Enabling certification for dynamic partial reconfiguration using a minimal flow

B. Rousseau[1], Ph. Manet[1], D. Galerin[1], F. Dedeken[2], Y. Gabriel[2]
D. Merkenbreack[1], J.-D. Legat[1]

[1] Université catholique de Louvain. Laboratoire de microélectronique. Place du Levant, 3. 1348 Louvain-la-Neuve
[2] Thales Communications Belgium. Rue des Frères Taymans. 1480 Tubize

{rousseau, manet, legat}@dice.ucl.ac.be, {denis.galerin, diego.merkenbreack}@student.uclouvain.be
{fabienne.dedeken, yves.gabriel}@be.thalesgroup.com

Abstract

As the trend in reconfigurable electronics goes towards strong integration, FPGA devices are becoming more and more interesting. They are already used for safety-critical applications such as avionics [9]. Latest FPGA's also enable new techniques such as dynamic partial reconfiguration (DPR), allowing new possibilities in terms of performance and flexibility. Their use in safety-critical systems is considered as impossible nowadays since they must be strictly validated, and DPR brings many new issues. Indeed, the tools used for DPR must be certified, which is barely impossible for the current DPR tools provided by the vendors. We have developed a simple flow upon the usual static one for Xilinx FPGA's that does not require any support of the vendor tools for DPR. This lessens the complexity of tools certification, and make a step towards enabling the certification of DPR for safety-critical applications. Moreover, under strong hypotheses, and by using safe design principles, we show how the complexity of certifying DPR can be reduced.

1. Introduction

It is getting hard nowadays to avoid the use of reconfigurable devices such as FPGA's in embedded applications. Numerous designs use them, even in safety-critical systems such as avionics [9]. In the future, FPGA's are undoubtedly going to be even more present. Moreover, FPGA's with an enormous amount of gates and integrated features (CPUs, DSPs, etc.) are becoming available, which makes these kind of devices an interesting option for a System-on-Chip (SoC) solutions.

These new devices also enable new techniques such as dynamic partial reconfiguration, a technique that consists in changing the configuration of a part of a circuit while the rest of it pursue its task. By using DPR, it becomes possible to experiment new solutions for building efficient electronic systems [11] [12] [7].

But electronic systems used in professional electronics and more specifically in avionics have to be very strictly validated in order to guarantee a high level of security. Because of this, safety-critical systems using FPGA's are generally built using very safe design principles. This way, it is possible to keep control on their behaviour and to guarantee their correct operation at each level and at each step of the design. Unfortunately, the use of DPR causes additional problems regarding certification since it increases the global complexity of a design. Most problems can actually be avoided by using simple DPR mechanisms and applying conservative design principles similar to those already used in safety-critical systems based on FPGA.

However the tools used to create the different configurations also require certification. DPR tools certification is at present still considered as being very hard or impossible since they are proprietary tools. This paper is focused on this issue. A new DPR flow has been developed which implements the minimal steps needed to be able to perform DPR on Xilinx latest FPGA's. The flow is based on the parsing of the bitstream and do not necessitate any specific tool from Xilinx. The bitstream organisation being relatively simple, this solution is interesting regarding validation and brings closer the possibility of certifying hardware using DPR on Xilinx FPGA's.

Section 2 of this paper will present dynamic partial reconfiguration in more details, section 3 will discuss certification issues, section 4 will introduce the main concepts on which our solution is based, section 5 will present our minimal flow and its interest, section 6 will discuss the results we get using our flow, and the solution it brings for certification, and finally section 7 will discuss some limitations.

978-3-9810801-2-4/DATE07 © 2007 EDAA

2. Dynamic Partial Reconfiguration

Dynamic partial reconfiguration is a technique that allows to reconfigure a part of a FPGA while the rest of it is still running. This brings new possibilities in terms of performance and flexibility [11] [12] [7]. For example, better performances can be achieved by adding dynamically more resources to process a computation. Flexibility is possible by changing parts of the system while running. That means that one system would be able to implement more functions than usual or to adapt itself to its environment. Usual designs use a static configuration that cannot be changed while running: thus most of its abilities should be included at once and the system would have to be dimensioned so that it could include them all. By using DPR it is only needed to have enough space to fit the biggest module, so less modules will have to be present in the system. Therefore, it enables to use less resources.

