Reconfigurable Systems with Xilinx FPGAs

Victor Orfeu Merlo
Undergraduate in Electronic Engineering
University of Applied Sciences - HSHL
Lippstadt, Germany
Email: victor.orfeu-merlo@stud.hshl.de

***Abstract*—Reconfiguration is the capability of an FPGA to easily change it's internal logic and functions. This reconfiguration can be complete or partial, when only part of the device is changed, and the latter can be done statically or dinamically, where the device still operates while reprograming. This feature is not supported by all Xilinx's FPGAs and it cannot be done in all of it's elements. The advantages of this feature come with increased design complexity, that Xilinx tries to mitigate using an adequately developed project flow. This paper provides a brief summary of some of Xilinx's materials on Partial Reconfiguration.**

***Index Terms*—FPGA, Partial Reconfiguration, Xilinx, Dynamic Function eXchange**

I. Introduction

To put it simply, reconfiguration is the capability of an FPGA to change it's internal logic, functions and behavior. It is a really powerfull feature, as it allows hardware designers to change their device without needing to manufacture or fabricate it all over again.

Reconfiguration can be classified as Complete or Partial. Complete Reconfiguration means the whole device is reconfigured at once. Partial Reconfiguration (refered from here on as PR) means only part of the device is reconfigured, while other parts stay the same. The latter takes one step further the already incredible flexibility of a FPGA.

Figure 1 shows the basic premise of PR. The FPGA has a full configuration, with a reconfigurable region defined inside it. This region can be reconfigured with diferent bitstreams.

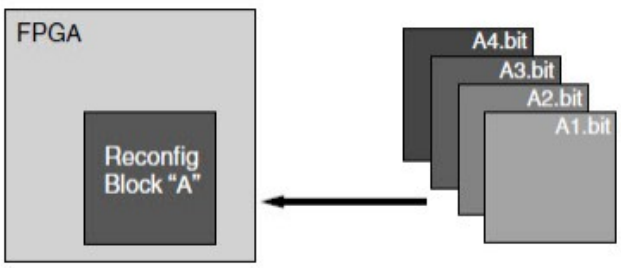

Fig. 1. Basic idea of Partial Reconfiguration. Taken from Xilinx's User Guide 909.

PR can be further classified in Static and Dynamic. Static Reconfiguration is when the device stops operating while it is reconfigured. Dynamic Reconfiguration is when the device continues operating while some parts are reconfigured.

Some reasons to implement PR in a project include [8]:

- Reducing the size of the FPGA required to implement a set of functions, with consequent reductions in cost and power consumption;
- Providing flexibility in the choices of algorithms or protocols available to an application;
- Enabling new techniques in design security;
- Improving FPGA fault tolerance;
- Accelerating configurable computing;

In 2020, Xilinx also "rebranded" Partial Reconfiguration in their devices and software to "Dynamic Function eXchange (DFX)", but the concepts and manuals are still the same [9].

II. Supported devices

A. 7 Series devices

Nearly all Virtex-7, Kintex-7, Artix-7, and Zynq-7000 All Programmable SoC devices are supported for PR. Spartan-7 devices, Artix-7 7A12T, 7A25T and Zynq-7000 devices with a single-core processor (Z-7007S, Z-7012S, Z-7014S) are not supported [8].

B. UltraScale devices

Place and route and bitstream generation is enabled for all production devices except the VU440. Bitstream generation is disabled by default for ES2 devices, but place and route can still be performed [8].

C. UltraScale+ devices

Place and route and bitstream generation is enabled for all production devices including all Zynq UltraScale+ RFSoC devices. Place and route is enabled for the Virtex UltraScale+ VU37P, but bitstreams are gated by a parameter as this device is only available as ES1. Place and route is enabled for many engineering silicon (ES1, ES2) versions of UltraScale+ devices. Bitstream generation is disabled by default for these devices [8].

III. Styles of Partial Reconfiguration

A. Difference-based or Small Bit Manipulations

When small minute changes were needed in the design, the Virtex-II device supported this feature. In the Xlinx's HDL software called *FPGA_editor*, changes could be made to specific elements in the device, for example LUTs, RAM contents, flip-flop initialization values. While it was possible to change routing information, that was not recommended because it could cause internal contention of signals and

resources. Then, the software would generate a new bitstream that changed only the differences between the old and new design [4]. Yet, it seems that this feature is not supported in current Xilinx devices, as there are no more manuals or tutorials about the subject.

