Circuit Merging versus Dynamic Partial Reconfiguration -The HoMade Implementation

Jean Perier, Wissem Chouchene, Jean-Luc Dekeyser

Circuit Merging versus Dynamic Partial Reconfiguration - The HoMade Implementation

Jean Perier, Wissem Chouchene, and Jean-Luc Dekeyser

INRIA Lille-Nord Europe DreamPal Project, France
University of Sciences and Technologies of Lille, France

jean.perier@polytechnique.edu
wissem.chouchene@inria.fr
jean-luc.dekeyser@univ-lille1.fr

Abstract

One goal of reconfiguration is to save power and occupied resources. In this paper we compare two different kinds of reconfiguration available on field-programmable gate arrays (FPGA) and we discuss their pros and cons. The first method that we study is circuit merging. This type of reconfiguration methods consists in sharing common resources between different circuits. The second method that we explore is dynamic partial reconfiguration (DPR). It is specific to some FPGA, allowing well defined reconfigurable parts to be modified during run-time. We show that DPR, when available, has good and more predictable result in terms of occupied area. There is still a huge overhead in term of time and power consumption during the reconfiguration phase. Therefore we show that circuit merging remains an interesting solution on FPGA because it is not vendor specific and the reconfiguration time is around a clock cycle. Besides, good merging algorithms exist even-though FPGA physical synthesis flow makes it hard to predict the real performance of the merged circuit during the optimization. We establish our comparison in the context of the HoMade processor.

Keywords

FPGA, partial dynamic reconfiguration, circuit merging, softcore

Circuit Merging versus Dynamic Partial Reconfiguration - The HoMade Implementation

INTRODUCTION

The ability to perform run-time reconfiguration is very attractive because of the occupiedresources lowering it promises and the flexibility it offers. In the case of FPGAs, dynamic partial reconfiguration (DPR) is the reconfiguration solution that comes to mind as it becomes more usable in reconfigurable systems targeting different application fields.This feature consists in reprogramming on-the-fly a given part of the FPGA without suspending the rest of the system execution. In fact, this presents an energy-efficient solution due to the hardware resources sharing. However, DPR technique introduced a new level of complexity in system design. This technique introduced by Xilinx [1] for example generates an overhead in terms of used logic and energy consumption. Implementing the PDR with Xilinx's flow implies instantiating various predesigned components including at least one Softcore (MicroBlaze) [2], a bus (AXI or PLB), an ICAP controller port with bulky complex interfaces and FIFOs. In order to optimize their systems designers have no need to implement all IPs and interfaces provided by Xilinx. In the case of simple applications, some works has been based on this approach to design a hardware implementation of the DPR to reduce architecture complexity and to decrease the reconfiguration time corresponding to the transfer of partial bitstreams through the ICAP interface. It's clear that the DPR presents an efficient technique in such systems but it is still limited in use because some FPGAs do not support it.

Another approach to spare resources and to lower the static consumption is to use resource sharing between different modules of a system. In this solution, two identical components of different modules are transformed into a shared one plus some multiplexers (MUX) to choose the data path if necessary. The modules that share components can be seen as a unique reconfigurable module that can be configured by selecting the desired data path with the MUX. The resource sharing problem applied to circuit has already been well explored and it can be seen as merging the modules Data Flow Graph (DFG). The complex part of the problem is to find a merge that optimize a cost function that represents the occupancy savings. This problem which is similar to a search for sub-isomorphism between graphs is NP-hard[3] and different algorithms to find exact or approximate solutions have been introduced [4][5].

This method has the advantage of always being available, even on low cost FPGA, because it is purely a design problem.

Our final goal would be to study the integration of run-time reconfiguration into massively parallel processor architecture, but first we have to look at the technical reconfiguration solution that exists and measure their impact in a single core environment. The aim of this paper is to establish a comparison of the two reconfiguration methods. We made a bench mark and used the Virtex-7 to draft the comparison. We also used HoMade processor as an environment to perform our tests as it gives us a simple standard reconfiguration frame. This allows us to reduce the general reconfiguration problem to simpler cases.

We begin by describing the different methods that exit to perform DPR and circuit merging. Then we briefly explain the methods we used in our comparison and how we applied them in the context of the HoMade processor. In Section IV we present our test bench and the results and we interpret them in the following section. In section VI we speak about the limitations of our comparison and our future works.