DPR provides interesting potential applications in many domains, including software-defined radio [14], evolvable hardware [12], wearable electronics [10]. But it is essentially studied in academic researches. Many different works have already used DPR in their approach, and this is not limited to solutions using FPGA's ([6], [4], [11], [12], [7], [5]). This paper concentrates its efforts on FPGA's from Xilinx, which are the only big reconfigurable matrices able to use dynamic partial reconfiguration since the Virtex-II pro model (i.e.: the VII-pro, V4, and the V5). Up to now using DPR on those devices is clearly not as mature as the usual static use of FPGA's, and is still experimental in many ways. Moreover, it requires proprietary closed source tools and can only be used with the Integrated Design Environment (IDE) of the vendor. Most steps are hidden, and this makes validation impossible.

3. Certification of safety-critical systems

The main concern when designing for a critical application is to lose control on the effective compliance of the products with the initial system specifications. For applications in the civil aeronautical world, this is not acceptable and led to the definition of development standard: the DO-178B standard for software developments [2] and the DO-254 standard [1] for hardware developments. The added value of these standards is to define common rules and requirements for all steps of the product life cycle, from the plans defining all the processes to be implemented in the life cycle, to the maintenance policy describing how the product upgrades will be managed ([8], [3]). In order to comply with a large scope of applications, the standards propose a tailoring taking into account the criticality of the services to be provided, from level D (low criticality) to level A (high criticality). For each level, the standards define the processes to be followed, the controls to be performed on these processes, the required independence level to execute them.

In order to make certification authorities accept a system for a critical application, every requirement of the system must be translated into component requirements and functionalities. 100% of these functionalities must then be validated. Moreover, every step of the design flow as well as the tools must be validated and traceability must be maintained through the entire development process. This enables to prove that the product works as expected at each step and each level of the design flow. A way to certify a complex system is to decompose it in sub-modules that are easier to validate. The frontiers of the modules must then correspond to implemented functions and embody a same level of criticality. Modules interfaces must be validated as well.

Certification is difficult to obtain for FPGA designs since they bring specific complexities. They are programmable, they have a configuration memory and use configuration mechanisms. They increase integration by providing possibilities to put several functions into one device. The FPGA flow has specific additional steps like bitstream generation and loading. In normal operation mode, integrity of the configuration memory needs to be ensured. Indeed it can be altered by soft-errors, generated for instance by SEU (frequent in avionics).

Static designs for FPGA's are already used in many systems for avionics. Nowadays several solutions exist to facilitate certification for FPGA designs. For complex FPGA-based application, one can use a modular design: modules are placed and routed and validated separately. Inter-module communications are realised by using external I/O's in order to further guarantee independence between modules, and avoid failure contagion.

Certification of the usual static configuration hardware of the FPGA can be obtained by the fact that this mechanism is used by a great numbers of designs. Every issue is followed and corrected by the component vendor and there are a great number of tests. To ensure the configuration memory integrity, some systems are continuously monitored by an external controller using CRC and readback capabilities of the configuration port of the FPGA.

A later main problem encountered when using FPGA's in critical applications is the certification of IP blocks and third-party code which are necessary to have a competitive development process.

Upon these issues, in order to use DPR in critical applications, other problems arise:

- Certification of the DPR tools (which are proprietary)
- Partial bitstream validation
- Internal reconfiguration port and hardware (ICAP) validation

- Transient state during reconfiguration process
- Several functional modes for a single module
- More complex designs (particularly if a complex scheduling is used)

4. FPGA reconfiguration

There are four important concepts to know in order to understand how to dynamically reconfigure a FPGA: the configuration interfaces, the bitstream, the bus macros and the constraints.