B. Module-based

In this style, the FPGA is divided in two kinds of regions during the design: static region and reconfigurable modules (RMs). The static region will not change during runtime and will remain operational. The RMs are regions that can be reprogrammed while the device is running and this should not affect the static region or other modules [4].

This style shapes the hardware design from the start, because you have to define module size, conections, floorplanning and so on. It is useful when large blocks of logic are meant to be changed in one go. A new bitstream has to be generated for each design of a RM [4], [6].

IV. Module-based project flow

As this is the most common style of PR, a kind of standard design flow was developed by Xilinx. It shares some similarities with a non-PR project design flow. The following subsections are directly transcripted from Xilinx User Guide 909 [9].

A. Enabling PR

First, just like a normal project, there needs to be sources in the project, like VHDL code or contraint files. A top level source combines every other source under it. There can also be IPs as sources, but not in the top level.

Then, PR must be enabled in the project so Vivado can prepare for the design flow. This cannot be undone, so Xilinx recommends making a backup of the project before this step.

B. Defining Reconfigurable Partitions (RPs) and Reconfigurable Modules (RMs)

The next step is to define RPs where the RMs will reside. Each source can be turned into one RP and it has to contain at least one RM. In Vivado, they are shown separately in the hierarchy as yellow diamonds with their RMs, as shown in Fig. 2. If there is yet no design source for a RP, a "black-box" module is created to separate the region for a future source.

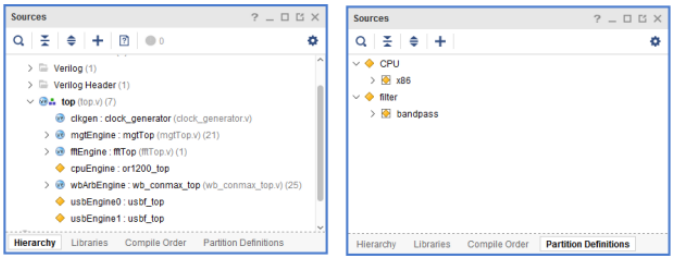

Fig. 2. Hierarchy shown in the Sources windows in Vivado. Taken from Xilinx's User Guide 909.

After the RMs are defined, they need to be positioned in the FPGA, this step is called *floorplanning*. A region called *Pblock* defines what resources are avaliable for a particular RM. Fig. 3 shows an example of Pblocks defined in a 7-series device.

There are a number of features, contraints and requirements for floorplanning, each of them covered in manuals for each device family. As an example, RMs in the 7-series have an option to *Reset After Reconfiguration* that is only avaliable if the Pblock vertical boundary coincides with a clock region boundary. This feature holds the RM in reset until the reconfiguration is done and ensures the module starts operating from a known state.

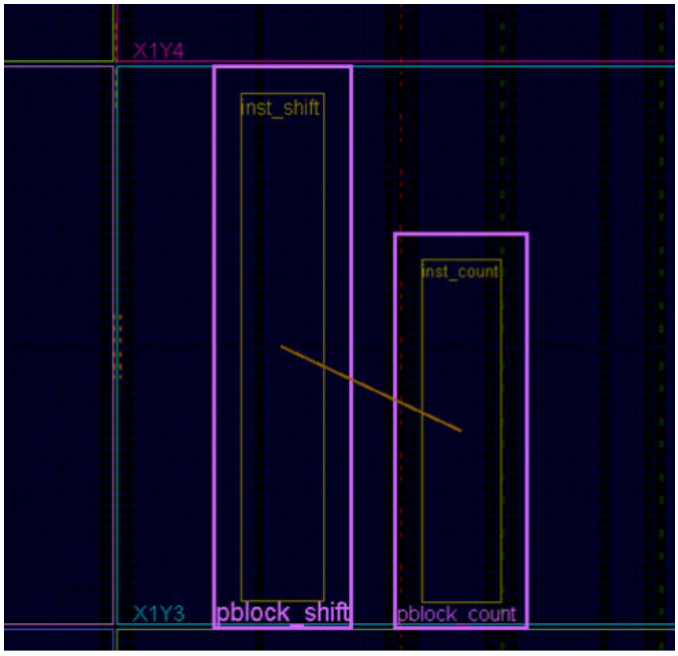

Fig. 3. Pblocks definde in a 7-series device. The left supports Reset After Reconfiguration while the right doesn't. Taken from Xilinx's User Guide 909.

