IMPRESS: Automated Tool for the Implementation of Highly Flexible Partial Reconfigurable Systems with Xilinx Vivado

Rafael Zamacola, Alberto García Martínez, Javier Mora, Andrés Otero and Eduardo de La Torre

Centro de Electrónica Industrial, Universidad Politécnica de Madrid

Email: {rafael.zamacola, javier.morad, joseandres.otero, eduardo.delatorre}@upm.es

Abstract—Dynamic partial reconfiguration is considered a great technique to increase flexibility in FPGA designs. However, partial reconfiguration flows supported by commercial tools, such as Xilinx Vivado, still have many limitations. Foremost among them are the lack of support for relocation, which leads to an increase in the on-system memory requirements and the synthesis time, as well as a reduced flexibility when it comes to the definition of reconfigurable regions. Several academic tools have appeared over the years to improve commercial flows. However, the technology shift from ISE to Vivado has left most of these tools unusable for newer FPGAs, including most of the Xilinx Series-7 devices. In this paper, authors present IMPRESS, a TCL script-based tool for the automated generation of relocatable partial bitstreams under Vivado, with a strong focus on the ease of use and the system flexibility. Special support is provided for the implementation of reconfigurable systems that include IP blocks generated with Vivado HLS and standardized bus interfaces. A stream-based reconfigurable architecture for image filtering, implemented in a fully automated manner on a Zynq SoPC, is provided as a use case of the tool.

Keywords—Partial Reconfiguration; Relocation; Design Tool; Reconfigurable Image Processing; HLS

I. INTRODUCTION

Dynamic Partial Reconfiguration (DPR) increases the flexibility of FPGA designs by enabling the reuse of logic resources over time among different reconfigurable modules, provided they are mutually exclusive. Silicon reuse makes possible the use of smaller FPGAs for the same design, thus reducing the overall cost and power consumption of the final system. Apart from that, it enables the adaptation of the system by configuring under demand new processing Intellectual Property (IP) cores according to the changing requirements and scenarios the system may face.

A dynamically reconfigurable system is made up of a static system, which does not change during system lifetime, and one or multiple **Reconfigurable Partitions** (RP), where different **Reconfigurable Modules** (RM) can be allocated in real time. Commercial tools available for the generation of DPR systems, such as Vivado from Xilinx, enforce the generation of one **partial bitstream** (PBS) for every reconfigurable module that is intended to be allocated in each reconfigurable partition. Thus, if the system contains m RMs that are allocable in n different RPs, it is necessary to generate $m \times n$ PBS. This leads to an increase in the synthesis time and the on-system memory requirements. An alternative is the generation of relocatable PBS, which can be configured in any compatible RP. To do so,

it is necessary to impose additional constraints during the synthesis of the RMs and the definition of the RPs.

Different tools have been proposed over the years in the academia [1-7] with the aim of extending the capabilities offered by Xilinx commercial tools for the design of reconfigurable systems. Most of these tools worked under the ISE design suite, and they based their operation on the proprietary **Xilinx Design Language** (XDL) [10]. XDL offered a human-readable view to both the internal resources of the FPGA as well as to the design netlists. However, the technology shift from ISE to Vivado made all these tools unusable for designs on newer FPGAs. XDL was discontinued, and it was substituted in Vivado by TCL commands, which also give the user total control on the design. Some tools leveraging on TCL commands [8] [9] have also been proposed having in mind the generation of PBS.

In this paper, the authors present IMPRESS, a new tool that aims at automating the generation of DPR systems. The tool is based on a set of TCL scripts working under Xilinx Vivado and it offers the generation of relocatable partial bitstreams. On top of that, the proposed tool is focused on increasing the flexibility of the reconfigurable designs produced with it, trying to get the most benefit from the use of DPR. At this regard, one of the main limitations of the Xilinx design flow covered by IMPRESS is the inability to stack vertically multiple RPs within a clock region. This way, it allows producing RPs with a finer granularity, reducing the amount of logic blocks wasted by the RMs with lower resource requirements.

Moreover, by enabling reconfigurable-to-reconfigurable module communication, the tool allows the distribution of logic resources in each RP at run-time, without requiring a fixed virtualization (definition of the floorplanning and the inter-module communication) at design time. This way, whenever the user respects the communication infrastructure with the static system and between RPs, the reconfigurable area can be distributed in a flexible way at run-time. An example of the benefits of this unrestricted definition of the reconfigurable regions are dynamically scalable systolic architectures described in [12], where reconfigurable modules are directly connected to compose dynamically scalable accelerators with the dimensions required by the system at run-time. This enables a dynamic trade-off between the quality of service provided by the accelerator and the amount of logic resources it occupies. IMPRESS also supports the implementation of hierarchical reconfigurable designs, which contain RPs with other RPs inside.

