2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines

Designing Partial Bitstreams for Multiple Xilinx FPGA Partitions

Victor M. Gonçalves Martins*†, João Gabriel Reis*, Horácio C. C. Neto†, Eduardo Augusto Bezerra*
*Department of Electrical and Electronic Engineering
Federal University of Santa Catarina
88040-900 Florianópolis, Santa Catarina - Brazil
{victor.martins, joao.reis, eduardo.bezerra}@eel.ufsc.br
†Electronic System Design and Automation - INESC-ID
Rua Alves Redol, 9; 1000-029 Lisboa, Portugal
{marte, hcn}@inesc-id.pt

***Abstract*—Although FPGA logic resources groups present some homogeneity, available *Dynamic Partial Reconfiguration* (DPR) tools consider them entirely heterogeneous. This assumption implies that each partial bitstream has to be allocated to one *Reconfigurable Partition* (RP). To solve this limitation, we present in this paper a new *Assisted Design Flow* (ADF) that implements a very efficient *Partial Bitstream for Multiple Partitions* (PB4MP) technique. Based on the Xilinx tools flow, the proposed ADF: takes advantage of the Xilinx tools constraints; does not require any intermediate manual steps; gives more freedom to the tools than similar approaches; and only repeats the regular flow once. Having the ability to disable all interface signals from each RP, this method allows to relocate modules safely in different RPs with the same bitstream or by an internal configuration memory swap, which increases system dependability and significantly facilitates the hardware tasks management.**

***Keywords*-Bistream for Multiple Partitions; FPGA Partial Reconfiguration**

I. Introduction

In some of the current commercial SRAM-based *Field Programmable Gate Array*s (FPGAs), *Partial Reconfiguration* (PR) can be performed with no need to stop the rest of the system and may, in fact, be done by the system itself.

FPGAs are often used for parallel processing as they can be adapted to the processing requirements of the application. The designer can normally create any number of specific tasks, provided their hardware requirements do not exceed the resources available in the FPGA. PR allows changing system components without interrupting the operation of the remaining components, which means that, using the PR, only the number of tasks being executed at the same time is limited by the amount of resources. The total number of tasks may exceed that amount. PR, therefore, provides extensibility of resources on a device.

However, the Xilinx flow requires different partial bitstreams for the same component, if it is to be implemented in different *Reconfigurable Partition*s (RPs) (even if they have the same structure). This means that N x M partial bitstreams must be generated to implement M components in N RPs. This restriction may significantly increase the amount of memory required to accommodate all partial bitstreams.

This paper presents a new *Assisted Design Flow* (ADF) that implements an efficient partial bitstream management mechanism. The ADF is based on the PR flow provided by the Xilinx tools [1], includes rules that are part of *Isolation Design Flow* (IDF) [2], and takes advantage of certain constraints [3]. The proposed flow was validated by implementing an embedded system with a softcore and several RPs [4], in a Xilinx Virtex-6 FPGA (XC6VLX240T).

The paper is organized as follows. Section II briefly discusses the state-of-the-art and related work. Section III presents the proposed architecture, the flow process associated, and the processing and memory requirements. Section IV summarizes our case studies and presents the results. Section V concludes this paper and presents some further work suggestions.

II. Related Work

PR tools, such as PlanAhead from Xilinx [1], require a partial bitstream to be generated for each module in each RP. In order to solve this limitation, some works with related topics have been published. The work in [5] summarizes a list of operations that allow to generate partial bitstreams that can be used in more than one RP. The paper does not show how to perform the routing of the remaining system that crosses the RPs, and does not present solutions to the routing of signals that connect to the RPs inputs, or to clock signals. The same authors presented the work in [6], where they indicate that the routing problem of the signals that are crossing the RPs can be resolved manually using the Xilinx FPGA Editor.

Another interesting approach is the GoAhead [7] tool. Instead of using the constraints to force a particular Placement and Routing, they propose a new tool to manipulate the *Xilinx Design Language* (XDL) obtained at the end of the Mapping, so that the Placement and Routing tool has only one option available for a given resource location or a routing.

978-1-4799-9969-9/15 $31.00 © 2015 IEEE
DOI 10.1109/FCCM.2015.10

The authors in [8] propose another process to relocate partial bitstreams in severals RPs. It is based on the work in [6] and eliminates the problem of routing signals that cross the RPs.

An important difficulty associated to the relocation of partial bitstreams in various RPs is the routing of the clock signals. The work in [9] presents the *Local Clock Domain*s (LCDs) as a solution for this problem.

The main contribution of the present work is to propose a new complete flow, with no intermediate manual step, and which only requires to repeat the flow Translation → Mapping → Placement and Routing once (in proposals [5] [6] [8] it is repeated twice).