C. Configurations

With the modules defined, the reconfigurable regions need to be combined with the static regions in *Configurations*. Each Configuration consist of all the static logic plus one RM per RP, and it's a full design image. They take into account the hierarchy of RPs and RMs set by the designer to connect each region.

Together, they are the combinations of RM circuits wanted by the designer and each Configuration will be synthesised as a bitstream. Contraints can be added on each RM to be taken into account by the Configurations. Vivado has an option to automatically create them, it creates Configurations such that all RMs are included somewhere at least once.

D. Synthesis and Implementation

After defining the Configurations wanted by the designer, their implementation can be done, along with *place and route* of each design.

Because the design has a static region that doesn't change, this region is implemented first and it's implementation is

reused for all subsequent Configurations of the same project design. With this, Vivado is able to synthesise and implement different Configurations of RMs in parallel.

After every Implementation, Vivado automatically runs verifications and checks if the designs obey standard and user defined contraints, ensuring it can be used to write bitstreams.

E. Generating bitstreams

A key difference in a PR project is it contains different types of bitstreams, described bellow:

1) Full Configuration Bitstreams: These are the same bitstreams used in non-PR projects, they reconfigure the whole device while it's not operational. They are used to configure the static region and the first Configuration desired.

If the designers wants no functionality in the RMs initially, the RMs can be configured as black boxes, leaving the region avaliable to be partially reconfigured later. This allows for bitstream compression, as the region is essentially empty.

2) Partial Bitstreams: These bitstreams are delivered during device operation and configure the desired region. They are totally self-contained, meaning they contain the adresses and options of configuration. The user just needs to upload them to the device.

Partial Bitstreams' sizes are proportional to the size of the region they are reconfiguring. For example, if the RP is composed of 20% of the device resources, the partial bitstream is roughly 20% the size of the full design bitstream.

3) Blanking Bitstreams: These are a specific type of Partial Bitstreams that represent a logical black box. They remove the functionality of a RM by replacing it with a RM that only ties off the connections with the rest of the design, ensuring no I/O is left floating and the region is ready to receive a new configuration later.

4) Clearing Bitstreams: Specific to Ultrascale devices (and not Ultrascale+), because of their architecture, immediately before reconfiguring a module, the region needs to be cleared and prepared for the delivery of the new configuration. They do not change the functionality but shut down clocks driving logic in the region.

Each configuration of each module has it's own Clearing Bitstream generated automatically by Vivado. These are not Partial Bitstream as they are less than 10% the size of a corresponding Partial Bitstream.

V. Configuration Modes and Ports

Since PR is done by downloading a partial bitstream to the FPGA, most of the steps for configuring the devices are the same [8]. The following configuration modes support PR:

- ICAP: A good choice for user configuration solutions. Requires the creation of an ICAP controller as well as logic to drive the ICAP interface.
- MCAP: (UltraScale and UltraScale+ devices only) Provides a dedicated connection to the ICAP from one specific PCIe block per device.
- PCAP: The primary configuration mechanism only for Zynq-7000 AP SoC and Zynq UltraScale+ designs.
- JTAG: A good interface for quick testing or debug. Can be driven with the Vivado Logic Analyzer.
- Slave SelectMAP or Slave Serial: A good choice to perform full configuration and Partial Reconfiguration over the same interface.