The proposed tool decouples the implementation of the static system and the reconfigurable modules making possible the validation of part of the design without the other. Moreover, this feature is useful to share a RM among several projects and in designs where the virtual architecture is going to be changed at runtime. To this end, the tool includes a parameterized **blackbox IP**, which can be integrated into static designs as a placeholder that is substituted at run-time by PBS, generated independently. Blackbox IP makes easier the generation of reconfigurable modules from IPs featured with standard AXI interfaces, implemented with Vivado HLS. Apart from the run-time reconfiguration of the generated bitstreams, IMPRESS also supports the composition of the static system and any combination of RMs for every RP, at design time. This aims at the functional validation of all the possible configurations of the system, as well as the generation of timing reports.

The rest of the paper is organized as follows. Main design rules for reconfigurable systems are provided in section 2. The State-of-the-Art is discussed in section 3, while the proposal for system specification with IMPRESS is presented in section 4. Section 5 details the proposed implementation flow, while a reconfiguration engine compatible with relocation is described in section 6. The use case for image processing is shown in Section 7. Finally, conclusions and future work are shown in Section 8.

II. DESIGN RULES FOR PARTIAL RECONFIGURABLE SYSTEMS

Partial reconfigurable systems must comply with specific design rules to guarantee their functional correctness. On top of that, specific constraints must be applied to generate bitstreams compatible with relocation. Differently from existing commercial tools, the solution proposed in this paper provides automatic mechanisms to address the requirements explained below, and so to ensure compliance with relocation without needing further manual effort.

A. Compatibility of Resource Footprints

Partial reconfiguration requires the generation of partial configuration bitstreams defining only the content of the reconfigurable region of the FPGA where the RM will be allocated. If the same PBS is intended to be reconfigured in multiple reconfigurable regions, the distribution of logic resources (i.e., the position of LUTs, BRAMs and DSP columns) must be the same for all these regions. However, it does not mean that the RPs must be necessarily equal. One RP can be a superset of other RP while fulfilling with the footprint compatibility requirements, making relocation still possible. The proposed tool exploits this feature enabling the implementation of flexible reconfigurable architectures without requiring a fixed virtualization at design time.

B. Compatibility of Shared Interfaces

All the RMs that can be allocated in one RP must share a common interface to the static system or to other RPs. This means that equivalent input and output ports in the different modules must use the same nodes of the device to cross the partition borders. However, not all the RMs must use all the signals in the interface. A special case occurs when the RM uses standardized bus-based interfaces, since the signals to be included in the reconfigurable interface will be known in

advance, making easier the implementation of compatible modules.

The generation of reconfigurable interfaces has evolved over the years, along with the different Xilinx PR design flows. From bus macros proposed in the early flows, to proxy logic, and finally, to partition pins [11]. A bus-macro consists of a pair of LUTs, each one placed on each side of the RP border, connected through a fixed set of wires. LUTs and wires jointly conform a hard macro that can be instantiated in every RP, guaranteeing the correctness of the interface. Differently, partition pins use a shared configurable node as the interface point for each of the crossing signals. This avoids the delay and area overhead introduced by LUTs in bus-macros.

In turn, relocation requires that all the RPs where a given module is to be reconfigured must also share the same physical interface. Therefore, once a reconfigurable interface is defined for a reference RP, it must be applied in the rest of compatible RPs, in the same relative locations.

C. Physical isolation between the Static System and the RPs

In a non-relocatable reconfigurable design, nets from the static system can use all the routing resources available in the FPGA, including those inside the RPs. The static system will be locked once routed, and therefore, all the RMs for the same RP will include the same *feed-through* nets of the static system. However, *feed-through* nets will be different for each RP, preventing relocation between RMs generated in different RPs. Therefore, relocation needs ensuring that the static system and the RMs are completely isolated, with the exception of the shared interfaces.

Apart from routing, also placement constraints must be applied to guarantee that no logic from the static system is included in a RP, and vice versa.

III. STATE OF THE ART

Different tools have been proposed in the last years targeting the design of relocatable DPR systems [1–9]. The most significant ones are analyzed in this section. A summary of the main features of each tool can be found in Table I.