III. Proposed Method Based on PB4MP

In this paper, a method to increase system dependability and that significantly simplifies the hardware tasks management is proposed. As it is developed using the PB4MP tool, these advantages are obtained with small FPGA resource overhead. However, this method uses the capabilities of PR [1] and IDF [2] to perform the desired implementation.

A. *The Proposed* Assisted Design Flow *(ADF)*

Figure 1 shows the full design flow of a PB4MP based FPGA system. It is based on the standard Xilinx tools flow (ISE and PlanAhead) and only requires user intervention at the initial phase of the hardware design, as in the IDF process [10].

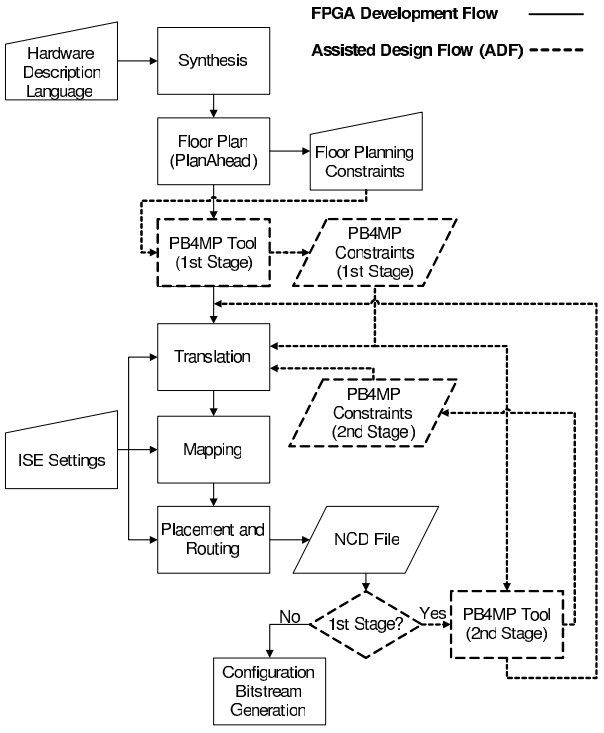

Figure 1. FPGA Development + *Assisted Design Flow* (ADF)

The base of the flow is the following sequence: Synthesis → Floor Plan → Translation → Mapping → Placement and Routing → Configuration Bitstream Generation. It is important that the "Hardware Description Language" input is structured to correctly assign the desired module to the RPs. This includes adding `BUFHCE` primitives, and deciding between adding the standard `SCC_BUFFER`, or a LUT2 primitive which allows disabling all interface signals. If the second option is chosen, each physical LUT6 can implement two LUT2 as presented in [8], helping reducing the FPGA resources usage. When deciding the RPs location, another step that needs the designer attention is the Floor Plan [1].

After the floor planning, the PB4MP tool generates the first constraints group. These constraints will be used by the buffers inclusion mechanism (`SCC_BUFFER` or LUT2). This time the tool does not block the resources for these buffers, and it only defines a window of possible locations for them. This is a way to give some freedom for the next steps to come up with reasonable placement and routing results.

At the end of the first Placement and Routing interaction, an *Native Circuit Description* (NCD) file is generated. The PB4MP tool is ran again, this time using the Directed Routing Constraints tool from `fpga_edline.exe`, in order to extract the nets that interconnect the modules in RPs to the system.

At this point, there is a `ROUTE` constraints group for each RP interface, which includes the correspondent resources location. The tool chooses one (automatically or defined by the designer), and with this information, the second constraints group is created. The sequence, Translation → Mapping → Placement and Routing is executed again. At the end of these two stages the final bitstream is generated and can be downloaded to the FPGA.

B. *PB4MP Adaptation for Increased Reliability*

To take advantage of PB4MP in order to increase reliability, an extra constraints list was added to the ADF process. This list is built using the `CONFIG PROHIBIT` constraint, which gives the option to select the undesirable FPGA resources. Considering a system with three RPs, a possible approach could be, at each three columns, to remove a whole esource column. In order to ensure the required reliability, a column in the same relative position to an already excluded column located in another RP, can never be removed.

With this distribution, even if a module presents a permanent fault, it is possible to replace the RP by another module that does not use the same resource where the permanent fault was detected. This can be a solution for FPGA systems suffering from aging effects. A hardware task management component can be used to periodically swap RPs in order to distribute time out periods among resources. The `CONFIG PROHIBIT` constraint excludes the resources (*Configurable Logic Block* (CLB), *Digital Signal Processing*

(DSP), BRAMs, etc) but it does not remove the associated switch boxes. However, if the resource is not used, there will be no routing to that resource. This means that the switch box is not used, or it is used in a different way.

