 2^n$

domain. However, extending the catalog by adding new FUs featuring a different set of functions is possible.

The main advantage of fine grain parameters and multiplexers is that they can be changed without using specific connections to the static system. The main application of these components is to generate scalable architectures combining medium and fine granularities, as will be explained later. Thus, these components have been designed so that they can be reconfigured by rewriting only one frame, thus reducing the reconfiguration time. Fine grain reconfiguration can also contribute to reducing resource occupancy in two different ways. First, a fine grain reconfigurable FU can substitute multiple non-reconfigurable FUs, followed by a selection multiplexer. Second, fine grain components allow circuit configuration without bus interfaces (e.g., AXI interfaces), reducing logic resources and routing congestion in a design. This approach has bandwidth limitations, which will be evaluated as part of the experimental results.

IMPRESS facilitates the use of fine grain reconfigurable components by automatically applying placement constraints so that all the dynamically reconfigurable LUTs occupy the minimum number of columns without user intervention. If a column is not filled with fine grain LUTs or any other fine grain component type, then IMPRESS automatically instantiates dummy LUTs (i.e., LUTs not used in the design). This way, it is ensured that the clock region column only contains reconfigurable LUTs of a unique component type (i.e., not mixing parameter, multiplexer, and FU LUTs). Packing LUTs in columns reduces the number of frames to be changed, and therefore reconfiguration time when compared to the random placement of the LUTs throughout the FPGA fabric. Fine grain components can be instantiated inside coarse and

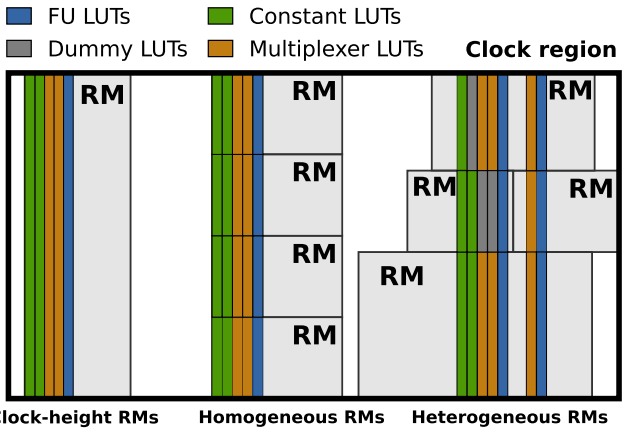


FIGURE 8. This figure shows how different RMs with fine grain components can be stacked in the same clock region. The typical scenarios are to use either clock-height RMs or homogeneous RMs. The most complicated scenario is mixing heterogeneous RMs, making it necessary to configure the fine grain components stacked in the same column.

medium grain reconfigurable modules, so all the reconfiguration granularities can coexist in multi-grain designs. Figure 8 shows different scenarios that can occur when mixing fine grain components inside the RMs. The most complicated scenario occurs when using heterogeneous sub-clock region RMs. To ensure that fine grain components are stacked in the same column, the user can specify the CLB column where to place the fine grain components.

VI. RUN-TIME SUPPORT FOR MULTI-GRAIN RECONFIGURATION

Besides including design-time support to generate reconfigurable systems, IMPRESS also includes run-time support to manage and download coarse, medium, and fine grain reconfigurable components into the FPGA. Two different reconfiguration engines tailored for each type of reconfiguration and a system management library are provided with this aim.

A. COARSE AND MEDIUM GRAIN RECONFIGURATION ENGINE

The coarse and medium grain RE is implemented as a software component that uses the PCAP configuration port available as a peripheral in the ARM hard-core processor of the Xilinx Zynq SoPC [17]. As the RE is implemented in software it does not use any reconfigurable logic resources.

The RE plays an essential role in relocating systems as it has to generate the configuration commands at run-time to indicate to the internal configuration logic of the FPGA the final position where the RM is going to be configured. The RE also has a significant function in sub-clock region reconfiguration as it has to ensure that the reconfigurable regions above and below the area under reconfiguration are left unchanged.

The process of downloading a PBS is as follows: the reconfiguration engine identifies all the clock rows occupied by the reconfigurable region. For each row, the RE reads the

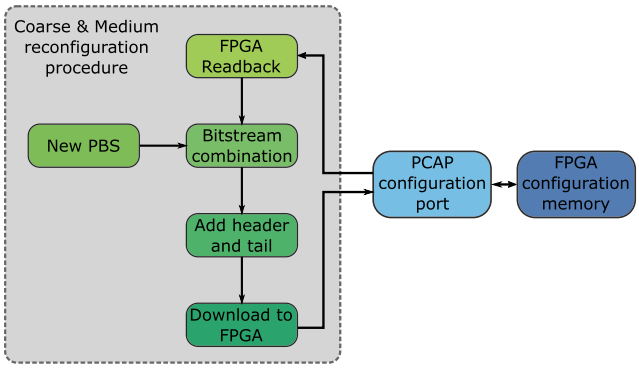


FIGURE 9. Coarse and medium grain RE procedure.

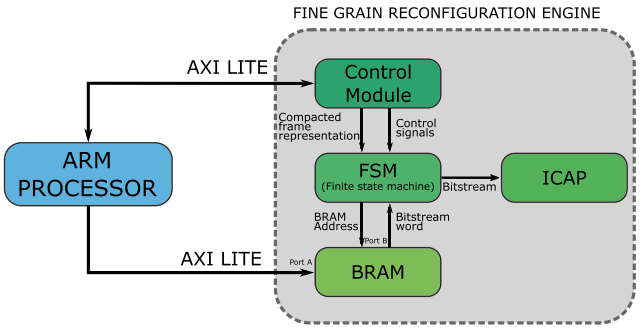


FIGURE 10. Fine grain reconfiguration engine schematic.

initial content of the configuration memory and combines it with the information in the PBS generated with the tool (affecting only the targeted RR), forming the new content of the configuration memory. Once the bitstream composition finishes, the reconfiguration engine adds a header, including the FPGA location, where the PBS will be configured, together with the configuration commands. The composed PBS is finally downloaded into the FPGA. This process is outlined in Figure 9.

When using the medium reconfiguration granularity, it is common to reconfigure several RMs stacked in the same clock region. If all the RMs are reconfigured individually, the RE repeats the same readback and bitstream download operations for each RM. For this reason, the RE includes an option to perform one readback for the first RM and downloads the bitstream only when the last RM has been combined with the previous configuration. Thus, enhancing the performance of the reconfigurable system using the medium reconfiguration granularity.

B. FINE GRAIN RECONFIGURATION ENGINE

Coarse and medium grain module reconfiguration is typically carried out as part of a preparation stage before the accelerator starts computing. Differently, fine grain reconfiguration is often interleaved with computation, and so it needs to be fast to be effective. For that reason, a specialized RE adapted from the one proposed in [43] is provided in IMPRESS. The fine grain RE has been implemented in two different ways. The first implementation is a hardware component inside the FPGA that uses the ICAP configuration port embedded in the programmable logic of the SoPC. In contrast, the second implementation does not use FPGA resources as it is implemented as a software component that uses the PCAP configuration port.