Starting with the tools intended to be used with Xilinx ISE, DREAMS [5] and GOAHEAD [6] must be highlighted. Both tools can implement reconfigurable interfaces without introducing resource overheads. DREAMS achieves physical isolation by using a custom router that reroutes all the feed-through nets generated by the ISE router in the RPs, after a first routing stage carried out with the commercial tool. On the other hand, GOAHEAD uses blocker macros. Both tools include extra features like reconfigurable to reconfigurable connections, support for multiple RPs stacked vertically in the same clock region, they decouple the static and reconfigurable designs, and the latter includes hierarchical reconfiguration.

R. Oomen presented in [8] the first tool that automates the generation of relocatable DPR designs under Xilinx Vivado. It uses bus-macros to implement the communications between reconfigurable modules and the static system, but instead of using regular LUTs, it relies on connections made of a two-input AND gate, which can be used to decouple the RM from the static

Table I - Comparison of the Main Supporting Tools for Implementing Reconfigurable Systems

Tool	IDE	Routing Isolation	Interface	GUI	Static / reconf. decoupled	Reconfigurable-to-reconfigurable communications	Sub-clock region RPs	Hierarchical reconfiguration	FPGAs
DREAMS [5]	ISE (XDL)	Custom router	Virtual borders	Yes	Yes	Yes	Yes	No	Virtex 5 and Spartan 6
GOAHEAD [6]	ISE (XDL)	Blocker macro	Direct wire binding	Yes	Yes	Yes	Yes	Yes	Virtex 4, 5, 6, 7 and Spartan 6
R. Oomen [8]	Vivado (TCL)	None	Bus macro	No	No	No	No	No	Virtex-7
RePaBit [9]	Vivado (TCL)	IDF	Bus macro	No	No	No	No	No	Zynq SoC
IMPRESS	Vivado (TCL)	Blocker macro	Virtual interface	No	Yes	Yes	Yes	Yes	Zynq SoC

system. To create these interfaces, the most complex RM is implemented in all the RPs, generating different interfaces for each RP. One of these interfaces is chosen as the reference to be replicated into all the relocatable RPs using placement and routing constraints. This work, however, does not address the isolation between the static system and the reconfigurable regions. Thus, it will not provide relocatable bitstreams for all the designs.

To the best of the author's knowledge, the most comprehensible tool for implementing DPR systems under Vivado is RePaBit, developed by J. Rettkowski et al. [9]. It uses both the partial reconfiguration flow (PRF) [13] and Xilinx isolation design flow (IDF) [14] to ensure isolation between the static and the reconfigurable regions. The use of the IDF implies that the designer must reserve a row of unused tiles between the static system and the RPs. The tool has a preparative phase where it checks that all the relocatable RPs are fully compatible and that each RP spans over one clock region. This is followed by an implementation phase where a reference RM is assigned to all the RPs. Then LUTs used for Bus-macro connections are inserted for the static system and the RM. After that, a placement & routing phase is executed using the Xilinx Reconfiguration Flow. Then, the constraints of the reference interface are saved and replicated for all the RPs. Using the previous constraints to generate the common interfaces, a new placement & routing phase is executed using the IDF in order to avoid the insertion of feedthrough nets. Finally, the Reconfiguration Flow is executed again to generate the PBS.

The tool presented in this paper aims at extending the features provided by the works in the state-of-the-art, guaranteeing the compatibility with Xilinx Vivado. Thus, IMPRESS supports module reallocation, trying to reduce resource and latency overheads as much as possible. To that end, feed-through nets are avoided by using blocker macros. Differently to the IDF used in [9], that forces to have a row of unused resources around each RP, blocker macros decouple the RP from the rest of the logic without having to keep unused resources among them. In turn, reconfigurable interfaces are implemented in IMPRESS by selecting fixed nodes in the device, shared between the static and the reconfigurable regions. This solution is named as **Virtual Interface**, and differently to the solutions in [8] and [9], it does not introduce extra overhead.

Furthermore, the tool implements several advanced features that are not present in any Vivado compliant academic tool, such as reconfigurable to reconfigurable connections, it allows

multiple RPs to be stacked vertically in the same clock region, the possibility of decoupling the static and reconfigurable designs, as well as hierarchical reconfiguration. It also enables design time system composition, useful for timing analysis and the system verification. The current version of IMPRESS supports Zynq SoPC devices, but future releases will include support for Xilinx Zynq Ultrascale+. It has been tested with Vivado 2017.3.

