Hardware task context management for fine grained dynamically recon gurable architecture

Samuel Garcia, Jean-Christophe Prévotet, Bertrand Granado

Hardware task context management for fine grained dynamically reconfigurable architecture

Samuel GARCIA and Bertrand GRANADO
Universite Pierre & Marie Curie - Paris 6 - EA2385
94200 Ivry/Seine
email: samuel.garcia@etu.upmc.fr,
bertrand.granado@upmc.fr

J.Christophe Prevotet
Universite de Cergy/Pontoise
ENSEA - ETIS - UMR 8051
95014 Cergy Cedex
email: prevotet@ensea.fr

Abstract—Today's FGDRA[1] could now be regarded, not only as prototyping platforms, but also as reliable alternative to ASICs for consumers products production platforms. To deal with such a system, we propose a middleware layer (RTOS) named SMILE, which could manage not only software process but also hardware tasks running on an FGDRA. Preemption issues for hardware tasks on such a system will be treated, introducing the concept of PDR-SoC[2]. In this paper, our work on hardware FGDRA based task contexts, its management and its evaluation, is exposed.

I. Introduction

The idea of reconfigurable computing has been introduced by Gerald Estrin in [1] in 1960. More recently, popularization of FPGA technologies as brought the subject on the foreground of today's research themes. The application field of this concept is very large, it goes from HPC[3] [2], [3], [4] to embedded SoC [5], [6] including telecommunication systems [7]. Several research have been led on this subject, some using fine grained reconfigurable platforms like FPGA [8], [9], [10], others using coarse grained reconfigurable platform [11], [12].

With the latest platform FPGA generation that integrate hardwired CPU core with the ability to manage the FPGA logic array's configuration (ex : Virtex II pro), the concept of PDR-SoC could now be a reality.

But today, most of time, DRAs[4] are considered as hardware accelerator for very specific tasks, like video encoding as an example. One of the keypoint of our project is that we consider a DRA as a full-fledged computational resource along with CPUs or DSPs. Several experimentation have shown that a fine grained DRA, like an FPGA, can perform any algorithm. Moreover, for any algorithm we can build an optimized architecture running on a FGDRA. In this context, a FGDRA may be seen as a general purpose computing device as well as a microprocessor or a DSP.

From this point of view, it is obvious that dynamical reconfiguration gives a better versatility to a FGDRA and that an operating system is required to manage the system optimally.

The SMILE project, in which this work take place, aim at developing a distributed RTOS kernel which is able to manage a whole DR-SoC including FGDRAs. The FGDRA's kernel must handle any algorithm and thus require preemption ability.

The main challenge of preemption being context management, the present article relates our work on this particular subject. After brief theoretical considerations about hardware task context on fine grained DRA and a short state of the art on the subject, a solution called CSB and several original improvements we made to this method will be presented.

II. Concept of hardware task's context

From a previous study led in our laboratory for the SMILE project, we were able to define what hardware task context exactly is. A task context, in the general case, relates to all the informations that must be saved to continue this task after an interruption, without modifying its work. In the case of a hardware task, this concept needed to be defined.

According to the Huffman model a synchronous architecture consists of several flip-flops (registers) and combinatorial logic blocks. For combinatorial parts, for each input vector we will always get a unique output result, so that for a totally combinatorial task, we do not need to save anything.

In case of a sequential task, the output result depends not only on the input signals, but also on the current state of the system. The current state of a system relates not only on the current control state but also on the current value of variables. This current state thus need to be saved and then restored to be able to stop a hardware task at any time, without any data corruption. Although this current state is defined by both registers and memories, we will focus only on the registers, considering that memory's issues could be managed by some kind of MMU as it's often performed in micro-processor based system.

[1]FGDRA : Fine Grained Dynamically Reconfigurable Architecture , the most known being FPGAs
[2]PDRSoC : Preemptive Dynamically Reconfigurable System on Chip
[3]HPC : High Performance Computing
[4]DRA : Dynamically Reconfigurable Architecture

The hardware task's context can thus be defined as the entire set of information memorized by all the flip-flops that compose this task.