As mentioned in Section V-B, the placement constraints that IMPRESS applies for stacking fine grained components in columns make it possible to skip the readback step in the reconfiguration process. To that end, the configuration of all the columns that contain fine grain components is stored in the DDR memory so that the hardened ARM core can access it without having to read the FPGA configuration memory. In this regard, IMPRESS run-time uses compressed representations of the columns to reduce the required memory. This approach avoids saving the four frames (i.e., 404 words, with 101 words per frame) that comprise a column of LUTs in 7 series devices. For reconfigurable parameters and multiplexers, each LUT is adequately represented with two bits (a LUT parameter has two bits, and a LUT multiplexer uses four inputs which can be selected with two bits). As there are 200 LUTs in a column (50 slice/column * 4 LUTs/slice), it is possible to represent a column filled with parameters/multiplexers with 400 bits grouped in 13 32-bit words. Differently, the minimum size of IMPRESS fine granularity FUs is 4-bit blocks, which contains 8 LUTs. Therefore a column can hold 25 FU blocks. It is possible to represent each FU block with 5 bits (an FU can implement 32 different functions). Therefore, an FU column can be represented with 125 bits (5 bits/FU * 25 FU/column) that can be packaged in 4 words.

In the hardware implementation, the reconfiguration process starts when the processor sends the compact representation of the new configuration of the column to the RE. The RE divides the received information into blocks of bits, each one representing the configuration of a fine grain reconfigurable element (e.g., 4 bits for parameters and multiplexers and 5 bits for FU). This compact representation has to be translated into the corresponding configuration frames before sending it through the ICAP. The translation is carried out by reading from a BRAM memory in the position encoded by the packaged bits, where the expanded frame words (i.e., partial bitstream) are stored. Therefore, the BRAM memory embedded in the RE stores the equivalent full frames for each packaged configuration. The software implementation follows a similar process, but the conversion from column configuration to FPGA configuration is done in the processor, without occupying logic resources in the device. Figure 10 depicts the schematic of the fine grain RE.

C. MULTI-GRAIN RECONFIGURABLE SYSTEM MANAGEMENT LIBRARY

IMPRESS includes a software library that runs on the hardened ARM core to manage multi-grain reconfigurable

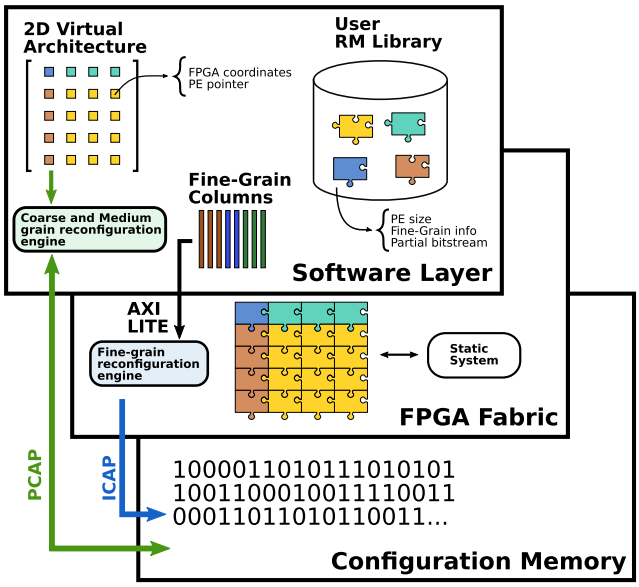


FIGURE 11. This figure shows the main components of the software library that support the run-time management of the multi-grain reconfiguration and how these components are connected to the FPGA fabric and the FPGA configuration memory.

systems at run-time. The library provides a set of primitives to handle any VA with coarse or medium grain RMs containing fine grain elements if required. Figure 11 shows the relation between the software library, the FPGA fabric, and its configuration memory. The library contains the following elements: a user RM catalog, a representation of the VA, and a description of the columns that include fine grain components.

The RM catalog has to be created by the user after implementing the RMs and the static system. It is a file with data structures in C describing each RM available. It contains an array in which each element is a structure that describes one RM, including its size, PBS file name, and a description of its fine grain components.

The VA's internal representation is a two-dimensional array where each element corresponds to a location inside the RR. At run-time, the user can select which RM has to be allocated in each VA element. Then, IMPRESS uses the coarse and medium grain RE to reconfigure the RM in the desired location without user intervention.

Once all the RMs have been placed inside the VA, it is possible to adapt each RM using fine grain reconfiguration. The library includes functions that can be used to change the value of specific fine grain components of a given RM. The user only has to specify the new value for the element and IMPRESS automatically searches which column is affected and changes its configuration automatically. The user can change the value of multiple fine grain components before IMPRESS transfers the new configuration data to the fine grain RE. In this way, it is possible to modify several fine grain components and perform only one reconfiguration process, speeding up the whole process.

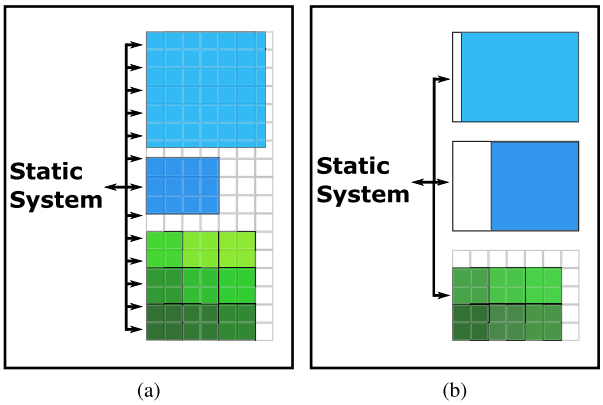


FIGURE 12. Coarse and medium granularities combined. Blue rectangles represent coarse grain RMs while green rectangles represent medium grain RMs.

VII. AN INTEGRATED APPROACH TO MULTI-GRAIN RECONFIGURATION

This section provides a critical analysis of how different granularity levels can be combined and the more convenient setup depending on the system requirements.

A. COARSE AND MEDIUM GRANULARITIES

This scenario is convenient when accelerators using highly regular architectures (e.g., systolic arrays or overlays) must share reconfigurable resources with monolithic accelerators. One option to combine both granularity levels is to use a flexible VA, as shown in Figure 12a. A second option is to define an island-based VA, where each island can be used as a grid-based VA if necessary, as shown in 12b. The latter provides benefits when it is needed to have more than one RR in the FPGA, for example, to provide access from some of these RRs to specific device resources, such as I/O pins.

B. COARSE AND FINE GRANULARITIES

With this combination, coarse grain reconfiguration can be used to swap RMs in and out of the system, while fine grain reconfiguration is useful to adapt the internal behavior of each RM, without reconfiguring it completely. Figure 13 shows an island-based VA with coarse and fine grain reconfiguration. The vertical lines crossing the reconfigurable region in orange represent fine grain components stacked in the same column. As explained in Section III, fine grain reconfiguration reduces the number of resources occupied by the system, so it allows reducing the size of each RM. Another benefit when using fine grain reconfiguration is to exploit its fast reconfiguration time if frequent changes are needed inside a given accelerator. Fine grain reconfiguration also allows reducing the cost of the interface between the static system and the RM. Instead of using a bus to change parameters within an RM, fine grain reconfiguration can be used to reconfigure those elements without a dedicated interface.