IV. SPECIFICATION OF THE RECONFIGURABLE SYSTEM

IMPRESS has been conceived with a strong focus on the ease of use. The user only has to specify the system to be designed, and the tool automatically carries out all the implementation steps up to the generation of the bitstreams.

The system specification is divided into the **Project**, the **Virtual Architecture** (VA) and the **Virtual Interface** (VI) files.

A. Project File

It includes three sections. The first one contains the general settings of the project, such as the FPGA model, the working directory and the path to the IP repositories. The second section details all the design sources corresponding to the static system, in the form of HDL descriptions, block design (.bd) files or synthesized design checkpoints (DCP). In the third section, the designer must specify the design sources for each RM and the type of RP where it can be reallocated.

The tool can implement the static system or any reconfigurable module independently from each other, and so the sections not required in each execution of the tool can remain unfilled.

B. Virtual Interface (VI)

To guarantee the compatibility of the shared interfaces, IMPRESS extracts (and applies) the mapping between each I/O signal of a RM and a shared node, accessible from both sides of the RP border. The data structure that maintains this mapping is referred to as a **Virtual Interface** (VI). As shown in Figure 1 (a), each VI is stored in a file including the I/O information related to a set of compatible RPs. VI file is divided into two sections: the first one covers all the global nets, i.e., nets that use clock resources. The second section contains the information of all the pins connected to local nets. VI files can be created manually by the designer or extracted automatically by the tool.

In order to achieve the maximum flexibility in the definition of RPs, only one-hop nodes located in bordering interconnection

Figure 1 – (a) Example of a north VI and the description in the file, (b) shared nodes used for the interfaces in between two regions.

tiles can be used in a VI, as shown in Figure 1 (b). Using larger wires would prevent the definition of RPs with a single CLB row. The interface is defined by specifying the cardinal direction (*NORTH*, *SOUTH*, *EAST* and *WEST*) and the relative tiles that can be used to make the connections, taking the RP border as the reference. For example, an interface that connects through the first 4 tiles of the North side of the RP to the static system or another RP would be defined as *NORTH_0:3*, as shown in the layout snapshot of Figure 1 (a). If all the tiles in a given border are valid, the VI just specifies their cardinal direction. Figure 1 (a) shows an example of reconfigurable-to-reconfigurable module interconnection.

Defining the interface by specifying which external tiles will be used increases flexibility in reconfigurable-to-reconfigurable module interconnections by allowing one RP to connect, and to be compatible, with multiple RPs with different shapes as long as they share the same virtual interface. Moreover, this feature can be used to change the virtual architecture at runtime, whenever the interfaces with the static system are preserved. Thus, the architecture can be implemented as a single empty RP that contains all the interfaces with the static system, and then the region can be used by the RP as a canvas where smaller RPs could be allocated at runtime.

C. Virtual Architecture (VA)

The virtual architecture refers to the system floorplanning, thus, how logic resources are distributed into RPs. To enable relocation, RPs on the device are arranged into categories, including all the RPs that have a compatible footprint as well as a compatible interface, and therefore, that are relocatable among them. For each RP category, a reference to the Virtual Interface file must be provided. As an advanced feature, a RP can be defined as a hierarchical partition including other RPs inside. This allows combining a coarser block-based reconfiguration granularity with a fine grain modification inside the RM.

V. SYSTEM IMPLEMENTATION FLOW

A graphical representation of the proposed implementation flow is presented in Figure 2. Further details are provided in the rest of the section.

The tool starts by reading the system description files provided as inputs during the tool invocation (project file,

virtual architecture and virtual interface files). Then, it generates the static system, if its definition is included in the project file. Finally, reconfigurable modules are implemented.

A. Static System Generation

The first step is to *synthesize* the static system. Then, the tool defines the *floorplanning* according to the VA file (i.e., it inserts the *Pblocks* and adds the corresponding cells to these *Pblocks*, using Xilinx nomenclature).

Next step is to *generate all the interfaces for the RPs*, according to the definitions provided in the VI files. As it has been explained before, it is possible to leave the interfaces undefined and let the tool to generate them. If this is the case, the procedure is as follows. First, the tool divides the I/O pins of each RP category into groups depending on the RP (or the static system) they are connected with, as shown in Figure 3 (a). For each of the groups of pins, the tool decides which cardinal direction and which edge tiles can be used. This is done by computing the common border between the two RPs (or between the RP and the static system). In the case of the interconnections with the static system, since multiple common borders may exist (also shown in the figure, where south and west borders are shared), cardinal directions are assigned with a priority rule that depends on the quadrant of the FPGA where the center of the RP is located. This aims at reducing (in average) the length of the nets in the interface. Notice that the resulting VI file provides for each pin a range of tiles that can be used for this interconnection, so no specific device nodes are assigned at this point. The generated VI files can be used for future RM implementations.