RELATED WORK

Resource sharing is a well-known problem in high level synthesis whose goal is to use a single computation circuit in different modules [5]. The most common approach is to build a compatibility graph between the DFG that one wants to merge and to find the clique that maximizes a certain cost function [6]. The nodes of the compatibility graph represent the possible association between the DFGs and two nodes of the compatibility graph are linked if the two associations can be made simultaneously. The use of heuristics in the clique finding problem allows fastening the algorithm. Another method that gives good results is the use of pattern-matching [7]. This solution uses heuristic methods that were developed to find the longest common subsequence string. The average area reduction achieved by their algorithm for pipelined data path is 55%. The cost function that is optimized is very important as the DFG merge must lead to physical results once implemented in the FPGA (power saving, reduction of slices...). In [8] a cost function is developed that takes into account the whole merged circuit rather than single associations independently. This allows making better choices during the clique finding. To solve the clique problem, Integer Linear Programming can be used such as in [9] where this method is applied to save space in a dynamically reconfigurable application specific coprocessor.

Few solutions take into account the elongation of the critical path caused by the additional multiplexers during the clique finding. An elegant solution is presented in [4]. They formalize the clique finding problem with constraint programming. This allows the addition of a specific constraint that controls the data path latency elongation. Their methods that use the constraint programming solver JaCoP [10] gives an average area reduction of 50% without adding any additional delay. Hence efficient methods are available to perform resource sharing although the problem remains complex because the abstraction chosen (the grain for instance) can have an impact on the result, and no method is therefore fully exhaustive in the search of the optimal physical solution.

Different methods also exit to perform DPR. Xilinx has proposed a simpler design for DPR [11] [12] compared with its basic architecture. The proposed architecture includes a Custom ICAP Processor to manage the ICAP interface without using the OPB_hwicap [13]. The aim of this idea is to alleviate the DPR process by employing a hardware accelerator instead of a Softcore (MicroBlaze). So it permits to reduce many control signals used by the bus interface. The advantage of this architecture is to manage the DPR as a hardware implementation to reduce software complexity. Despite this improvement, the used 100MHz frequency limits the user from obtaining a maximum throughput to transfer bitstreams through the ICAP interface. FaRM [14] is a high speed ICAP controller which is able to perform readback that reduces the configuration overhead. FaRM employ a DMA, ICAP overclocking and bitstream preload into controller. FaRm can operate with overclocked ICAP to 200 MHz and can reach 800 MB.s-1 throughputs. However, FaRM is limited to a certain frequency with variable compression ratios. It depends on the uniformity of the Bitstreams, which explains the variation of maximum throughput. The UPaRC [15] is a controller able to enhance the reconfiguration throughput to 1.433 GB.s-1 at 362.5MHz. UPaRC can also auto-adapt to various performance and consumption conditions. The architecture tasks are managed by a MicroBlaze which provides a lot of control signals. Moreover, the application of a high frequency of 362.5MHz on BRAM reading interface can produce data integrity problems because the BRAM maximum frequency is about 300MHz. The use of the internal memories of the FPGA for such methodology is quite expensive since the size of a bitstream varies depending on the size of the reconfigurable region prompting designers to include a compression /decompression bitstream mechanism to reduce the on-chip BRAM occupied size. The decompression time can affect effectively

the speed of the bitstream transfer. The [16] enhance the UPaRC architecture including a DDR2/DDR3 SDRAM interface managed by MicroBlaze through the MPMC [17] that can operate at 200 MHz of 64-bit. In addition, it includes an ICAP controller operating at 370MHz. It permits to obtain a maximum throughput of 1.48GB.s-1. The system we proposed exceeds all the maximum reconfiguration throughputs of state-of-the-art controllers and offers high-capacity of bitstream storage. These important features permit to construct powerful reconfigurable systems with a large choice of system architectures without the need of bitstream preloading and bitstream compression and decompression. On the other hand, it is clear that the proposed architecture provides a satisfactory flow for high performance applications but it remains dependent on the use MPMC interface which is limited to certain FPGAs.