C. MEDIUM AND FINE GRANULARITIES

The main problem that restricts the usage of medium grain reconfigurable architectures built on grid-based VAs with

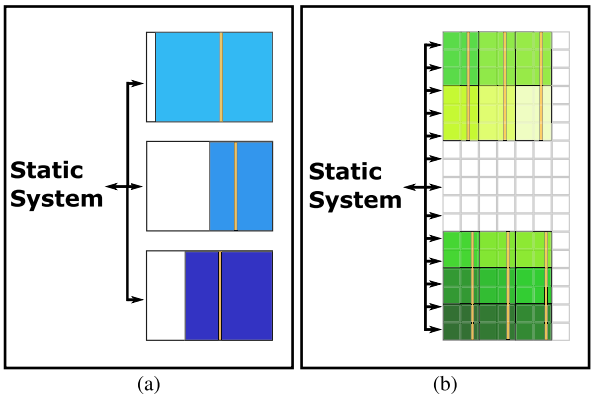


FIGURE 13. (a) Fine and coarse granularities combined (b) fine and medium granularities combined. In both cases the yellow lines represent fine grain components (e.g., LUTs) stacked in one clock-region column.

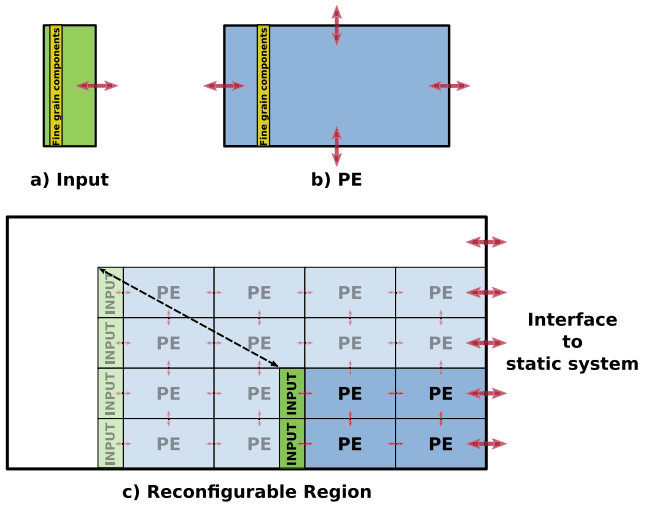


FIGURE 14. Input (a) and PE (b) are reconfigurable modules with fine-grain components that are used to build a run-time scalable 2D architecture (c).

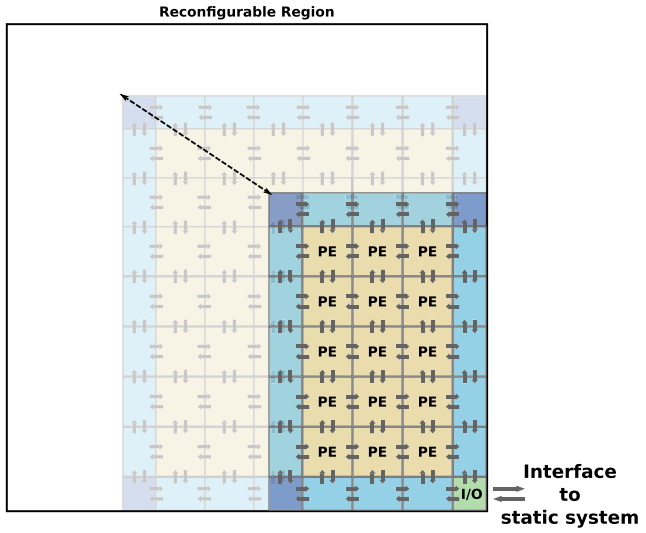


FIGURE 15. Using communication RMs to distribute the input/outputs to all the PEs in a scalable 2D architecture.

flexible, reconfigurable interfaces is how internal submodules in the grid can exchange data with the static system without sharing a common physical border. Fine grain reconfigurable parameters, used as an alternative to memory-mapped buses for data supply to internal modules in medium grain reconfigurable architectures, constitute a convenient solution to this problem. Figure 13b shows a grid-based VA with two medium grain accelerators that contain fine grain elements inside.

Furthermore, combining medium and fine grain reconfiguration enables run-time scalable 2D processing architectures where only the minimum necessary resources of the RR are used at any time, leaving the unused area available for other applications. On the one hand, medium grain reconfiguration is used to change the architecture's submodules or to change its size by adding or removing submodules at run-time. On the other hand, fine grain reconfiguration allows configuring the RM's behavior in a fast and centralized way without the need for a direct connection between the static system and every RM in the architecture. Moreover, fine grain reconfiguration can be used to feed input data to the architecture. This approach solves one of the main problems of scalable architectures, that is the fact that the inputs' position varies with the size of the architecture changes, which hinders a direct connection to the static system. An example of a scalable architecture using fine and medium granularities is shown in Figure 14. There are two types of RMs: an input module and a PE, both of them using fine-grain components. In the case of the PE, the fine-grain components allow to change the configuration of each PE while in the case of the input module, it is possible to feed the inputs using fine-grain reconfiguration. As shown in Figure 14c, the outputs of the architecture are fixed at the right border of the RR. In contrast, the location of the inputs varies according to the size of the architecture; therefore, it is not possible to connect them to the static system using a physical interface. In this scenario, the input modules with fine-grain components can be used to feed the inputs independently of the size of the architecture.

There are applications where the throughput provided by fine-grain reconfiguration might not be enough. In this case, it is possible to use another alternative where all the input/outputs are fixed in one corner of the RR and distributed to the PEs using specialized communication RMs, as shown in Figure 15. The main drawback of this solution is an overhead in reconfigurable resources, especially when the size of the architecture is small. Notice that even in this scenario, fine-grain reconfiguration is still needed to configure the PEs. IMPRESS supports both approaches for building run-time scalable 2D architectures.

D. COARSE, MEDIUM AND FINE GRANULARITY

Finally, it is possible to combine all the reconfiguration granularities and use each one for a specific purpose. Coarse grain reconfiguration is used to swap monolithic accelerators and to replicate accelerators for redundancy and acceleration purposes. Medium grain reconfiguration provides support for

TABLE 5. Coarse grain RE performance.

RR size	Readback (μs)	recombination (μs)	download (μs)	**total** (μs)
one clock region	3351	1542	4128	**9021**

composing modular accelerators. Lastly, fine grain reconfiguration allows saving resources in RMs or to provide a connection of the static part with each RM.

VIII. EXPERIMENTAL RESULTS

This section evaluates the performance of IMPRESS for each granularity level.

A. COARSE GRAIN

The performance of coarse grain reconfiguration depends on the size of the FPGA region to reconfigure. As a reference, Table 5 shows the time needed to reconfigure a whole clock region in a Xilinx *Zynq-7020* SoPC using a PBS of 412.8 kB. As shown in Table 5, the time spent in the reconfiguration process is divided into three different phases. The readback process where the previous configuration is read takes 3351 μs. The previous configuration is combined with the new partial bitstream in 1542 μs to compose the configuration sequence, which is finally sent through the PCAP configuration port in 4128 μs.