Once all the *Virtual Interface* files are available, they will be *applied*. This means to assign specific nodes of the device to each RP I/O pin, according to the permitted tile ranges provided in the VI. This assignment is always done maintaining the order of appearance of the pins in the file. In the same way, physical nodes are always taken in a fixed order: tiles in the permitted range are selected from bottom to top, in the case of vertical borders (or left to right, in the case of horizontal borders). Within each tile, up to six device nodes can be used, and they are also used in a fixed order (see Figure 1 (b)). This way, the position agnostic VI file results in a deterministic mapping of RP I/O pins to physical nodes in the device. Once the specific node is assigned, it is introduced in Vivado as a constraint to the Partition Pin (property *HD.PARTPIN_LOCS*).

A blackbox IP including a configurable number of AXI interfaces (both lite and streaming) is also provided within the tool. It can be used as part of the block-based design of the static system, in the case of having a processor-centric architecture with reconfigurable IPs connected by means of standard AXI interfaces. The blackbox can be used as a placeholder that can be later substituted by any reconfigurable IP, whenever they have the same interfaces. This contributes to making more independent the design of reconfigurable modules and the static system.

The next step is to *add dummy lookup tables* (LUTs) inside each RP cell for every interface pin. This is required by Vivado, since its placer cannot work with empty black boxes in the design. Moreover, in the case of global nets (clock, reset or user-

Figure 2- Overview of the proposed DPR system implementation flow

defined signals), several LUTs must be placed on each top and down edge tile of the RP. This ensures that global signals will reach all the columns in the RP after reconfiguration. Global buffers are also instantiated by the tool for the nets considered as global in the Virtual Interface.

Finally, the tool launches the **placement and routing** of the system. In order to ensure the physical isolation between the RPs and rest of the design, blocker macros are used. A blocker macro is a net created with the only purpose to ensure that the Vivado router cannot use the nodes included therein. In this case, the tool creates one global blocker macro to route each global net and a local blocker macro to route the rest of the design. The local blocker macro includes all the nodes shared by the RP and the rest of the system, excluding the nodes used in the Virtual Interface. This is shown in Figure 3 (b), where only the nodes in green are allowed to be used during the routing of a module, guaranteeing the physical isolation from the rest of the system. Blocker macros are deleted before the **bitstream is generated**.

B. Reconfigurable Module generation

The implementation of RMs follows a flow similar to the case of the static system. Main differences are highlighted next.

First, Reconfigurable Modules must be synthesized in **out-of-context mode** to avoid Vivado adding IO buffers to the external pins. The next step is to define the floorplanning, consisting of the main reconfigurable region and the hierarchical sub-regions. After that, for each of these regions, the tool applies the corresponding virtual interface and instantiates the dummy logic. In this case, dummy logic is inserted both to create a dummy static system as well as to convert all the hierarchical RPs from empty black boxes to valid netlists. After that, the tool places and routes the design with a blocker net.

As the static system is a dummy one, it is not possible to use Vivado commands to obtain the PBS. Instead, a custom PBS extractor tool has been implemented to get a PBS from a complete bitstream. It aims at minimizing the memory footprint of the extracted PBSs. So, it does not include the *header* (it would be in any case replaced at run-time for relocation), it only has configuration information of the RP region, i.e., it does not need to contain a whole frame in case of sub-clock region reconfiguration, and it does not include the clock words.

C. Design Time Composition

The tool supports the composition of the static system and any combination of reconfigurable modules for every reconfigurable partition, at design time. This enables the functional validation of all the possible configurations of the system, as well as the generation of timing reports to obtain the maximum clock frequency for each possible configuration.

This feature requires carrying out some extra steps during the implementation flow. In particular, it is necessary to save three different files. The first one contains internal design constraints (using the Vivado *LOCK_PIN*, *BEL*, *LOC* and *ROUTE* properties) saved relatively to the RP. The second file contains partial *ROUTES* of the interface nets from the shared nodes to the RM. The third file is the design checkpoint (DCP) of the synthesized reconfigurable cell.

To combine the RMs with the static system, the corresponding RP is converted into a black box, the DCP is imported, and the internal constraints from the stored files are

Figure 3- In (a) the arrangement of I/O pins into groups is shown, in (b) a blocker-macro is sketched.

applied. Finally, the tool combines previous static interface routes with the ones from the RM to form the complete nets.