MERGING METHODOLOGY AND DYNAMIC PARTIAL RECONFIGURATION INTEGRATION IN HOMADE PROCESSOR

HoMade processor

We implemented the two reconfiguration methods in the context of the HoMade processor. This processor is an ultra RISC stack-based processor with only 12 instructions used essentially for controlling the execution flow (i.e. jumps, procedure call/return, master/slaves invocation...). One other instruction is used to trigger IPs via their identifier (ID). The IPs can be arbitrary complex, including ALU functions, register files, load/store units, or even other processors. Therefore it can be considered as an IP integrator. We made our experiments in HoMade in order to have a common ground to establish a comparison.

The HoMade IP frames is the following: there can be between zero and three 32-bit inputs plus a mandatory 10-bit IPcode input and between zero and three 32-bit outputs. The 32-bits inputs and outputs of all the IPs are linked to a common stack. An additional 1-bit output signal can be added if the IP performs in more than one cycle. The IP use it to notify the controller when it has finished. In other cases the IP should perform in less than a cycle. Lastly an IP is triggered when it receives its specific code on the IPcode bus. We integrate the run-time reconfiguration in this context by giving multiple configurations to a same IP. Hence we have chosen that all the configuration of a reconfigurable IP should have the same IPcode and the same number of inputs and outputs to have a similar effect on the stack. A special instruction is added to trigger the run-time reconfiguration of an IP. In the following parts we present the

application of the reconfiguration techniques to the case where we want to transform several IP with the same interface into one reconfigurable IP.

Merging methodology

To use circuit merging based run-time reconfiguration in HoMade we add registers into the reconfigurable IP to encode its configuration state. These registers are then used to control the different multiplexers that have been added. A reconfigurable IP is then given two consecutive IPcodes. When receiving the first one it performs its operation according to its configuration. When receiving the second one it reads the word on the top of the stack and uses it to set the configuration registers. The consecutiveness of the codes lessens the logic to control the IP. Hence it costs one cycle to perform the reconfiguration of the module.

To perform the IP merging we used a method that is described in [4] and is based upon clique finding in a compatibility graph with constraint programming. This method is divided into three steps. The first one builds a compatibility graph. During the second, the clique that maximizes a given cost function is searched. The last step consists in building the merged circuit and adding the multiplexers.

During the first phase, IPs that we want to merge are described as data flow graphs. This description defines the granularity of the merge. Nodes of the DFG are modules of the circuit (adders, multiplier...); they will be considered as black boxes during the rest of the methods: nodes of the DFG are fully shared or not shared at all, subcomponents are not considered. Nodes are abstracted to a type. We also have to define rules for the compatibility. Two nodes of the same type can obviously be associated, but sometimes it is also worth transforming two specialized modules into one more generic. For instance an adder and a subtractor can be merged into a component that can be configured as both. In a way these associations allow to consider finer grain merging without paying the computational price of opening the black box, but it requires some previous work (knowing how to merge an adder and a subtractor in our exemple). An exhaustive list of all the possible associations between two nodes is built. These form the nodes of the compatibility graph (GC). Two GC nodes are linked if the association can be made at the same time. The second phase consists in finding the clique in GC that maximizes the weights. A clique is a sub-graph where all nodes are linked to each other, hence in the case of GC, a clique is a set of compatible association. The cost of the clique is the sum of the weight of its nodes (it

can represent the occupancy saving for instance). The third step is the construction of the merged DFG from the input DFGs and the clique that has been found. It consists in applying the association choice that have been made and to add the necessary MUXs.

The complete solution used more advanced techniques that can be found in [4]. We choose to base our algorithm on the method because it is easy to tune-up. During the first step it is possible to choose the grain and which associations are allowed and the second step allows you to control the critical path elongation. Yet, if the problem merging efficiently two data-flow graph is well resolved in [4], some questions remains before merging two circuits that will be implemented on a FPGA. The first question that arouses is at which level should we decide to share resources. The maximum sub graph isomorphism problem being NP hard [3], large optimal low-level resource sharing would be tedious. Furthermore we do not want to impose gate level synthesis to let some liberty to the FPGA vendor's compiler for optimization, hence we implemented resource sharing at high-level: we try to share adders and bigger modules and we see these modules as black-boxes. Then, FPGA are not ASIC and it is not an easy task to accurately predict which resources will implement a given module before the place and route process (PAR). This could be a problem because the merging tools needs accurate physical information to make decisions during the minimization of the cost function. The prediction of the delay is even harder. Indeed a great part of the delay comes from the interconnect and it cannot be predicted accurately before PAR[18]. Hence, even though the method [4] can control the logical delay, we chose to let it loose because up to a certain point we can suppose that the logical delay that we add will be negligible compared to the noise of the interconnect delay.