Configuration Mode	7 Series	Zynq	UltraScale	UltraScale+	Zynq UltraScale+ MPSoC
JTAG	Yes	Yes	Yes	Yes	Yes
ICAP	Yes	Yes	Yes	Yes	Yes
PCAP	N/A	Yes	N/A	N/A	Yes
MCAP	N/A	N/A	Yes	Yes	Yes
Slave Serial	Yes	N/A	Yes	Yes	N/A
Slave SelectMap	Yes	N/A	Yes	Yes	N/A
SPI (any width)*	No	N/A	No	Yes	N/A
BPI sync mode*	No	N/A	No	Yes	N/A
BPI async mode	Yes	N/A	Yes	Yes	N/A
Master modes	No	N/A	No	No	N/A

Fig. 4. Supported configuration ports for PR. Taken from Xilinx's User Guide 909.

One of the reasons to implement PR is to quickly change the functionality of the FPGA, so the configuration time of each mode becomes important. The time is dictated by the device and port capabilites, mainly bandwidth and clock rate [10]. Of course, the size of the bitstream also interferes, but that's not as predictable as it depends on the design. A brief comparison is made in Table I.

TABLE I
MAXIMUM BANDWIDTHS FOR SOME CONFIGURATION PORTS IN 7-SERIES AND ULTRASCALE DEVICES.

Port	Device	Max clock	Data width	Max bandwidth
ICAP	7-Series	100 MHz	32 bit	3.2 GB/s
	Ultrascale	200 MHz	32 bit	6.4 GB/s
MCAP	7-Series	N/A	N/A	N/A
	Ultrascale	200 MHz	32 bit	6.4 GB/s
SelectMAP	7-Series	100 MHz	32 bit	3.2 GB/s
	Ultrascale	125 MHz	32 bit	4.0 GB/s
JTAG	7-Series	66 MHz	1 bit	66 MB/s
	Ultrascale	50 MHz	1 bit	50 MB/s

The configuration is done by a controller, which can be an external device or a PL section in the Static Region of the FPGA. The external device can be a processor, for example, as in Zynq devices. Figure 5 shows how this can be organized. The bitstreams are stored outside the FPGA and are downloaded to the chip through the controller. Since the bistream are self-contained, the controller takes and delivers them to the appropriate address/region through the selected PR port [8].

A. Dynamic Reconfiguration Port (DRP)

Devices from the 7-Series, Ultrascale and Ultrascale+ families have a DRP that can be used to dynamically reconfigure logic in the static region. This port only exists in some logic elements and can change only some parameters of logic elements such as: Phase-Locked Loops (PLLs), Mixed-Mode

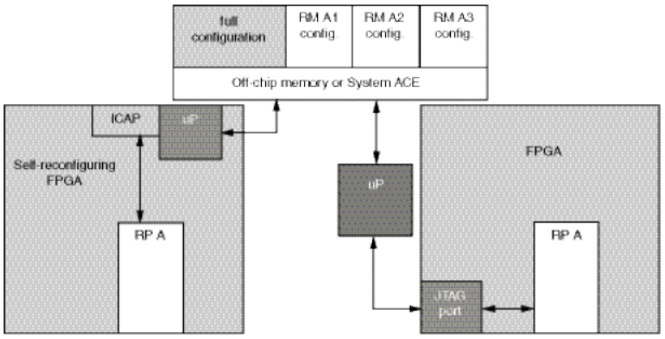

Fig. 5. Example of configuration using a microprocessor (μP) as the controller. Taken from Xilinx's User Guide 909.

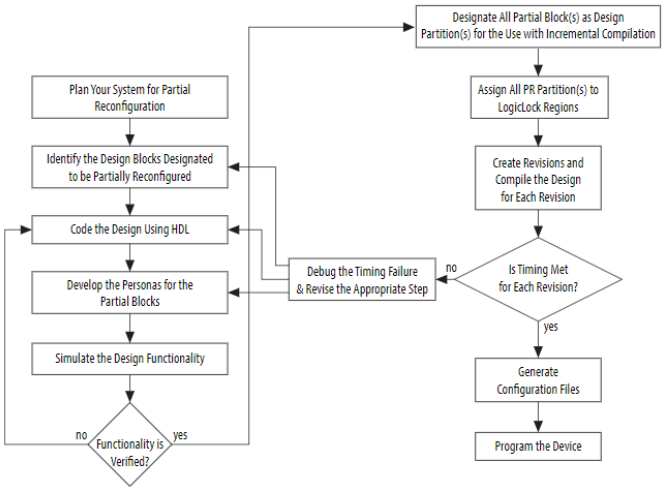

Fig. 6. Partial Reconfiguration Design Flow in Intel's Quartus Prime software and Altera devices. Taken from Intel's PR Tutorial webpages.