VI. SUB-CLOCK REGION RECONFIGURATION ENGINE

The **Reconfiguration Engine** is a software program that runs in the target device and that is in charge of loading into the configuration memory of the FPGA the partial bitstreams generated by IMPRESS. Bitstreams provided by the tool only contain programming information of the reconfigurable module, in a position agnostic way. The Reconfiguration Engine adds the configuration commands and the clock configuration words at run-time, depending on the final partition where the module is to be configured. Run-time bitstream relocation has also been tackled in some other works in the state-of-the-art [15] [16].

The proposed reconfiguration engine also plays a major role in the case of sub-clock region reconfiguration. The atomic reconfigurable unit of a Xilinx FPGA bitstream is called frame. A frame spans the entire clock region height. Therefore, it is not possible to reconfigure at once regions spanning less than the whole height of the clock region. If the entire frame is reconfigured, the content of the logic outside the RP, placed below or above it within the same clock region, will also be affected by the reconfiguration. It is necessary to read the previous content of the configuration memory (which can be part of other RPs or the static system), and to combine it with the PBS that is intended to be reconfigured.

The process to download a PBS is as follows: the reconfiguration engine identifies all the clock rows occupied by the RP. For each row, it reads the configuration from the configuration memory and combines it with the information contained in the PBS generated with the tool (affecting only the RP), thus forming the new content of the configuration memory. Once the bitstream composition is finished, the reconfiguration engine adds a *header*, which includes the location within the FPGA where the PBS will be downloaded, together with some configuration commands. Finally, the composed PBS is downloaded to the FPGA. In the current version of the tool, the RE runs in the Processing System (PS) of the *Zynq-7020*, and the memory is configured using the PCAP port.

VII. USE CASE FOR RECONFIGURABLE IMAGE FILTERING

A reconfigurable architecture based on a streaming pipeline for image processing is provided as a use case to evaluate the benefits of the proposed tool.

A. Reconfigurable Image Filtering Architecture

The architecture implemented for image filtering is based on the AXI4-Stream protocol, which is a simple handshake interface optimized for applications following a dataflow paradigm. It includes a set of reconfigurable filters, each one featured with two AXI streaming interfaces (one input and one output). Both streaming interfaces are connected to an AXI switch, which makes it possible to set any connection between the filters at runtime, without modifying the static system.

A Video Direct Memory Access (VDMA) controller has also been integrated to gather input images from the external DDR memory as well as to write back the output processed images. The system is controlled by the Zynq PS (that contains

Figure 4 – Detail of the block-based architecture with three reconfigurable filters, the AXI switch and the VDMA controller

a dual core ARM processor), integrated into the *Zynq-7020* FPGA used for the implementation.

Since IMPRESS allows decoupling the implementation of the static and the reconfigurable modules, a single baseline static architecture has been designed. It contains a customizable number of reconfigurable partitions (each one may contain a RM) as shown in Figure 4 (for the case of an architecture with three reconfigurable filters). The customizable black-box IP has been used as a placeholder in the static design, instead of the final filters. The *black box* is configured with two AXI Stream interfaces, so it is compatible with all the filters the user may finally reconfigure in the final system. The creation of the baseline system has been automated by means of a set of TCL scripts, which makes it easily repeatable for a different number of filters just by receiving the desired value as a parameter. To increase the ease of use, and taking into account that standardized interfaces are used, the generation of the VI and the VA files has also been fully automated, up to the generation of the full and partial bitstreams. In the case of the VA, a simple slot-based structure has been used, using Reconfigurable Regions occupying the whole height of the clock regions in the device, as shown in Figure 5.

Each of the filters to be integrated at run-time in the architecture has been implemented as an independent RM, using IMPRESS. In particular, four different image processing filters (*pass through*, *dilate*, *erode* and *Sobel*) have been designed using the *OpenCV* library in Vivado HLS, also proving the benefits of the proposed tool to create reconfigurable systems without having an advanced background in HW design. All the four filters can be reconfigured in any of the three RPs of the device by reallocating the same partial bitstream.

B. Performance Analysis of IMPRESS

The performance of IMPRESS tool will be compared in this section with the commercial Xilinx partial reconfiguration flow [13]. With this aim, the image filtering architecture described in the previous section has been implemented with both tools.