Regarding the implementation time to build a reconfigurable system, this metric is highly dependent on the target application. The use case designs included in the next section of this article cannot be implemented with Xilinx reconfiguration flow. For that reason, it is not possible to compare the implementation time of both reconfiguration flows. We refer the reader to the original paper of IMPRESS [11] that compared the implementation time to build an image filter architecture using an island VA with IMPRESS and with Xilinx reconfiguration flow. In the case of Xilinx reconfiguration flow it took 131 seconds to implement the static system and 100 seconds to implement each RM. In contrast, IMPRESS spent 229 seconds to implement the static system and 75 seconds in each RM.

B. MEDIUM GRAIN

This subsection evaluates the performance of medium grain reconfiguration in two different scenarios: reconfiguring a modular accelerator and scaling a 2D architecture.

In the first scenario, we compare the difference between reconfiguring a monolithic accelerator versus dividing it into different modules. To do so, we divide a clock region of the Xilinx *Zynq-7020* SoPC into five slots that can allocate a module of the accelerator. The slots can be arranged in two configurations: one top of the other (i.e., vertical configuration) or side by side (i.e., horizontal configuration). When using coarse grain reconfiguration, it is necessary to use a PBS of 412.8 kB requiring 9021 μs. In contrast, when using medium grain reconfiguration, it is necessary to change only the number of slots that differ from the previous

TABLE 6. Medium grain performance.

	Reconfiguration time (μs)	PBS size (kbytes)
Coarse grain	9021	412.8
1 slot (horizontal configuration)	1853	86.4
1 slot (vertical configuration)	7933	82.6

TABLE 7. Reconfiguration performance in 2D scalable architectures.

	Reconfiguration time (μs)
Coarse grain: all sizes	9021
Medium grain: 1x1 -> 2x2	3267
Medium grain: 2x2 -> 3x3	4885
Medium grain: 3x4 -> 4x4	6348
Medium grain: 3x3 -> 4x4	6456
Medium grain: 4x4 -> 5x5	8184
Medium grain: 1x1 -> 5x5	8727

accelerator. To reconfigure one slot of approximately 84.5 kB takes 1853 μs when the slots follow a horizontal configuration and 7953 μs when the slots are placed on top of each using a vertical configuration, as shown in Table 6. The results show that dividing the slots in a vertical style is inefficient regarding reconfiguration time with respect to the horizontal configuration. However, it has the advantage of making the PBS relocatable in all the slots. In conclusion, in this scenario, medium-grain reconfiguration improves reconfiguration time up to 4.8×.

In the second scenario, we assess the performance of medium grain reconfiguration to build a scalable 2D architecture. We compare it to a solution where all the possible architecture sizes are pre-implemented using coarse grain reconfiguration in an island-based VA. The theoretical architecture has only one PE that is replicated in all the locations. The architecture is implemented in a clock region of the the Xilinx *Zynq-7020* SoPC. When using coarse grain reconfiguration, the whole RR is used. In contrast, when using medium grain reconfiguration, the RR is virtually divided into a 5×5 grid, where we allocate only the necessary PEs.

The time needed to reconfigure the architecture with coarse grain reconfiguration is 9021 μs.In contrast, when building a scalable architecture, the reconfiguration time depends on the number of PEs that have to be reconfigured. Table 7 shows the reconfiguration time required to increase the size of the 2D architecture for different configurations. The results show that, in this scenario, using medium grain reconfiguration is more efficient.

Using the medium reconfiguration granularity also reduces the memory footprint and the time required to implement all the RMs. Using medium grain reconfiguration reduces the total number of RMs from 25 (one for each architecture size) to 5 (one PE for each column). The memory is reduced from 10320 (25 x 412.8) kB to 83 (5 x 16.6) kB, a 124× reduction. Similarly, the time required for implementing the RMs is significantly reduced, although the actual reduction is application dependent.

TABLE 8. Performance comparison of different REs.

RE	Read frame (μs)	Write frame (μs)	Total time (μs)
HWICAP [49]	111.5	100.5	212
MiCAP [50]	97	95	192
MiCAP-PRO [48]	23	23.1	46.1
MiCAP-PRO [48] (no read back)	0	23.1	23.1
IMPRESS HW RE (parameters and muxes)	0	6	6
IMPRESS HW RE (FU)	0	0	5.2
IMPRESS SW RE (parameters and muxes)	0	7.5	7.5
IMPRESS SW RE (FU)	0	0	7.7

TABLE 9. Resource utilization of different REs.

RE	FFs	LUTs	BRAMs
HWICAP	675	500	1
MiCAP	221	290	0
MiCAP-PRO	$\approx$ 3*HWICAP = 2025	$\approx$ 3*HWICAP = 1500	-
IMPRESS RE	1189	874	1

C. FINE GRAIN

The fine grain RE is specialized in modifying the fine grain reconfigurable parameters, multiplexers, and FUs supported by IMPRESS. It has been designed in two different flavors: as a software component and as a hardware module. Both deliver the same functionality, with different resource/performance metrics. Table 8 shows a comparison of the performance of both implementations (i.e., hardware and software) of the IMPRESS fine grain RE compared to other REs found in state of the art. The IMPRESS RE outperforms up to 3.8 times the reconfiguration time for multiplexers and parameters, and up to 4.4 times for FUs. This results from using the compressed representation for LUT's columns, the design of the components which require modifying a reduced number of frames and the avoidance of a readback step due to IMPRESS placement constraints. It must be noted that data in Table 8 has been obtained with the REs working at 100 MHz. However, as shown in [47], it is still possible to reduce the reconfiguration time by overclocking the ICAP.

Regarding the resource utilization, Table 9 shows a comparison with existing solutions in the literature. It can be seen that the hardware implementation of the RE uses 2.35 times more logic than the HWICAP (available in Xilinx IP library), whereas the MiCAP-Pro consumes 3x more logic than the HWICAP [48].

The performance offered by fine grain parameters to substitute bus infrastructures as a mechanism to exchange data with the hardware accelerators configured in the programmable logic is discussed next. Since the minimum reconfigurable unit spans a whole clock region in Xilinx FPGAs, fine grain reconfiguration is highly dependent on the column's occupation with fine grain elements. Each frame can contain up to 12 32-bit parameters, which can be reconfigured simultaneously. Table 10 shows the time needed to reconfigure a different number of 32-bit parameters with fine grain reconfiguration, compared to data writing operations in memory-mapped registers using an AXI lite bus. When using the AXI bus, the time to change the parameters is proportional to the total number of parameters, needing approximately

TABLE 10. Performance of fine grain reconfiguration and AXI lite for different number of parameters.

number of 32-bit parameters	HW fine grain RE (μs)	SW fine grain RE (μs)	AXI lite μs
6	0.4 + 6 = 6.4	0.4 + 7.5 = 7.9	1.3
12	0.8 + 6 = 6.8	0.8 + 7.5 = 8.3	2.6
24	1.5 + 9.1 = 10.6	1.5 + 14.7 = 16.2	5.1
36	2.1 + 12.1 = 14.2	2.1 + 22.2 = 24.3	7.6

0.21 (μs) per each one. In contrast, when using fine grain reconfiguration, the time is spent in two phases. When a user changes a fine grain parameter, the IMPRESS run-time software has to find the parameter's location and modify its internal configuration. Once the user has selected the new value for all the parameters, the new configuration is sent to the fine grain RE that reconfigures the selected frames. The time spent in the first phase is proportional to the number of parameters changed (i.e., approximately 0.06 μs per parameter). However, the time spent by the RE depends on the number of frames that have to be reconfigured. In this case, for each group of 12 parameters, a single frame has to be reconfigured. In the software implementation of the RE, the time to reconfigure multiple frames is proportional to the total number of frames (approximately 7.5 μ per frame). Differently, the hardware version has been designed to be more efficient when reconfiguring multiple frames. Thus, the time to reconfigure the first frame is higher than the subsequent frames (i.e., the first frame is reconfigured in 6 μs while the following frames are reconfigured 3 μs).