Dynamic Partial Reconfiguration

We integrate DPR in HoMade through a dedicated IP to manage the transfer of a partial bitstream to a reconfigurable region (RR) included in a reconfigurable IP. Thanks to a standard interface of a reconfigurable IP, DPR can be easily implemented in HoMade processor. The reconfigurable IP illustrated in Figure 1 can handle several configurations using the same IP_Code. In order to perform DPR during run-time, the dedicated hardware HoMade-IP is called as a regular IP and activated when its specific IP_code is equal to the current execution code given by a user program. It then triggers the reconfiguration process to read a partial bitstream from an external memory to a specific RR. When the

reconfiguration is finished, the module notifies this information to the HoMade controller. With this approach, the reconfiguration is seen as a regular operation in the HoMade processor and it corresponds to a single instruction in the software.

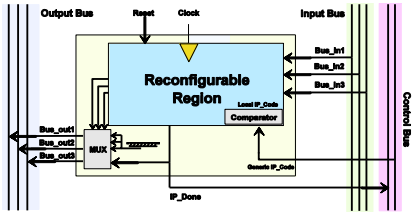

Fig.1: Structure of a reconfigurable IP

Dedicated hardware is also added inside HoMade to store the partial bitstreams and to transfer them to the ICAP. It is a high speed controller based on DDR memory. It provides a rate of 1.425 GB/s at a frequency of 400MHz.

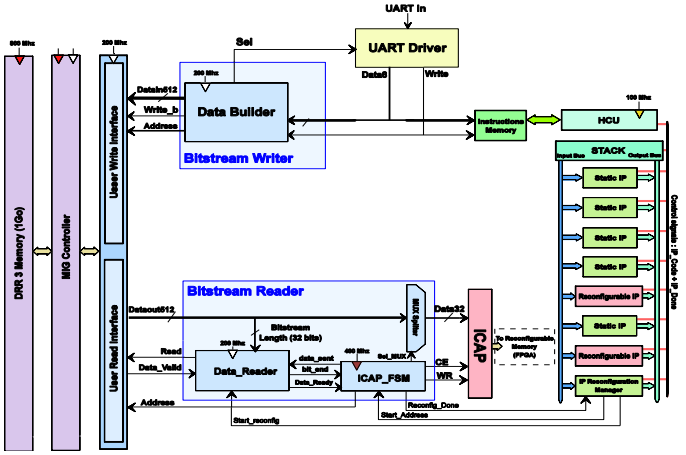

Figure 2: The proposed architecture of the partial dynamic reconfiguration controller based on HoMade processor.

The internal structure of the proposed architecture depicted in Figure 2 consists of two separate data transfer flows: the writing flow and the reading. Both transfer flows use the MIG [19] controller to read or write bitstream words from the DDR memory using a separate 512-bit bus. The writing flow is firstly processed during the initialization phase to transfer a set of bitstreams contained in a single file that contains its length as header trough UART. The file body contains the concatenation of a lot of partial bitstream with their relative length and written in the DDR memory. Next, the user binary program is sent to the HoMade instruction memory. HoMade is started at the end of this process. The reading flow is used during the DPR phase. When the reconfiguration IP is called, it triggers the reading flow with an

address corresponding to the target partial bitstream in the DDR memory. The target bitstream is read from DDR by pieces of 512-bit. These words are then split by a MUX into 32-bit words to be transferred to the ICAP. The first 32-bit word is used to determine the bitstream length. When all the bitstream has been transferred to the ICAP, the module notifies the reconfiguration IP by asserting the IP_Done signal. It then goes back into the initial state, waiting for the next reconfiguration. The "ICAP_FSM" module addressing can transfer a 512-bit in 16 clock cycles at 400MHz frequency. This allows to reduce the reconfiguration duration using burst and pipelined reads from the DDR memory which permit to double the throughput rate since the reading MIG interface is running at 200Mhz.