4.1 Configuration interfaces

FPGA's have a special memory layer dedicated to its configuration which is called the configuration memory. Configuring the FPGA actually means writing configuration data in this memory. Access to configuration memory can only be done through the configuration interfaces of the device. There are multiple interfaces for configuration in the latest Virtex FPGA's: several external interfaces (Select Map, JTAG and serial), and one internal interface. In this paper we concentrate exclusively on the Select Map port and on the internal port. This internal port is called the ICAP (Internal Configuration Access Port).

Figure 1 shows Select Map and ICAP ports. As we can see, the ICAP could be described as a simplified version of the Select Map port. Both of them are basically composed of a data port, a clock signal, and status/control signals. For the ICAP, these signals are the WRITE signal for writing or reading, the BUSY signal (which indicates if the component is ready to send readback data) and the CE (Chip Enable) signal. The Select Map port contains extra control pins enabling to specify the configuration mode (CONFIG_MODE), to specify the start of the configuration (CONFIG_START), to know when the configuration is finished (DONE), or to reset the whole FPGA configuration (RESET_FPGA). The data width of these ports is configurable: it can have a width of 8 or 32 bits.

When configuring the FPGA data are written to a reconfiguration port. These ports allow configuration readback which enables to read to the internal configuration memory. The ICAP is accessible from the user-logic plane of the FPGA, thus it enables self-reconfiguration (i.e. configuration of the device by its own logic). There are two ICAPs in V4 and V5 FPGA's (one in the upper half, one in the lower half).

As illustrated in figure 2, an intermediate logic block, the configuration engine, is present between the configuration port and the memory. The configuration engine manages

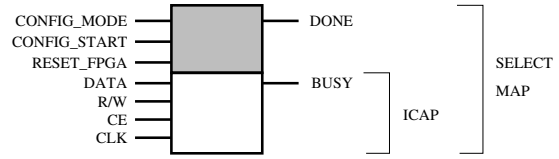

Figure 1: Select Map and ICAP ports

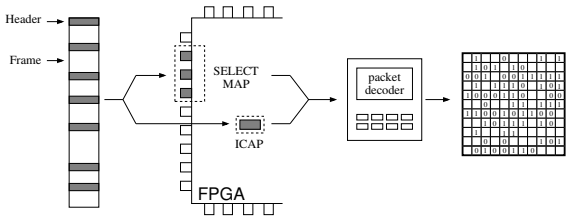

Figure 2: FPGA configuration chain

the configuration of the FPGA and the access to its configuration memory. This engine uses a simple packet-oriented communication model.

4.2 Bitstreams

The bitstream is the configuration file for the FPGA. It consists in the data to be written to one of the configuration interfaces of the device. A bitstream is simply a sequence of packets, each packet containing a header and its data. The headers specify where the data will be written and the number of datawords present in the packet. Data can be used to program the configuration engine or can be configuration data that will be written into the configuration memory of the FPGA. Bitstreams structure is simple to understand, and is explained in details in the Xilinx documentation ([13] and [15]).

Configuration data are composed of frames. A frame is the smallest configurable part of the FPGA. In V4 and V5 FPGA's, a frame is composed of $41 * 32$ bits. Physically, it represents a tile of the logic contained in the device that has the height of a row of the FPGA, and the width of a fraction of a FPGA column. Each frame in the device has a unique address. This frame address is composed of several fields, concatenated: minor address, column, row, location, and type. Knowing the address of a frame, it is possible to find its position in the FPGA, and vice-versa. Configuring any region of a FPGA actually comes to write the configuration data for all the frames that compose this region.