Figure 5 -Implementation of the Reconfigurable Streaming Processing Architecture, including the static system and two of the reconfigurable modules (dilate and Sobel filters). Implemented on a Zynq-7020 FPGA

Figure 6 shows the netlist including three dilate filters (one on each RP) resulting from the implementation with a) the Xilinx PR flow and b) the IMPRESS tool. The IMPRESS design time composition feature has been used to perform the analysis. As can be seen in the image, design b) has consistent interfaces in all the RPs and so the same partial bitstream can be relocated in the three RPs. This is not the case of a), where a different interface is generated in each of the RPs.

Table II presents a comparison of the maximum operation frequency with different filters allocated in each RP. The maximum frequency obtained with the proposed tools is slightly below the values achieved when the Xilinx PR flow is used. This is due to the constraints introduced to allow module relocation and the partition pin placement.

The implementation time of the proposed tool is also compared to the Xilinx PRF. The Xilinx PRF needs 131 seconds to implement the static system and 100 seconds (on average) to implement a RM on all the three RPs at the same time. When using the proposed tools, the static system implementation time is incremented to 229 seconds and the RM implementation to 75 seconds on average. It is necessary to highlight that IMPRESS is a script-based tool that runs on top of Xilinx Vivado and thus it suffers an overhead in terms of implementation time. However, as relocation reduces the number of PBS that are to be generated IMPRESS is more efficient generating RMs (75s against 100s). All the measurements have been done using an i7-7700 @3.6GHz processor with 16GB of DDR4 RAM. Pre-synthesized design checkpoints have been used as inputs for the static system and all RM. Therefore, the synthesis time is not included in the given values.

Table III presents a detailed analysis of the time required by each stage performed with the tool. As can be noted, the most time-consuming task is routing. The reason is that this step

Table II - Comparison of maximum frequency with the two design flows

	IMPRESS	Xilinx PRF	% decrease
Dilate	97.8MHz	102.9MHz	4.96%
Erode	94.4MHz	100.2MHz	5.79%
Pass through	93.6MHz	102.1MHz	8.33%
Sobel	96.9MHz	103.2MHz	6.10%

Figure 6 design implementation with dilate filters in every reconfigurable partition using a) Xilinx reconfiguration flow b) IMPRESS using reconstruction at design time.

Table III - Implementation time details

	Static system	Reconfigurable module (in average)
Input file processing	22s	-
Netlist manipulation	55s	10s
Apply VA and VI	15s	5s
Place design	18s	6s
Route design	106s	37s
Bitstream and DCP generation	17s	10s
Generate design-reconstruction files	23s	7s
Total	256s	75s

involves finding the nodes that form the blocker macro, applying them and routing the design with those constraints.

The main advantage of using relocatable designs is the reduction in the memory footprint. It is necessary to store a single PBS for each RM, regardless the number of RPs where it must be reallocated. Moreover, PBS generated with the tool only include the minimum content required to reconfigure the RM: the header is included at run-time, BRAM contents are not included and in the case of sub-clock region RPs, only the used region is included in the PBS. The memory footprints for the use case design are presented in Table IV. As it can be seen, there is a substantial reduction in memory footprint, 68% for one PBS (which can be more considerable in sub-clock PRs) and 89.3% for all the PBS needed in the use case design (12 PBS without relocation and 4 PBS with relocation).

Table IV - Partial bitstream memory footprint

	Xilinx PRF	Proposed tools	Percentage decrease
1PBS	837KB	268KB	68%
All PBS needed for the 4 RMs in the 3PRs	10044KB	1072KB	89.3%

Figure 7 stacked sub-clock region filters including the static system and two of the reconfigurable modules (dilate and pass through filters)

C. Sub-clock region designs

The previous design uses the whole height of the clock region areas and each filter is connected to each other through an AXI switch that is part of the static system. Some of the most powerful IMPRESS features are the ability to stack vertical RPs in the same clock region and to have direct reconfigurable-to-reconfigurable interconnections. To valid both features, the previous design has been modified connecting two filters in cascade and stacking one filter on top of the other as shown in Figure 7.

D. Performance Analysis of the Reconfiguration Engine

The size of the PBS has also impact on the reconfiguration time. As explained in section VI the first step carried out by the reconfiguration engine is to read the previous configuration of the RP and to combine it with the PBS. In the case of one of the RPs in the proposed image filter architecture, it takes 8529 μ S to complete this step. Then a header is created and sent to the PCAP. This is done in 5 μ S. The combined PBS is also sent through the PCAP in 2334 μ S. Finally, a tail is sent requiring 4 μ S. Therefore, the total time required to change one of the filters in the design is 10.872 ms. This time will be significantly reduced in future work.