Finally, we carried out a comparison between the resource occupancy in fine grain reconfigurable FUs compared to a non-reconfigurable alternative, in which all the required functions are implemented simultaneously as a single Functional Unit, and the functionality is configured by using a multiplexer which connects the output of the FU with the results produced by the selected function. Taking the list of functions in Table 4 as a reference, the number of resources used would amount to 528 LUTs for a 32-bit non-reconfigurable FU while the area occupancy for a fine grain FU with the same functionality os reduced to 64 LUTs.

IX. USE CASES

Two different use cases have been implemented in a Xilinx *Zynq-7020* SoPC to show how IMPRESS can be used to build reconfigurable systems with multiple granularity levels. The first scenario combines the three different granularities in an image-processing pipeline. The second use case combines medium and fine granularity levels for implementing a scalable block-based neural network (BbNN).

A. IMAGE PROCESSING PIPELINE

Coarse, medium and fine grain levels have been combined to design a reconfigurable architecture based on a streaming pipeline for image processing. The static system is composed of the Zynq Processing System (i.e., the integrated ARM

TABLE 11. Filter reconfiguration times.

	Readback (μs)	recombination (μs)	download (μs)	**total (μs)**
fast corner	2434	867	2804	**6105**
sobel	1101	315	1435	**2851**
dilate	1236	355	1467	**3058**

TABLE 12. Filter resource utilization.

	LUT	FF	DSP	BRAM
fast corner	4327 (80.13%)	3353 (31.05%)	-	22.5 (83.33%)
dilate	902 (31.32%)	763 (13.25%)	-	6 (33.33%)
/sobel	702 (27.86%)	517 (10.26%)	-	1.5 (8.33%)

processor), a Video Direct Memory Access (VDMA) block to move images from and to the memory RAM and an empty black-box cell that represents the RR where different image filters will be allocated. The filters have an AXI stream interface for the transference of the image pixels. The filter settings (e.g., image size and start/stop signals) have been implemented with fine-grain parameters avoiding the utilization of AXI lite interfaces. Using AXI interfaces can limit the maximum number of filters that can be configured in this architecture since this number cannot exceed the number of AXI lite interfaces that have been instantiated in the static system at design-time. In contrast, when using fine grain reconfiguration, the number of filters that can be allocated in the RR is only limited by the number of reconfigurable resources available in the region. Therefore, achieving a more flexible usage of the reconfigurable area.

The implementation of the static system is shown in Figure 16a, for which three filters have been designed: the *fast corner*, the *dilate* and *sobel* filters. There is only one RR in the static system that can be used to allocate one RM that occupies the whole RR, such as the *fast corner* filter shown in Figure 16b. The RR can also be used to allocate several filters connected, without using static resources, in a pipeline style, such as the *dilate* and *sobel* filters shown in Figures 16c and 16d.

The PBS size of the fast corner filter is 244.8 kbytes, which is approximately two times bigger than the sobel (121 kbytes) and dilate (123.8 kbytes) filters. As shown in Table 11 it takes 6105 μs to download the fast corner, of which 2434 μs are for the readback, 867 μs for the bitstream recombination and finally, 2804 μs to write the contents in the configuration memory. In contrast, when reconfiguring the sobel filters it takes 2851 μs (1101 μs in the readback, 315 μs in the bitstream recombination and 1435 μs to write the contents in the configuration memory) to download the PBS. Table 12 shows the resource utilization of each filter.

This example shows how IMPRESS can be used to build an image filter application where a RR can allocate one filter using coarse grain reconfiguration or several filters in a pipeline style using medium grain reconfiguration. The main benefit of using medium grain reconfiguration in an image

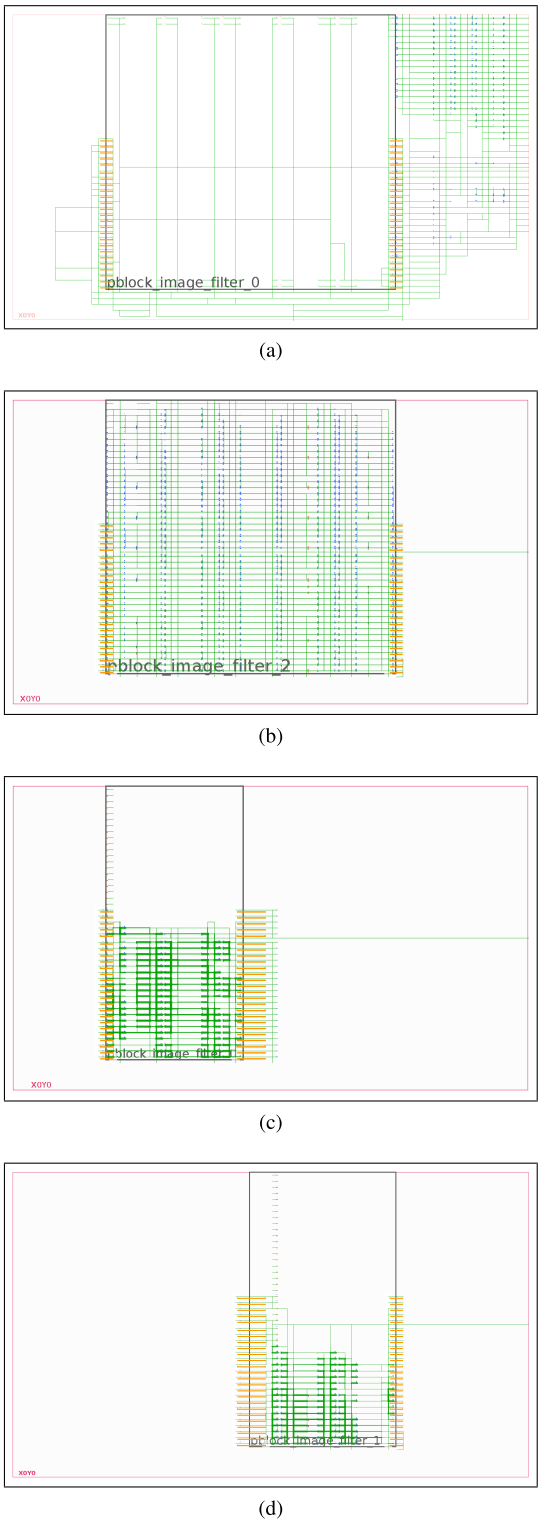


FIGURE 16. These figures show the implementation in Vivado of (a) the static system and different RMs implemented in different RRs: (b) fast corner filter (c) dilate filter and (d) sobel filter. Orange lines represent the interface of the RR while the green lines represent the routes between logic components.

filter pipeline arises when multiple filters can be combined in multiple configurations. Using coarse grain reconfiguration would require implementing a different RM for each pipeline

configuration. In contrast, when using medium grain reconfiguration, the user only has to design an RM for each filter and then combine them at run-time. Therefore, using medium grain reconfiguration simplifies the design and reduces the implementation time and the partial bitstreams' memory footprint. Moreover, if the user wants to change the filter pipeline to another combination, only the filters that differ have to be reconfigured, thus reducing the reconfiguration time. It is important to notice that the filters that are shared can vary between different pipeline combinations. Using fine grain parameters allows controlling the filters without needing a bus infrastructure for each filter. This allows changing the number of filters in the pipeline without being limited by the number of bus interfaces allocated at design time.