COMPARISON OF THE TWO RECONFIGURATION TECHNIQUES

The test bench

To get a better understanding of the performance of the DPR and the circuit merging we used a set of six image processing filters: Gaussian, mean, Prewitt-diagonal, Sobel, Kramer and median. The size of the filters is described in Table 1. All these filters use a square matrix of nine 8-bit pixels encoded on three 32-bit words as an input and they all return a single 32-bit output. They are also all single cycle components in HoMade.

Filter	Gaussian	Mean	Prewitt	Sobel	Kramer	Median
LUTs used	82	98	166	166	187	237

Table 1: Size of the different filters

The experiment made was to take HoMade with some of its basic IPs (adder/substractor, shifter, and/or…) and to test the methods on six different cases described in Table 2.

Experiment	1	2	3	4	5
LUTs used	Gaussian Mean	Gaussian Mean Prewitt	Gaussian Mean Prewitt Sobel	Gaussian Mean Prewitt Sobel Kramer	Gaussian Mean Prewitt Sobel Kramer Median

Table 2: Experiments of our test bench

We added a “No reconfiguration” method to our study where each filter of a given experiment is implemented in a different IP, and all these IPs are added to HoMade. In the two reconfigurable methods, all the filters considered in an experiment are put in a single reconfigurable IP. We have chosen to add the filters to HoMade from the smallest to the biggest. The experiments were made inside HoMade but the following figures concern only the filters that where added with the different techniques. We chose to make the measures inside HoMade because the environment may have an impact during the PAR. HoMade occupancy without the filters is 2542 LUTs and 630 registers. The results were measured using Xilinx ISE and Plan Ahead 14.2 tools after the PAR for the virtex-7.

Results

LUTs and registers Occupancy: Figure 3 gives the occupancy in terms of LUTs and registers. During the first experiment, DPR consumes more LUTs than actually putting all the different modules. This can be explained by the overhead of the proxy logic that has to be added for each of the signal that is going between the static and the reconfigurable part. In our case we have four 32-bit buses, so we would expect 128 LUT in overhead. The result is slightly bigger; this might be explained by the shape of the reconfigurable region. It has to be a rectangle of slices. Hence there is a step effect and we can infer the following rule: the minimum number of LUTs used will be the number of LUTs of the smallest rectangle region of entire slices that contains at least the LUTs required by the biggest configuration plus the proxy logic. The solution to this formula is to take a rectangle with one side measuring one slice. But if the module is big, it might lead to a bigger delay because the maximum internal distance would grow linearly with the surface. A trade of has to be manually found.

The experiment 2 to 4 also show the fact that adding many modules of similar sizes do not increase the DPR occupancy as all that as to be taken into account is the maximum module. During the merging of the first four modules, circuit merging is efficient. Indeed the modules of these filters are mainly composed of adders and their structures are close. During the fourth experiment, this solution rises suddenly because Kramer that is composed mainly of comparators does not share much of its logic. Yet during the fifth experience, the investment made by integrating Kramer in the merged module allows

to share many of the median filter comparators. The last two experiments show that DPR becomes competitive when there are enough configurations to compensate the proxy logic price.

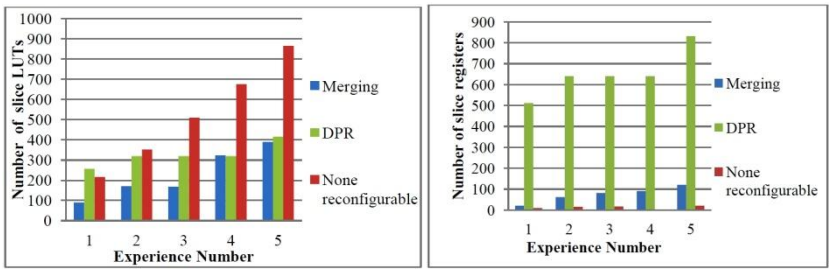

Figure 3: LUTs and registers occupancy in the different experiments

Our filters do not need slice registers as they do not use any flip flop. The few slice registers in the non-reconfigurable test are used as logic element. This optimization made during PAR is more visible in the circuit merging case. We could suppose that there is an increase in the density to respect the timing in the merge module and that using slice registers as logic becomes necessary to follow the increase in density. The case of DPR is very different. There is exactly twice as much required slice registers as slice LUTs. This is because the RR is a rectangle of slice, and all the resources in this rectangle will be privatized for this module. Even though none of the configuration actually uses the RR slice registers, these registers are tied up to our reconfigurable IP. However, it would be inaccurate to say that all unused resources in RR are lost because routing can occur between two static zones through a reconfigurable area.