A typical packet writing a frame is presented in fig 3. The header is composed of an address writing instruction,

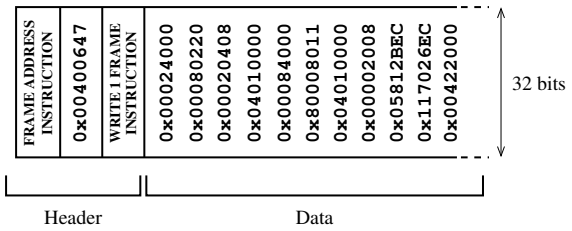

Figure 3: Bitstream packet writing one frame in configuration memory

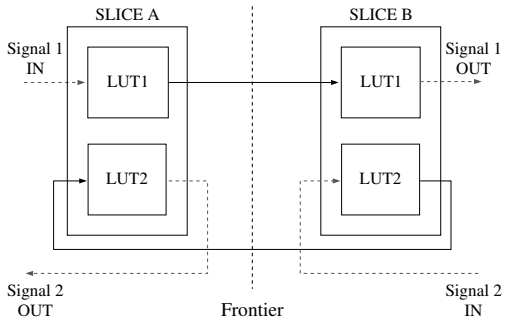

Figure 4: Bus macro made of interconnected LUTs (bidirectional 1-bit connection)

the address of the frame, then a frame writing instruction. The data of the packet are the frame bits. A packet like this one can be followed by packets telling the configuration engine to copy this frame at other places of the configuration memory. This technique, called "Multiple Frame Writing" is used to reduce the size of the bitstream. Packets can contain several frames. In this case, the configuration engine automatically increments the frame address and writes the next frame to the next address[1]. Knowing this, it is possible to identify the packets containing frames in the bitstream, and associate each frame with its address (and thus, its position in the FPGA).

Typically, a bitstream that configures completely a FPGA is composed of three parts: the first is composed of several packets that initiate the device, the second is one long packet containing all the frames, and finally the last is composed of several packets that close the configuration process and start the FPGA.

4.3 Bus macros

Synthesis softwares provide no possibility to impose specific routing between two blocks. Routing is determined by the routing algorithm, which is impossible to predict. This is a problem for DPR since there are multiple configurations for a region of the system, and thus routing of communication resources between the reconfigurable region and the fixed part can change. This can lead to contention and prevent the application from working.

Nevertheless, the usual static flow makes it possible to create pre-routed blocks. One can force a signal to pass through a specific entry of a look-up table (LUT), and insert such a block in a design. These blocks are called "bus macros". Using them, the routing will stay the same in every configuration of the subset since the synthesis application will use it in the design without trying to reroute it. The simplest bus macro consists in two pairs of interconnected LUTs (see figure 4). It enables a 1-bit bidirectional communication between two modules. As a configurable

[1]When using automatic incrementation, two empty frames are added at the end of each consecutive sequence of frames configuring a row.

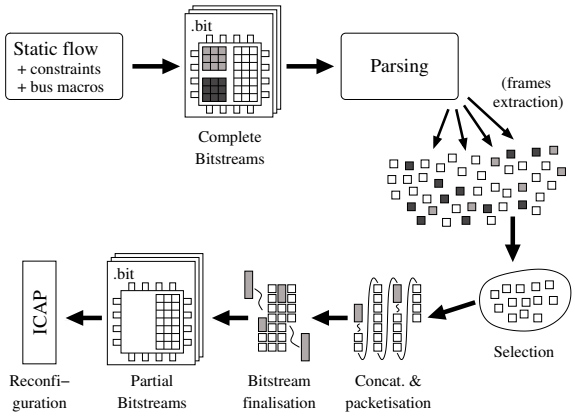

Figure 5: Minimal DPR flow: steps

logic block (CLB) contains 8 LUTs, two complete CLB's are needed to transmit a byte in one direction between two modules. Bus macros must be designed following the communication needs between the modules, and their respective positions in the device.

4.4 Constraints

Another problem with synthesis softwares and DPR is the placement of the resources used in the design. To be able to reconfigure a reconfigurable region, a way of locating it in the device is needed. The solution here is to force this region to be placed inside a specific and identifiable area of the FPGA. This is made possible by using placement constraints. Constraining a logic function in a region requires to correctly evaluate the size of the area beforehand.