VIII. CONCLUSIONS AND FUTURE WORK

Partial reconfiguration is still considered by main players in the market as an advanced design flow, which requires an in-depth knowledge of the low-level details of the device, reducing its applicability in the industry domain. The proposed tools make easier the use of reconfiguration and enable its use to non-HW users as explained in the image processing use case. Moreover, IMPRESS removes much of the constraints imposed by commercial tools by allowing the relocation of RMs, the stacking of multiple RPs in a clock region, the hierarchical reconfiguration, enabling reconfigurable to reconfigurable communications and decoupling the implementation of the static and the reconfigurable systems.

Regarding the future work, the tool will be provided as an open-source solution to make it available to be used by the community. It will also be ported to the Zynq Ultrascale+ family. In order to improve the reconfiguration engine performance, it will be reimplemented as a pure hardware

component with a special focus on the acceleration of the sub-clock region reconfiguration process.

ACKNOWLEDGMENT

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732105.

REFERENCES

- [1] T. Drahonovský, M. Rozkovec and O. Novák, "Relocation of reconfigurable modules on Xilinx FPGA," *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, Karlovy Vary, 2013, pp. 175-180.
- [2] 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," *2012 International Conference on Reconfigurable Computing and FPGAs*, Cancun, 2012, pp. 1-6.
- [3] A. A. Sohaghpurwala, P. Athanas, T. Frangieh and A. Wood, "OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs," *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, Shanghai, 2011, pp. 228-235.
- [4] D. Koch, C. Beckhoff and J. Teich, "ReCoBus-Builder — A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs," *2008 International Conference on Field Programmable Logic and Applications*, Heidelberg, 2008, pp. 119-124.
- [5] A. Otero, E. de la Torre and T. Riesgo, "Dreams: A tool for the design of dynamically reconfigurable embedded and modular systems," *2012 International Conference on Reconfigurable Computing and FPGAs*, Cancun, 2012, pp. 1-8.
- [6] C. Beckhoff, D. Koch and J. Torresen, "Go Ahead: A Partial Reconfiguration Framework," *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, Toronto, ON, 2012, pp. 37-44.
- [7] A. Lalevé, P. H. Horrein, M. Arzel, M. Hübner and S. Vatou, "AutoReloc: Automated Design Flow for Bitstream Relocation on Xilinx FPGAs," *2016 Euromicro Conference on Digital System Design (DSD)*, Limassol, 2016, pp. 14-21.
- [8] R. Oomen, T. Nguyen, A. Kumar and H. Corporaal, "An automated technique to generate relocatable partial bitstreams for Xilinx FPGAs," *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, London, 2015, pp. 1-4.
- [9] J. Rettkowski, K. Friesen and D. Göhringer, "RePaBit: Automated generation of relocatable partial bitstreams for Xilinx Zynq FPGAs," *2016 International Conference on ReConfigurable Computing and FPGAs (ReConFig)*, Cancun, 2016, pp. 1-8.
- [10] C. Beckhoff, D. Koch and J. Torresen, "The Xilinx Design Language (XDL): Tutorial and use cases," In the 6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), Montpellier, 2011, pp. 1-8.
- [11] D. Koch *et al.*, "Partial reconfiguration on FPGAs in practice — Tools and applications," *ARCS 2012*, Munich, Germany, 2012, pp. 1-12.
- [12] A. Otero, E. de la Torre, T. Riesgo and Y. E. Krasteva, "Run-Time Scalable Systolic Coprocessors for Flexible Multimedia SoPCs," *2010 International Conference on Field Programmable Logic and Applications*, Milano, 2010, pp. 70-76.
- [13] Xilinx, "Vivado Design Suite User Guide - Partial Reconfiguration" UG909 (v2017.3) October 4, 2017
- [14] Xilinx, "Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools)" XAPP1222 (v1.3), September 23, 2016.
- [15] F. Ferrandi, M. Novati, M. Morandi, M. D. Santambrogio and D. Sciuto, "Dynamic Reconfiguration: Core Relocation via Partial Bitstreams Filtering with Minimal Overhead," *2006 International Symposium on System-on-Chip*, Tampere, 2006, pp. 1-4.
- [16] Heiko Kalte and Mario Pörmann. 2006. REPLICA2Pro: task relocation by bitstream manipulation in virtex-II/Pro FPGAs. In *Proceedings of the 3rd conference on Computing frontiers (CF '06)*. ACM, New York, NY, USA, 403-412.