B. BLOCK-BASED NEURAL NETWORK

A BbNN is a particular type of artificial neural network, initially proposed in [51], in which neurons are arranged in a 2D grid of processing elements (PEs). An Evolutionary Algorithm (EA) conducts the training process of the BbNN. The EA tries to optimize the weights, bias, and topology of the network, as part of an iterative process in which a considerable amount of candidate circuits are evaluated. For this reason, reconfiguration time is a critical factor in this application. The reader is referred to the work in [51] for further information on the training process.

The second use case showing IMPRESS features is a scalable BbNN, originally published by the authors in [52], whose network size can be used by the EA as a configuration parameter during training. To build an scalable BbNN, we have combined medium and fine granularities. Medium grain reconfiguration is exploited to allocate individual PEs inside the RR. This technique allows adding or removing PEs from the structure at run-time. Therefore, the number of reconfigurable resources that the BbNN utilizes is proportional to its size; Leaving unused reconfigurable resources free to be used by other RMs. Fine grain is applied to change the internal settings of each PE in the network (so providing fast adaptivity during training), as well as to provide input data to the BbNN. Using fine grain reconfiguration to change the configuration of the PEs avoids having to implement direct connections from each PE to the static system. Additionally, the BbNN input modules' position depends on the BbNN size (see subsection VII-C), so there is no guarantee that they are located in the region border. Differently, output PEs are fixed at the west RR border, so an AXI interface is directly used to gather the processed results.

The static system is composed of the Zynq PS, the coarse and medium grain RE, a BbNN controller, and an empty RR used to allocate the BbNN. The whole system operates at 100 MHz, except the RE, which works at 175 MHz. The RR can allocate up to 3×5 PEs. The static system's implementation is shown in Figure 17, with the BbNN outputs placed in the west border of the RR.

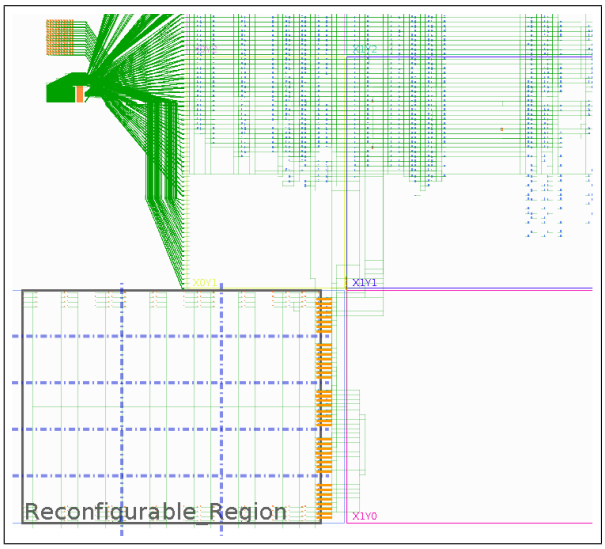


FIGURE 17. Layout of the BbNN static system implementation. We have depicted on top of the empty RR the grid where the PEs are allocated at run-time to build the scalable BbNN.

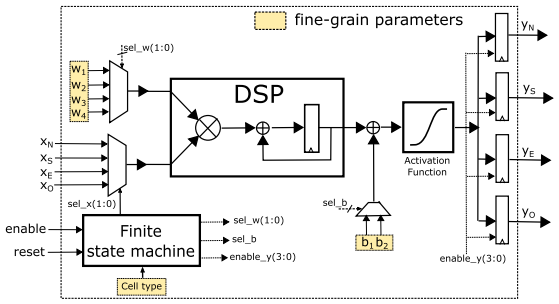


FIGURE 18. BbNN PE structure. Fine-grain parameters are represented as yellow rectangles.

The internal structure of the PEs has been modified compared to the proposal in [51]. We have reduced the number of DSPs per PE to one by reutilizing it through the seven clock cycles required for processing each input data. The weight values, bias values, and the PE configuration are implemented using fine grain reconfigurable parameters, as shown in Figure 18. The layout of the PE is shown in Figure 19, and its resource utilization in Table 13.

Table 14 shows the time needed to reconfigure the entire PE of the BbNN using medium grain reconfiguration. The total time is 2104 μs (887 μs for readback, 50 μs for bitstream recombination, and 1167 μs to write the contents in the configuration memory).

Table 15 shows the time needed to change the parameters of a 3×3 and a 3×5 BbNN with fine grain reconfiguration using the hardware-based fine grain RE, compared to the use of an AXI lite interface in a non-reconfigurable BbNN working at 100 MHz (the non-reconfigurable BbNN cannot work at higher frequencies). In this example, it must be noticed that scaling the BbNN from 3×3 to a 3×5 size does not have much impact on the reconfiguration time. The reason is that the number of LUT columns to be reconfigured is not

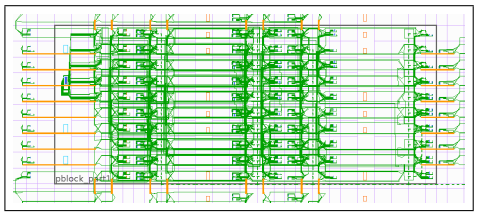


FIGURE 19. BbNN PE layout in Vivado.

TABLE 13. BbNN processing element resource utilization.

	LUT	FF	DSP	BRAM
BbNN processing element	473 (98.54%)	163 (16.98%)	1 (25%)	-

TABLE 14. PE reconfiguration time.

	Readback (μs)	recombination (μs)	download (μs)	**total** (μs)
1 PE (medium grain)	887	50	1167	**2104**

TABLE 15. Performance comparison for different BbNN sizes using fine grain and AXI lite.

BbNN size	Input data (μs)		BbNN configuration (μs)	
	fine grain	AXI	fine grain	AXI
3x3 BbNN	4.3	0.65	22.2	6.6
3x5 BbNN	4.5	1	26.2	10.9

increased, and so the number of frames remains the same. The time difference is due to the execution of the software routine in charge of finding in which device column each network parameter is located. Table 15 shows that the AXI lite is more efficient. However, when using larger sizes in the BbNN, the fine grain reconfiguration performance increases.