5. A minimal flow

The flow we propose is illustrated on figure 5. It is composed of six steps: static synthesis, bitstream parsing, frame

selection, concatenation and packetisation, bitstream finalisation and reconfiguration.

Static synthesis: the first step is the static synthesis of each configuration of the system. The usual static flow is used here. Each design must be divided in a fixed part and a reconfigurable region using constraints, and bus macros must be designed and placed between them if they communicate. One complete bitstream per reconfigurable region configuration is created.

Bitstream parsing: during this step, the bitstreams are read and every frame they contain is extracted. For each bitstream a set with all the frames is created. As the bitstream organisation is known, it is possible to associate each extracted frame with its position in the FPGA.

Frame selection: as every frame composing the reconfigurable region is identified, it is possible to find all of them in the frames extracted in the previous step. Therefore, in this step, every frame from this region is selected from the complete set of frames for each bitstream.

Concatenation and packetisation: once all the frames from the reconfigurable region are selected, they are placed one after the other, in their address order. For each sequence of frames having consecutive addresses, a frame writing header starting at the first address of the sequence is added.

Bitstream finalisation: in this step several packets are added to the ordered frames to compose a valid bitstream. These are the initialisation header and the bitstream tail. They contain instructions that start and close the configuration process. One partial bitstream per reconfigurable region configuration is created during this step.

Reconfiguration: each partial bitstream can reconfigure a part of the FPGA. This is done by writing every dataword composing the bitstream to one of the reconfiguration interfaces. For a self-reconfiguration, it will be the ICAP.

As we could see, this flow is based on tools from the usual static flow. Our flow adds to it a script that realises bitstream parsing, frame manipulation and bitstream composition. Every step has a low or moderate complexity.

6. Results

6.1 Experimental results

A system has been designed to test the flow. It consists in a D8PSK modulator, this modulation is used for control tower/airplane communications during landing of civil airplanes. It can be reconfigured as a QAM-16 modulator. The design has been made using Xilinx ISE 7.1. It is composed of a Microblaze CPU which controls the reconfiguration of the system. The reconfigurable module is constrained in a quarter of the device (a Virtex 4 LX60 FPGA). Upon request, the Microblaze writes a partial bitstream into the ICAP, causing the reconfiguration of the module.

The system successfully reconfigures the module from one modulation to another. About 2800 frames are written during this operation. The bitstream size is about 452kB. Each reconfiguration take about 137ms to complete, which is slow considering a theoretical ICAP bandwith of 100MHz. This is due to the Microblaze CPU.

6.2 Certification

This flow eases the certification of a solution using DPR since it is validable. Indeed it is possible to directly check and validate the source code of the solution, with for example DO-178B. We implemented it as a script of only a few hundred lines long (500 lines of python code), which does simple operations: file reading and writing, address computations, concatenation, etc.

With this validable flow, and under the following hypotheses certification for partial reconfiguration can be considered:

- Consider applications with well-separated functionalities in the fixed and reconfigurable module. It is for example the case in our radio application where an entire waveform resides in the reconfigurable module. Thus the partitioning methodology explained in section 3 can be used.
- The communications between modules are performed outside the component using I/O. This avoids Bus Macros validation.
- Reconfiguration is done using the usual external configuration port. This avoids ICAP validation.
- The component is stopped during reconfiguration. This means that reconfiguration is actually static (not dynamic), but is still a partial reconfiguration. This avoids transient dysfunctioning problems in the fixed part during the reconfiguration process.
- Use an external controller to compute a global CRC of the entire configuration memory. It will prove that, after the reconfiguration process, (i) the integrity of the configuration memory of the fixed part is preserved and (ii) the reconfigurable region holds the expected function. This technique is already used for static designs.

For all this, either the bitstream documentation or the modular design must be reliable and validated so that one can check correct reconfiguration only by using the global CRC.