III. State of the art

This section mainly summarizes the survey of the IRISA lab about hardware preemption on FPGA[13].

A. Bitstream based solution[14], [15], [16]

This method consists on reading back the configuration bitstream of the FPGA as it contain all data we want to save. The bitstream transfer can use either the FPGA JTAG chain or a parallel interface such as the SelectMAP/ICAP interface on Virtex Family. But this method presents a lots of drawbacks in both cases, especially for the saving process.

The first drawback is due to the fact that the useful information is scattered among a large quantity of data. So that a lot of useless information must be transferred and the task context must still be extracted from it.

The second drawback is that the structure of this bitstream varies from one FPGA's family to another. Moreover, the read back bitstream is not the same than the configuration bitstream, and not all FPGA family support a read back process.

B. Ad-Hoc solution[17], [18]

This method consists in providing a predefined interface to the application designer (API[5]), the designer must then take context management of his application into account in respect of the given API.

These methods can not be used here as our goal is to simplify the design process of an application, and not to make it more complex. We cannot then relieve us of a hard job on the application designer.

C. Scanpath based solutions

This method is inspired by circuit testing. It consists on providing additional logic to all flip-flops of the design, allowing both a 'normal' running mode and a 'scan' mode. During the scan mode, all flip-flops are chained to build a shift register. We then just need to shift as many bits as there are flip-flops in the design, to read or to write the required information.

This method has many advantages :

- First, it can be easily implemented directly into any existing FPGA just using an HDL definition of special flip-flop. It is thus an universal method.
- Secondly, modifications that must be attempted to a standard design to make it into a preemptable task in a system based on this method can be easily automated.

But this method has also some drawbacks. It introduces overheads in both size and maximum speed of the task and most of all it suffers of poor transfer speed. In this article this last point is addressed.

[5]API : Application Programmer Interface

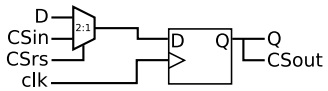

Fig. 1. CSB D flip-flop

IV. Retained solutions and Implementation

A. Scanpath

Taking into account all that we have just seen in the preceding section, we have chosen to concentrate on the use of a scanpath. As we wanted to be able to test different kinds of chainable flipflop, we chosen to work at the lowest possible level.

A previous study has been performed in our lab about hardware task's context saving, that use the JTAG Boundary Scan bus for context transfer. From this previous study, it comes out that the use of the JTAG's Boundary Scan standard increased in an useless manner both data transfer and the surface occupied in the FGDRA. Indeed, each flip-flop having to be doubled, the size of the application was also doubled. Although this experiment validated the use of a scanpath to save and restore an hardware context from a functional point of view, it appeared that there was some weaknesses in this system.

A solution consisted in putting aside the JTAG protocol and preserving only the scanpath to reduce at the same time the configuration sequencer, which does not have to manage JTAG protocol any more and the transfer since all the protocol part is removed.

B. CSB

The main idea of our preemption scanpath is to modify chainable flip-flops to minimize both occupied surface and the number of control signals. We have then introduced the CSB[6]. Whereas previously set up JTAG chainable flip-flops were each made up of three multiplexers, two flip-flops and a combinatorial block, the new CSB flip-flops only require one flipflop and one multiplexer as it can be seen on (fig. 1). Moreover, the old design required five additional control signals versus one for the new one, thus simplifying drastically the place&route process. We just had to this a global clock multiplexer in order to be able of using a different clock frequency in run mode.

The question of the relative slowness of the transmission still remains. To circumvent this issue, two approaches have been considered :

- first saving the contexts of the various hardware tasks, directly into internals ram blocks(eg: BRAM in FPGA). In this way, the operating frequency of our scanpath can be maximized. The transfer time of a context by a scanpath being directly related to its working frequency, this enables to save time.
- secondly using several scanpath working simultaneously. Memories often requiring an 8bit data access,

[6]CSB : Configuration Scan Bus

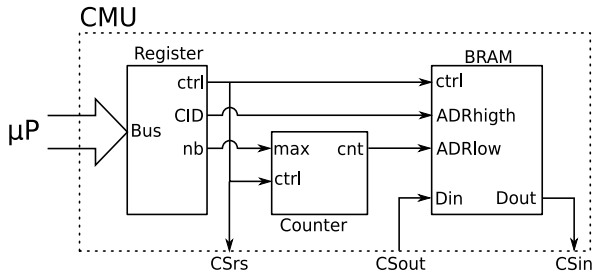

Fig. 2. Context Management Unit

bit	15	14	13 10	9 0
nom	run	S/R	CID	nb

TABLE I
CMU register

we have chosen to use an 8 scanpath architecture called PCS8[7].