Clock Managers (MMCMs), and serial transceivers (MGTs). Information about the DRP and how to configure each element can be found in the many manuals provided by Xilinx [8].

VI. Xilinx's Intelectual Properties (IPs) for PR

Xilinx has created four IPs, freely avaliable in Vivado, to aid designers to quickly design PR projects. They are not required for the project to work [9]. Additionally, Xilinx adopted for its IPs the AXI protocol, a interface protocol defined by ARM [1], [7].

- DFX Controller: A controller that manages the partial bitstreams in the FPGA. It requires that all RMs are previously specified to it. The optional AXI4-Lite interface allows the controller to be dynamically reconfigured.
- DFX Decoupler: This coreprovides safe and managed boundaries between static logic and Reconfigurable Partitions during reconfiguration.
- DFX AXI Shutdown Manager: Similar to the Decoupler, this core provides safe reconfiguration through AXI interfaces.
- DFX Bitstream Monitor: Used to identify and track partial bitstream as they flow through the design. Very useful for debugging or blocking bitstream loads in case something goes wrong.

VII. Brief mention of Altera

Altera's FPGAs also support PR, mainly in the families of devices Arria and Stratix, using their Quartus Prime software [2]. A simple flowchart of their design flow is shown in Figure 6. Apart from naming conventions, some of the differences between this and Xilinx's flow are the requirement for a controller IP and the pBlocks (here called Logic Lock regions) already exist in non-PR projects. Other design steps, such as organizing hierarchy, floorplanning and timing validation, remain essentially the same [3].

VIII. High-Level Modeling

It may be clear that designing a dynamic reconfigurable system is no easy task, even in phases before simulations. The benefits of Partial Reconfiguration must overcome the setbacks of designing, implementing and managing resources. That is why the focus is shifting to actually developing high level tools to help model PR projects before time and resources are used on devices [11].

Many tools have been developed to aid in this area, to take requirements and high level hierarchy and turn them into HDL files or do analysis based on them. Features and device support vary between softwares. Their final goal is to be able to predict resource utilization, not only of FPGA elements but power and time, before spending time implementing any design [5].

References

[1] florentw. *AXI Basics 1 - Introduction to AXI*. Avaliable in: https://support.xilinx.com/s/article/1053914?language=en_US, 2021.

[2] Intel. Partial Reconfiguration. Avaliable in: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/partial-reconfiguration.html.

[3] Intel. Partial Reconfiguration Design Flow. Avaliable in: https://www.intel.com/content/www/us/en/docs/programmable/683499/18-1/partial-reconfiguration-design-flow.html.

[4] Davin Lim and Mike Peattie. *Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations*. Xilinx, 2002.

[5] Daniel M. Muñoz and Regina Marcela Ivo. RTRLib: A High-Level Modeling Tool for the Implementation of Dynamically Partial Reconfigurable System-on-Chips. In *International Conference on ReConFigurable Computing and FPGAs*, 2019.

[6] M. Angelin Ponrani, G. Manoj, and R. Rajesvari. Module based partial reconfiguration on bitstream relocation filter. *International Journal of Computer Applications*, 66:P. 23–28, 2013.

[7] Xilinx. *Vivado Design Suite: AXI Reference Guide*, 2017. Web version. Avaliable in: https://docs.xilinx.com/v/u/en-US/ug1037-vivado-axi-reference-guide.

[8] Xilinx. *Vivado Design Suite User Guide - Partial Reconfiguration*, 2018.

[9] Xilinx. *Vivado Design Suite User Guide - Dynamic Function eXchange*, 2022.

[10] Xilinx. *Vivado Design Suite User Guide - Partial Reconfiguration*, 2022. Web version. Avaliable in: https://docs.xilinx.com/r/en-US/ug909-vivado-partial-reconfiguration.

[11] Xuejie Zhanga and Kam W Ng. A review of high-level synthesis for dynamically reconfigurable fpgas. *Microprocessors and Microsystems*, 24:P. 199–211, 2000.