IV. Experimental Results and Discussion

In order to exercise this new approach, the System-on-Chip plataform [4] shown in Figure 2 has been implemented. Our platform supports dynamic reconfiguration and was implemented in a Xilinx XC6VLX240T FPGA.

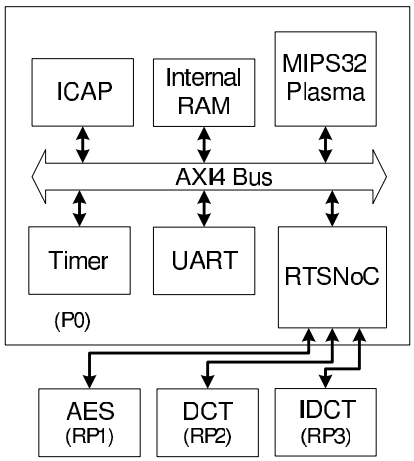

Figure 2. System Diagram

It is based on an implementation of the MIPS32 ISA, the Plasma softcore, which is freely available at Opencores [11]. The internal structure of a *Central Processor Unit* (CPU) node is based on the AXI4 family of protocols, which is becoming the industry's standard for bus-based interconnection. Hardware reconfiguration is held by the *Internal Configuration Access Port* (ICAP) interface on the AXI4.

The Plasma CPU runs the *Embedded Parallel Operating System* (EPOS) [12], which provides the necessary run-time support to implement system features, and the communication with other modules.

RP1, RP2 and RP3 are the RPs holding, in this case, the *Advanced Encryption Standard* (AES), the *Discrete Cosine Transform* (DCT) and the *Inverse Discrete Cosine Transform* (IDCT) components, respectively.

A *Real-Time Star Network-on-Chip* (RTSNoC) [13] based interconnection scheme has been chosen for the reconfigurable partitions. A bus-based interconnect was discarded as it is not the most suitable choice in heterogeneous designs where hardware components have active roles [14]. The RTSNoC consists of routers in a star topology that can be arranged forming a 2-D mesh. Each router has eight bidirectional channels that can be connected to cores or to other routers channels. Each component is connected to a port of an RTSNoC router and one of the RTSNoC routers ports is connected to an AXI4 bridge.

The largest module uses 4,172 *Look Up Table* (LUT) Flip-Flop pairs (synthesis estimate). As in the Virtex-6 family each slice has four LUT Flip-Flop pairs, one CLB has two slices, and a section of a CLB column has 40 CLBs, each of these sections has 240 LUT Flip-Flop pairs. This means that about 18 of this CLBs columns are needed. As the approach requires 33% of additional resources, so the resources windows are defined with 24 CLB columns. Unfortunately, during the Placement and Routing, without any resources distribution policy, the tool failed due to routing congestion. After some interactions, we found that 27 were the minimum window size in order to complete the full Placement and Routing operation. The application of the resources distribution in the second stage brought no any extra routing congestion.

All sections of resources for each RP need 1744 frames to be configured. As in the Virtex-6 family the frame size is 324 words of 32 bits, this means that 2,260,224 bytes are needed to configure each RP. As the partial bitstreams can be allocated in any RP, a total memory size of 6,780,672 bytes will be required. Without using the PB4MP mechanism, and if we wanted to allocate all components in any RP, the required memory would had been 20,342,016 bytes (3X).

In order to evaluate the Buffers Window strategy, the critical path of the various modules were analyzed. This review was done comparing to the "bus macro" option, used in a recent work [8]. In that work the authors place Proxy Logic at the border of the RP and the new (added) LUT1 is placed in the static region. These two LUTs are forced to be placed as close as possible in order to control the short section of the wires. We found that in our implementation there is an 8% to 11% decrease in the propagation time on the critical path. This reduction was due to the fact that the tools may initially place the two LUTs following its own algorithm, with only a limitation to the area defined by the Window Buffer. This gain also depends on the complexity of each module, the percentage of the used FPGA resources, or if the I/Os of the modules are registered. However, regardless of the greater or lesser influence, the use of Window Buffer always improves the critical path.

Considering a system with three RPs and three modules, a basic management is to implement a periodic swap between the RPs in a circular way, as shown in the pseudocode next:

```
void main_application(){
  ...
  init_swap_counter();
  while (1){
    ...
    if (read_swap_counter() >= SWAP_COUNTER_LIMIT) {
      RP_swap(&RP1, &RP2);
      RP_swap(&RP2, &RP3);
      reset_swap_counter();
    }
    ...
  }
}
```

A counter is initialized in the beginning and, while the application is running, the system checks periodically the

counter value. If the counter reaches a specified limit, two swap operations are executed, implementing the sequence RP1 $\rightarrow$ RP2 $\rightarrow$ RP3 $\rightarrow$ RP1... Before performing the swaps, all interface signals are disabled on both RPs. This is a simple example. In an actual system there are more modules than RPs, and the partial bitstream generation is performed by distributing the modules among the RPs. With this distribution and a correct hardware task management, it will be possible to control FPGA resources usage. At the same time, the hardware tasks can be selected and allocated to implement the desired application.