X. CONCLUSION AND FUTURE WORK

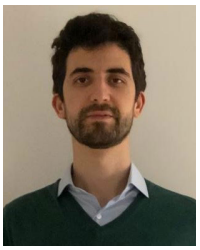
In this article, we propose an integrated view of multi-grain reconfigurable systems, complemented with a critical analysis of how to combine effectively coarse, medium, and fine grain reconfigurable components in real systems. This discussion is presented in multiple scenarios, in which coarse grain reconfiguration is proposed to swap monolithic reconfigurable accelerators, fine grain reconfiguration is used to change individual elements of a circuit when fast and low overhead reconfiguration is required, and medium grain reconfiguration is the preferred option when accelerators are 2D regular architectures that can be reconfigured or scaled at run-time by changing specific building modules. Each granularity level has different implementation requirements, which have been implemented in IMPRESS, an automated tool proposed for the design of multi-grain reconfigurable systems. In the particular case of fine grain reconfiguration, different parameterized HDL components with built-in reconfigurable features are provided so that the users can instantiate them in their design. Apart from the provided design features, IMPRESS provides two different run-time reconfiguration mechanisms tailored for each type of granularity, a software version for the coarse and medium grains, and a specialized reconfiguration engine for fast fine grain reconfiguration implemented both as a hardware and a software component. Two different use cases have been implemented in a commercial SoPC to show how IMPRESS can be successfully used to build reconfigurable systems with these multiple granularity levels in the neural network and video processing domains. These use cases show how the granularity selected affects the flexibility, reconfiguration time, and the memory footprint of the system under design.

Future work involves improving the fine grain capabilities of IMPRESS. In particular, we aim to investigate the possibility of reading the internal state of flip-flops, writing and reading the content of Block RAMs, as well as on the implementation of new domain-specific functional units.

REFERENCES

[1] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proc. IEEE*, vol. 103, no. 3, pp. 332–354, Mar. 2015.
[2] 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 Proc.-Comput. Digit. Techn.*, vol. 152, no. 2, pp. 193–207, Mar. 2005.
[3] M. D. Santambrogio, "From reconfigurable architectures to self-adaptive autonomic systems," in *Proc. Int. Conf. Comput. Sci. Eng.*, vol. 2, Aug. 2009, pp. 926–931.
[4] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: A survey and a cost model," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, pp. 1–24, Dec. 2011, doi: 10.1145/2068716.2068722.
[5] K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–39, Sep. 2018, doi: 10.1145/3193827.
[6] S. Vassiliadis and D. Soudris, *Fine-and Coarse-Grain Reconfigurable Computing*, vol. 16. New York, NY, USA: Springer, 2007.
[7] S. Kung, "VLSI array processors," *IEEE ASSP Mag.*, vol. 2, no. 3, pp. 4–22, Jul. 1985.
[8] S.-Y. Kung, Arun, Gal-Ezer, and B. Rao, "Wavefront array processor: Language, architecture, and applications," *IEEE Trans. Comput.*, vol. C-31, no. 11, pp. 1054–1066, Nov. 1982.
[9] G. Stitt and J. Coole, "Intermediate fabrics: Virtual architectures for near-instant FPGA compilation," *IEEE Embedded Syst. Lett.*, vol. 3, no. 3, pp. 81–84, Sep. 2011.
[10] S. G. Merchant and G. D. Peterson, "Evolvable block-based neural network design for applications in dynamic environments," *VLSI Des.*, vol. 2010, p. 4, Feb. 2010, doi: 10.1155/2010/251210.
[11] R. Zamacola, A. Garcia Martinez, J. Mora, A. Otero, and E. de La Torre, "IMPRESS: Automated tool for the implementation of highly flexible partial reconfigurable systems with Xilinx Vivado," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2018, pp. 1–8.
[12] R. Zamacola, A. Garcia Martinez, J. Mora, A. Otero, and E. de la Torre, "Automated tool and runtime support for fine-grain reconfiguration in highly flexible reconfigurable systems," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2019, p. 307.
[13] Xilinx, "7 series FPGAs configuration user guide," Xilinx, San Jose, CA, USA, Tech. Rep. UG470, Aug. 2018.
[14] (2020). *SymbiFlow Prjxray*. [Online]. Available: https://github.com/SymbiFlow/prjxray
[15] A. Otero, E. de la Torre, and T. Riesgo, "Dreams: A tool for the design of dynamically reconfigurable embedded and modular systems," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Dec. 2012, pp. 1–8.
[16] D. Koch, J. Torresen, C. Beckhoff, D. Ziener, C. Dennl, V. Breuer, J. Teich, M. Feilen, and W. Stechele, "Partial reconfiguration on FPGAs in practice—Tools and applications," in *Proc. ARCS*, Feb. 2012, pp. 1–12.
[17] Xilinx, "Partial reconfiguration user guide," Xilinx, San Jose, CA, USA, Tech. Rep. UG909, Apr. 2018.