Occupancy prediction for circuit merging: Figure 4 shows the comparison on the number of LUTs saved by merging circuit technique between the prediction at RTL level and the measure after PAR. The prediction at RTL level is made with Xilinx synthesis report. We believe that the gap that we can see is globally due to the fact that the synthesis report seems to assign an overestimate the cost of the MUX in terms of occupancy.

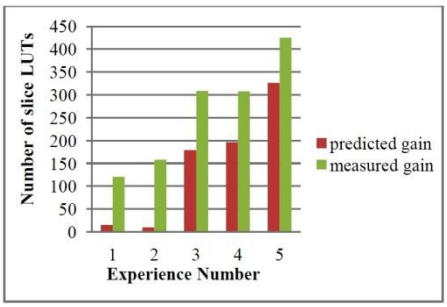

Figure 4: Prediction of the occupancy gain with merging

The gap always tends to be in the same direction which underestimates the merged IP performance. This has less impact than the contrary because with this bias we can suppose that merged IP found during the merging process will save even more once implemented.

Impact on the Critical path: The time measurement corresponds to the maximum delay of a combinatorial path in HoMade. We verified as shown in Figure 5 that this worst delay is always found in the filter IP that we are studying, which is what we expected because all the other IPs are very small in comparison. Hence this measure allows us to judge the impact of the merging techniques on the speed. We added a curve that corresponds to the prediction of the delay in the non reconfigurable case. We can easily make this prediction looking at the delay of each individual filter measured in HoMade (Thus containing a part of the interconnect influence). In every experiment, we could expect this delay to be the worst one of the filters considered because they are in parallel. The timing results are counter intuitive, we would expect the merging module to get worse result than none merged solution. But as we previously stated, the inter-connect and the PAR process has such an important impact that we cannot conclude over which solution is going to be better in our experiment.

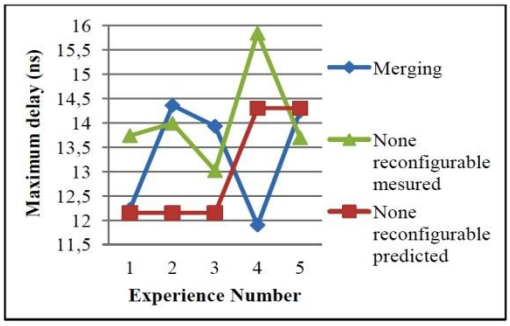

Figure 5: Maximum delay analysis in the case of merging

Even none merged predicted delay is false. One possible explanation in this case would be to accuse the bus interface between the IP and the controller. Adding IPs increases its capacity and increase the complexity of the BUF-T. In the experiment 1-4, this seems an interesting explanation, and indeed a precise measure in the experiment 4 shows that 3 ns were lost in the paths related to the bus compared to the merged solution that do not impact the buses. But the fact that the delay decrease in experiment 5, which is more complex shows that PAR and its optimization brings an imprecision which is bigger than the delay difference in terms of logic, making high-level small timing optimization useless or worse, counterproductive (Non merged in experiment 4). Hence we believe that in the case of design intended for FPGA, aggressive merging based on occupancy saving is worth trying. Obviously, this can sometimes give results with poor timing performance. In 4 and 5 we made an experiment to share the adder and comparator through the use of adder-substractor, the result delay was 29 ns. In the case of DPR the maximum delay is the one of the worst configuration. The results for DPR in terms of timing were good, between 10 to 12ns delay, but they may suffer from the same incertitude due to the PAR.

Reconfiguration time overhead: The last aspect that we compare is the reconfiguration duration. In the case of circuit merging, as we stated previously, the reconfiguration time is a cycle length because of our choice to use registers. In the case of DPR the reconfiguration time linearly depends of the partial bitstream size which is proportional to the surface of the RR. For instance, in the case of the 5th experiment, the size of a partial bitstream is 50ko. With our method that allows a 1,425Go/s reconfiguration throughput it takes 35µs or 1750 cycles to perform the reconfiguration.