Another advantage of this method is the possibility of having permanent fault recovery capability. For that the system must present a fault detection mechanism, as the approach discussed in [15]. Even if an RP has a permanent fault, all modules can continue to be allocated and all RPs can still be available to allocate modules. Only some modules cannot be allocated in this RP.

V. Conclusions and Future Work

The main objective of this work is to propose a new automatic process, based on the Xilinx ISE flow, to generate partial bitstreams which can be allocated in multiple RPs (PB4MP). The proposed flow has the following features and advantages: automatic execution; it significantly simplifies the hardware task management; it provides more freedom for the placement tools for resource allocation; and it provides a "disable" interface.

PB4MP can be used also in other FPGA systems where it is important to expand the life cycle. Implementing the FPGA system with a proper resource management makes it possible the recovery from a permanent fault without sacrificing the RP. As a future work, the authors are working in the migration of the ADF to the Vivado flow, adjusting the flow to make the full development (software and hardware tasks) more user friend. The authors are also starting to investigate how the PB4MP can be used to enhance the FPGA life cycle by reducing the stress effects caused by *Negative Bias Temperature Instability* (NBTI).

Acknowledgements

This work has been funded by the National Counsel of Technological and Scientific Development (CNPq).

References

[1] Xilinx Inc. Partial reconfiguration tutorial: Planahead design tool v14.5, April 2013. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/PlanAhead_Tutorial_Partial_Reconfiguration.pdf.

[2] Xilinx Inc. Isolation design flow. http://www.xilinx.com/applications/isolation-design-flow.html, 2014.

[3] Xilinx Inc. Constraints guide v14.5 (ug625). http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/cgd.pdf, April 2013.

[4] Tiago Rogério Mück and Antônio Augusto Fröhlich. Seamless integration of hw/sw components in a hls-based soc design environment. In *Proc. International Symposium on Rapid System Prototyping (RSP)*, pages 109–115, Montreal, Canada, October 2013.

[5] Yoshihiro Ichinomiya, Sadaki Usagawa, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. Designing flexible reconfigurable regions to relocate partial bitstreams. In *Proc. IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, page 241, Toronto, Canada, May 2012.

[6] Yoshihiro Ichinomiya, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. A bitstream relocation technique to improve flexibility of partial reconfiguration. In *Proc. IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, pages 139–152, Fukuoka, Japan, September 2012.

[7] Christian Beckhoff, Dirk Koch, and Jim Torresen. Goahead: A partial reconfiguration framework. In *Proc. IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 37–44, Toronto, Canada, May 2012.

[8] Tomás Drahonovský, Martin Rozkovec, and Ondrej Novák. Relocation of reconfigurable modules on xilinx fpga. In *Proc. IEEE Annual International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 175–180, Karlovy Vary, Czech Republic, April 2013.

[9] Adam Flynn, Ann Gordon-Ross, and Alan D. George. Bitstream relocation with local clock domains for partially reconfigurable fpgas. In *Proc. IEEE Annual International Conference on Design, Automation and Test in Europe (DATE)*, pages 300–303, Nice, France, April 2009.

[10] Xilinx Inc. Single chip crypto lab using pr/iso flow with the virtex-5 family for ise design suite 12.1 v1.1.2 (xapp1105). http://www.xilinx.com/support/documentation/application_notes/xapp1105_V5SCC_PRISO.pdf, June 2013.

[11] OpenCores. http://opencores.org, November 2014.

[12] The EPOS Project. Embedded parallel operating system. http://epos.lisha.ufsc.br, 2014.

[13] Marcelo Daniel Berejuck. Dynamic Reconfiguration Support for FPGA-based Real-time Systems. Technical report, Federal University of Santa Catarina, Florianópolis, Brazil, 2011. PhD qualifying report.

[14] G. De Micheli, C. Seiculescu, S. Murali, L. Benini, F. Angiolini, and A. Pullini. Networks on chips: from research to products. In *Proc. 47th Design Automation Conference (DAC)*, pages 300–305, Anaheim, USA, June 2010.

[15] Victor M. G. Martins, Frederico Ferlini, Djones V. Lettnin, and Eduardo A. Bezerra. Low cost fault detector guided by permanent faults at the end of fpgas life cycle. In *Proc. IEEE Latin America Test Workshop (LATW)*, pages 1–6, Fortaleza, Brasil, March 2014.