7. Limitations

Latest and access-limited experimental DPR flow from Xilinx (”Early Access” DPR flow) allows reservation of resources inside modules. This allows to use long communication wires in the FPGA that passes through reconfigurable regions. Our flow is actually unable to do so. Moreover, the two columns in the centre of the device are not supported by our flow at present. These columns contain among others the FPGA clock tree configuration, and its mechanism is not documented. The script implementing our minimal flow is component-dependant: the positions of the different columns composing the FPGA are hardcoded. It is also impossible to configure the device at finer granularity than frame level.

8. Conclusions

In this paper, a minimal flow for DPR on Xilinx FPGA is presented. Partial bitstreams are generated from complete bitstreams issued from the standard static flow. Only operations such as binary file parsing, packet ordering, and file writing are needed. As such, it has been possible to build a telecommunication avionics demonstrator using DPR with a script of only a few hundred lines of code with no support of specific Xilinx tools for DPR.

The certification aspects of hardware development are also presented and show that in order to be used in specific applications such as avionics, systems and tools have to be certified to guarantee safety. In order to certify designs using FPGA's some safe design principles are presented.

DPR causes additional issues regarding certification. We show that many issues can actually be avoided by simplifying DPR mechanisms and applying conservative principles similar to those used in static FPGA designs targeting certification. However, the tools must still be validated.

This paper shows that using the flow presented here, tools certification is made easier, which is not the case when using current solutions for DPR provided by the vendors. This enables to make a step towards the use of dynamically reconfigurable systems in applications that require advanced certification.

Acknowledgements

B. Rousseau and Ph. Manet are funded by the Walloon region, by the NANOTIC and RECOPS projects. The authors would also like to thanks M. Sarlotte and D. Maufroid from Thales Communications France for their help in the writing of this paper.

References

[1] *RTCA DO-254. Design Assurance Guidance for Airborne Electronic Hardware.*, April 2000.

[2] *RTCA-EUROCAE DO-178B/ED-12B. Software Considerations in Airborne Systems and Equipment Certification*, December 1992.

[3] J. Ashley. *Practical Guide to Certification and RE-Certification of AavA Software Elements. Software for Programmable Logic Devices.*, July 2005.

[4] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt. Pact xpp - a self-reconfigurable data processing architecture. *J. Supercomput.*, 26(2):167–184, 2003.

[5] B. Blodget, P. James-Roxby, E. Kelle, S. McMillan, and P. Sundararajan. A self-reconfiguring platform, FPL, Lisbon, Portugal, 2003.

[6] R. David, D. Chillet, S. Pillement, and O. Sentieys. Dart: A dynamically reconfigurable architecture dealing with future mobile telecommunications constraints. In *IPDPS*. IEEE Computer Society, 2002.

[7] R. J. Fong, S. J. Harper, and P. M. Athanas. A versatile framework for fpga field updates: An application of partial self-reconfiguation. *RSP*, page 117, 2003.

[8] A. Hilton and J. G. Hall. Developing critical systems with pld components. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 72–79, New York, NY, USA, 2005. ACM Press.

[9] Hispano-Suiza. Fadec3, a new generation of full authority digital engine controls., 2005.

[10] C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, and L. Thiele. Reconfigurable hardware in wearable computing nodes. In *Proc. Int. Symp. on Wearable Computers (ISWC'02)*, pages 215–222. IEEE, October 2002.

[11] M. Ullmann, M. Huebner, B. Grimm, and J. Becker. An fpga run-time system for dynamical on-demand reconfiguration. In *Parallel and Distributed Processing Symposium. Proceedings. 18th International*, pages 135+, 2004.

[12] A. Upegui and E. Sanchez. Evolving hardware by dynamically reconfiguring xilinx fpgas. In J. M. Moreno, J. Madrenas, and J. Cosp, editors, *ICES*, volume 3637 of *Lecture Notes in Computer Science*, pages 56–65. Springer, 2005.

[13] Xilinx. *Virtex-4 Configuration Guide*, 2006.

[14] Xilinx. *JTRS SDR Kit*, August 2006.

[15] Xilinx. *Virtex-5 Configuration Guide*, May 2006.