Interpretation of the results

Occupancy: Our test bench gives best results for the circuit merging. Of course this may be due in great part to the fact that the modules we studied are close. The 4th experience shows that when adding a very different module, the price to pay is high. On the other hand the more different types are merged together, the more likely a new module will find resource to share in it. For circuit merging, our results also show that the existing merging tools occupancy optimization translates into physical results on FPGA in our experiments.

It's seems that DPR is not meant to be used with a few middle size modules and we have to acknowledge that our test bench is not optimal for DPR. The more there are configurations, the more it may become an interesting option. It can also be interesting with a few big modules with a high ratio of internal logic compare to the number of interface wires. To compensate the effect of the whole RR allocation, it would also seem more reasonable to put configuration that will occupy most of the RR resources.

Impact on the Critical path: Even if it makes sense to say that DPR would result in a smaller critical path (once the proxy logic is balanced), our experiment shows that with middle size modules, the PAR process brings too much noise to draw an absolute conclusion here, though in our case DPR gave best results. Our experiments also allow questioning the usefulness of harsh critical path elongation control during the circuit merging process when implementing the solution on FPGAs.

Reconfiguration time overhead: It is not a big surprise to observe that circuit merging is more than a thousand time faster than DPR. This means that DPR should not be used in an environment with a high frequency of reconfiguration operations. On the other hand, it seems sometimes acceptable to lose some micro seconds once in an interval of more than one second, if it does not impact a critical operation. This means that contrary to circuit merging, the DPR reconfiguration time have to be taken into account when designing the system. If well scheduled, the DPR reconfiguration time overhead could be masked.

LIMITATION AND FUTURE WORKS

Our test bench is limited to small to middle size module; hence we cannot draw a general conclusion from this comparison. Besides if our comparison is made easier thanks to the HoMade context, it may be a bias and the result could differ in another environment. However this limitation to small and middle size module makes sense to us as we are willing to use run-time reconfiguration in a massively parallel environment, where there would be many small reconfigurable modules rather than a big one. We are planning to expand this first comparison in a more detail approach in massively parallel HoMade, where a master can order the reconfiguration of all modules inside all its slaves in a single instruction. We can already predict that in this context, the DPR reconfiguration time would linearly grow with the number of slave cores because for now the DPR of all the RR as to be sequential, even-tough we hope it will not be the case with 3D FPGA. In this context, circuit merging reconfiguration with its

single cycle reconfiguration seems promising, but it remains to be tested in HoMade. Another limitation of this comparison is that we have only studied combinatorial modules. Indeed we have not yet built our theoretical approach of finite state machine merging and register sharing with the context switching problem it may bring to merging. Combinatorial function of different states should also be merged. We plan to build a more complete IP merging methodology and to test it.

CONCLUSION

DPR and circuit merging are two techniques to build run-time reconfigurable systems but our experiments show that they are not meant for the same reconfiguration style. DPR gives more flexibility to the designer because the configuration does not have to be close in terms of logic, and we can hope that in the future it will be easy to design new configuration without having to re-implement the whole system. Yet we have found that this technique is not very optimal for the small and middle size reconfigurable region with few configurations of our bench mark. Circuit merging methods can be scaled-up but the solution finding will take much more time. One of its drawbacks is also that contrary to DPR that is only impacted by the maximum, each configuration added makes it bigger. Yet our experiments show that it gives good occupancy results with small and middle size module and few configurations. Its cycle length reconfiguration time and the 50% space saving compared to none reconfigurable solution makes it an interesting option.

References

[1] Xilinx corporation, “Dynamic and partial reconfiguration” in http://www.xilinx.com/tools/partial-reconfiguration.html

[2] …, “MicroBlaze Processor Reference Guide UG081," 2011.

[3] S. A. Cook, “ The complexity of theorem-proving procedures ”, in Proceedings of the third annual ACM symposium on Theory of computing, 1971, p. 151–158.

[4] C. Wolinski, K. Kuchcinski, E. Raffin, et F. Charot, “Architecture-Driven Synthesis of Reconfigurable Cells”, in 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, 2009. DSD ’09, 2009, p. 531-538.

[5] S. O. Memik, G. Memik, R. Jafari, et E. Kursun, “Global Resource Sharing for Synthesis of Control Data Flow Graphs on FPGAs”, in Proceedings of the 40th Annual Design Automation Conference, New York, NY, USA, 2003, p. 604–609.