[18] C. Beckhoff, D. Koch, and J. Torresen, "Go ahead: A partial reconfiguration framework," in *Proc. IEEE 20th Int. Symp. Field-Program. Custom Comput. Mach.*, Apr. 2012, pp. 37–44.
[19] D. Koch, C. Beckhoff, and J. Teich, "Recobus-builder—A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs," in *Proc. Int. Conf. Field Program. Log. Appl.*, Sep. 2008, pp. 119–124.
[20] A. Otero, E. De La Torre, T. Riesgo, T. Cervero, S. Lopez, G. Callico, and R. Sarmiento, "Run-time scalable architecture for deblocking filtering in H.264/AVC-SVC video codecs," in *Proc. 21st Int. Conf. Field Program. Log. Appl.*, Sep. 2011, pp. 369–375.
[21] Intel, "Partial reconfiguration user guide," Intel, Santa Clara, CA, USA, Tech. Rep. UG-20136, Nov. 2019.
[22] T. Drahonovsky, M. Rozkovec, and O. Novak, "Relocation of reconfigurable modules on Xilinx FPGA," in *Proc. IEEE 16th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2013, pp. 175–180.
[23] L. Gantel, M. E. A. Benkhelifa, F. Lemonnier, and F. Verdier, "Module relocation in heterogeneous reconfigurable Systems-on-Chip using the Xilinx isolation design flow," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Dec. 2012, pp. 1–6.
[24] C. Beckhoff, D. Koch, and J. Torresen, "The xilinx design language (XDL): Tutorial and use cases," in *Proc. 6th Int. Workshop Reconfigurable Commun.-Centric Syst.–Chip (ReCoSoC)*, Jun. 2011, pp. 1–8.
[25] J. Rettkowski, K. Friesen, and D. Gohringer, "RePaBit: Automated generation of relocatable partial bitstreams for Xilinx Zynq FPGAs," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Nov. 2016, pp. 1–8.
[26] B. Gottschall, T. Preußer, and A. Kumar, "Reloc—An open-source Vivado workflow for generating relocatable end-user configuration tiles," in *Proc. IEEE 26th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr./May 2018, p. 211.
[27] R. Oomen, T. Nguyen, A. Kumar, and H. Corporaal, "An automated technique to generate relocatable partial bitstreams for xilinx FPGAs," in *Proc. 25th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2015, pp. 1–4.
[28] M. Nguyen and J. C. Hoe, "Amorphous dynamic partial reconfiguration with flexible boundaries to remove fragmentation," 2017, *arXiv:1710.08270*. [Online]. Available: http://arxiv.org/abs/1710.08270
[29] K. Dang Pham, E. Horta, and D. Koch, "BITMAN: A tool and API for FPGA bitstream manipulations," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 894–897.
[30] A. Vaishnav, K. D. Pham, D. Koch, and J. Garside, "Resource elastic virtualization for FPGAs using OpenCL," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 111–1117.
[31] S. Ma, Z. Aklah, and D. Andrews, "A run time interpretation approach for creating custom accelerators," in *Proc. 25th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2015, pp. 1–4.
[32] S. Ma, D. Andrews, S. Gao, and J. Cummins, "Breeze computing: A just in time (JIT) approach for virtualizing FPGAs in the cloud," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Nov. 2016, pp. 1–6.
[33] M. Nguyen and J. C. Hoe, "Time-shared execution of realtime computer vision pipelines by dynamic partial reconfiguration," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 230–2304.
[34] A. Sudarsanam, A. Dasu, R. Kallam, J. Carver, and R. Barnes, "Dynamically reconfigurable systolic array accelerators: A case study with extended Kalman filter and discrete wavelet transform algorithms," *IET Comput. Digit. Techn.*, vol. 4, no. 2, pp. 126–142, Mar. 2010.
[35] A. Otero, E. de la Torre, T. Riesgo, and Y. E. Krasteva, "Run-time scalable systolic coprocessors for flexible multimedia SoPCs," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2010, pp. 70–76.
[36] A. Gallego, J. Mora, A. Otero, R. Salvador, E. de la Torre, and T. Riesgo, "A novel FPGA-based evolvable hardware system based on multiple processing arrays," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum*, May 2013, pp. 182–191.
[37] E. Eto, "Difference-based partial reconfiguration," Xilinx, San Jose, CA, USA, Tech. Rep. XAPP290, Dec. 2007.
[38] C. H. Hoo and A. Kumar, "An area-efficient partially reconfigurable crossbar switch with low reconfiguration delay," in *Proc. 22nd Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2012, pp. 400–406.
[39] R. Backasch and C. Hochberger, "Custom reconfigurable architecture based on Virtex 5 lookup tables," in *Architecture of Computing Systems–ARCS*, H. Kubátová, C. Hochberger, M. Daněk, and B. Sick, Eds. Berlin, Germany: Springer, 2013, pp. 183–194.
[40] E. Vansteenkiste, K. Bruneel, and D. Stroobandt, "Maximizing the reuse of routing resources in a reconfiguration-aware connection router," in *Proc. 22nd Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2012, pp. 322–329.
[41] E. M. Abdali, M. Pelcat, F. Berry, J.-P. Diguet, and F. Palumbo, "Exploring the performance of partially reconfigurable point-to-point interconnects," in *Proc. 12th Int. Symp. Reconfigurable Commun.-Centric Syst.–Chip (ReCoSoC)*, Jul. 2017, pp. 1–6.
[42] L. Bozzoli and L. Sterpone, "Self rerouting of dynamically reconfigurable SRAM-based FPGAs," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jul. 2017, pp. 77–84.
[43] J. Mora and E. de la Torre, "Accelerating the evolution of a systolic array-based evolvable hardware system," *Microprocessors Microsyst.*, vol. 56, pp. 144–156, Feb. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933117305483
[44] K. Bruneel, W. Heirman, and D. Stroobandt, "Dynamic data folding with parameterizable FPGA configurations," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 16, no. 4, p. 43, Oct. 2011, doi: 10.1145/2003695.2003703.
[45] F. Abouelella, T. Davidson, W. Meeus, K. Bruneel, and D. Stroobandt, "How to efficiently implement dynamic circuit specialization systems," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 18, no. 3, p. 35, Jul. 2013, doi: 10.1145/2491477.2491479.
[46] (2020). *IMPRESS Tool*. [Online]. Available: https://des-cei.github.io/tools/impress
[47] S. G. Hansen, D. Koch, and J. Torresen, "High speed partial run-time reconfiguration using enhanced ICAP hard macro," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 174–180.
[48] A. Kulkarni and D. Stroobandt, "How to efficiently reconfigure tunable lookup tables for dynamic circuit specialization," *Int. J. Reconfigurable Comput.*, vol. 2016, pp. 1–12, 2016.
[49] Xilinx, "AXI HWICAP LogiCORE IP product guide," Xilinx, San Jose, CA, USA, Tech. Rep. PG134, Oct. 2016.
[50] A. Kulkarni, V. Kizheppatt, and D. Stroobandt, "MiCAP: A custom reconfiguration controller for dynamic circuit specialization," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2015, pp. 1–6.
[51] S.-W. Moon and S.-G. Kong, "Block-based neural networks," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 307–317, Mar. 2001.
[52] A. García, R. Zamacola, A. Otero, and E. de la Torre, "A dynamically reconfigurable BbNN architecture for scalable neuroevolution in hardware," *Electronics*, vol. 9, no. 5, p. 803, May 2020.



RAFAEL ZAMACOLA received the M.Sc. degree in industrial engineering from the Universidad Politécnica de Madrid (UPM), Spain, in 2014, where he is currently pursuing the Ph.D. degree. He has worked three years as a Research and Development Engineer of developing electromedical devices. His research interests include reconfigurable tools and architectures on FPGAs and evolvable hardware.

ANDRÉS OTERO (Member, IEEE) received the M.Sc. degree (Hons.) in telecommunication engineering from the University of Vigo, in 2007, and the Master of Research and Ph.D. degrees in industrial electronics from the Universidad Politécnica de Madrid (UPM), in 2009 and 2014, respectively. He is currently an Assistant Professor of electronics with UPM and a Researcher with the Centro de Electrónica Industrial (CEI). His current research interests include embedded system design, reconfigurable systems on FPGAs, evolvable hardware, and embedded machine learning. During the last years, he has been involved in different research projects in these areas. He is the author of more than 30 papers published in international conferences and journals. He has served as the Program Committee Member of different international conferences in the field of reconfigurable systems, such as SPL, ERSA, ReConFig, DASIP and ReCoSoC.

ALBERTO GARCÍA received the M.Sc. degree in industrial engineering from the Universidad Politécnica de Madrid (UPM), Spain, in 2019, where he is currently pursuing the Ph.D. degree. He was on an internship with the Centro de Electrónica Industrial (CEI) during the last year of his studies. His research interests include development of neural networks on FPGAs and evolutionary algorithms.

EDUARDO DE LA TORRE received the Ph.D. degree in electrical engineering from UPM, in 2000. He is currently an Associate Professor of electronics with the Universidad Politécnica de Madrid (UPM), Spain, doing his research with the Centre of Industrial Electronics. His main expertise is in FPGA design, embedded systems design, HW acceleration, signal processing, and partial and dynamic reconfiguration of digital systems. He has participated in more than 40 projects, eleven of them being EU funded projects and, overall, in nine funded projects related with reconfigurable systems. He has been a Program Chair of ReConFig, the General Chair of ReCoSoC, two conferences with strong interest in hardware reconfiguration.