[6] N. Moreano, E. Borin, C. de Souza, et G. Araujo, “ Efficient datapath merging for partially reconfigurable architectures ”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no 7, p. 969-980, juill. 2005.

[7] P. Brisk, A. Kaplan, et M. Sarrafzadeh, “ Area-efficient Instruction Set Synthesis for Reconfigurable System-on-chip Designs ”, in Proceedings of the 41st Annual Design Automation Conference, New York, NY, USA, 2004, p. 395–400.

[8] S. Raje et R. A. Bergamaschi, “ Generalized Resource Sharing ”, in Proceedings of the 1997 IEEE/ACM International Conference on Computer-aided Design, Washington, DC, USA, 1997, p. 326–332.

[9] Z. Huang, S. Malik, N. Moreano, et G. Araujo, “ The Design of Dynamically Reconfigurable Datapath Coprocessors ”, ACM Trans. Embed. Comput. Syst., vol. 3, no 2, p. 361–384, mai 2004.

[10]K. Kuchcinski, “ Constraints-driven Scheduling and Resource Assignment ”, ACM Trans. Des. Autom. Electron. Syst., vol. 8, no 3, p. 355–383, juill. 2003.

[11]Xilinx corporation, “Reconfiguring User Logic Using Custom ICAP Processor and Monitoring ICAP Signals Using ChipScope Core Lab”, in http://forums.xilinx.com/xlnx/attachments/xlnx/EDK/23008/1/lab04.pdf

[12]Xilinx corporation, “Driving ICAP Resource”,in http://home.mit.bme.hu/~feher/Reconf_Comp/10_Driving_ICAP.pdf

[13]Xilinx Inc., “OPB_HWICAP (v1.00.b) Product Specification”, DS280, Jul.2006.

[14]F. Duhem, F. Muller, and P. Lorenzini, “FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA, "Reconfigurable Computing: Architectures, Tools and Applications, pp. 253-260, 2011.

[15]Robin Bonamy, Hung-Manh Pham, Sebastien Pillement, and Daniel Chillet, “UPaRC—Ultra-fast power-aware reconfiguration controller”, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012

[16]Hung-Manh Pham, Van-Cuong Nguyen, Trong-Tuan Nguyen, “DDR2/DDR3-based ultra-rapid reconfiguration controller”, Communications and Electronics (ICCE),2012 Fourth InternationalConference on DOI: 10.1109/CCE.2012.6315949 -2012.

[17]Xilinx, Inc, \Logicore ip multi-port memory controller (mpmc)(ds643 v6.03.a),"March 2011.

[18]V. Manohararajah, G. R. Chiu, D. P. Singh, et S. D. Brown, « Difficulty of Predicting Interconnect Delay in a Timing Driven FPGA CAD Flow », in Proceedings of the 2006 International Workshop on System-level Interconnect Prediction, New York, NY, USA, 2006, p. 3–8.

[19] …, “7 Series FPGAs Memory Interface Solutions User Guide UG586” March 1, 2011

About Authors

Jean Perieris a 5th year student at Polytechnic Paris, France and intern at INRIA Lille - Nord Europe DreamPal project. Parallelism is at the core of his research interests. He is more particularly interested in bringing performance though software language compilation optimizations and also allowing software languages to target reconfigurable hardware.

Wissem Chouchene is a PhD Candidate at University of Sciences and Technologies of Lille1, France. His research activities are doing at Centre de Recherche en Informatique, Signal et Automatique de Lille (CRISTAL).His research activities include the network on chip design flow and automatic synthesis of Network interface for SoC based NoC and actually focuses on massively and parallel computing based on dynamic reconfigurable systems.

Jean-Luc Dekeyser is a Professor in computer science at University of Sciences and Technologies of Lille1, France. His research activities are done atFundamental computer sciences Laboratory of Lille. He is also member ofINRIA Lille - Nord Europe Dreampal project. His research activities focus on modeling, design and management of dynamically massively parallel reconfigurable systems. More precisely heinvestigates the following areas:MARTE extensions for reconfigurable systems, Distributed reconfiguration control,Reflexive soft-core processors, the **Homade** processor and massively parallel reflexive processor.