C. CMU with embedded Block RAM

Finally, a configuration controller allowing to save or to restore a given context on request have been defined and implemented. This sequencer is directly linked with a BRAM memory thus forming a complete context management block called CMU[8]. Using an external internal BRAM permits to maximize the working frequency of the CMU as it only require internal signals which are quicker than external ones. Moreover, this topology authorizes to place the memory close to the sequencer resulting in a shorter critical path.

This CMU can be directly controlled using a single 16 bits register that can be accessed by a standard microprocessor bus. This register contains four fields, "nb" to specify the number of shift required, "CID" which identifies the context that we want to save/restore, "S/R" to choose whether we want to save or restore the context, and a "run" bit that must be set to 1 to begin the transfer. This bit gets back to 0 automatically when the transfer is completed. This configuration allow to deal with 16 tasks of 1024 flip-flops each for a simple CSB configuration or 8192 flip-flops each for a PCS8 configuration. Note that those numbers have been chosen arbitrarily and can easily be increased if necessary, the only limitation being the restricted amount of RAM in the targeted FGDRA. The CMU offers a simple connection with another system like a micro-processor and provides a simple and classical programmer view of context management. In the long term, CMU should be integrated inside the FGDRA, like MMU are in most microprocessor.

V. Design flow, methodology and tools

As a designer realizes an IP, he spends his time on the functionality of the IP, he does not want nor have the time to add specific material to make his IP preemptable. It is thus necessary to describe a design flow that takes into account this point of view. In such a flow, the flip-flops replacement and chaining must then be automatic. Although some solutions exist for DFT[9] purposes [19][20][21], those solutions were not applicable here. We have thus decided to set up our own solution.

For that purpose, we based our work on a standard workflow as can be seen on figure 3a. The synthesis is done in two stages, RTL synthesis and Technological synthesis (Place & Route). Each stage has two outputs, one for implementation and one for simulation. As an example, the RTL synthesis provides an RTL VHDL model of the design, in our case, as we targeted a Xilinx's FPGA, this model is based on the UNISIM library which is provided by Xilinx.

A TCL script have been developed to add chaining signals to the RTL simulation model, and a technological library called CSBlib that contains a synthesisable VHDL model of every component that can be instantiated at this level have been implemented.

Based on CSBlib and on our conversion script, a new designflow can be proposed (figure 3b) in which only the RTL synthesis is first performed. The conversion script is then executed on the RTL simulation model provided. The whole synthesis (RTL and Technological) must then be performed.

This method has been implemented for CSB and the whole process is quasi-automatic. The final version should be able to deal with a PCS8 context management configuration and should be fully automatic. The current version have been successfully tested on an UART downloaded on OpenCores.org. This show that IP designer do not need to bother about context management while designing an application as any IP can be converted as long as a VHDL description of this IP is provided.

VI. Tests and comparisons

In order to evaluate the proposed architectures, a 7th order fully pipelined FIR filter have been implemented in three different configurations. The first configuration use standard design method (not preemptable), the two others use respectively a CSB and a PCS8 configuration. Note that in order to simplify the design we chose to fix all coefficients to one.

For each test, pre and post-synthesis simulation were performed. At the light of those tests, we can say that both CSB and PCS8 effectively allow to save and to restore an hardware task's context. Quantitative results about used place and speed are extracted from synthesis reports. Note that those tests were made targeting a Xilinx Virtex II-Pro
XC2VP30FF896 FPGA, but we can use any other FGDRA as long as it provide sufficient internal memory.

[7]PCS : Parallel Configuration Scan 8bit
[8]CMU : Context Management Unit
[9]DFT : Design For Test

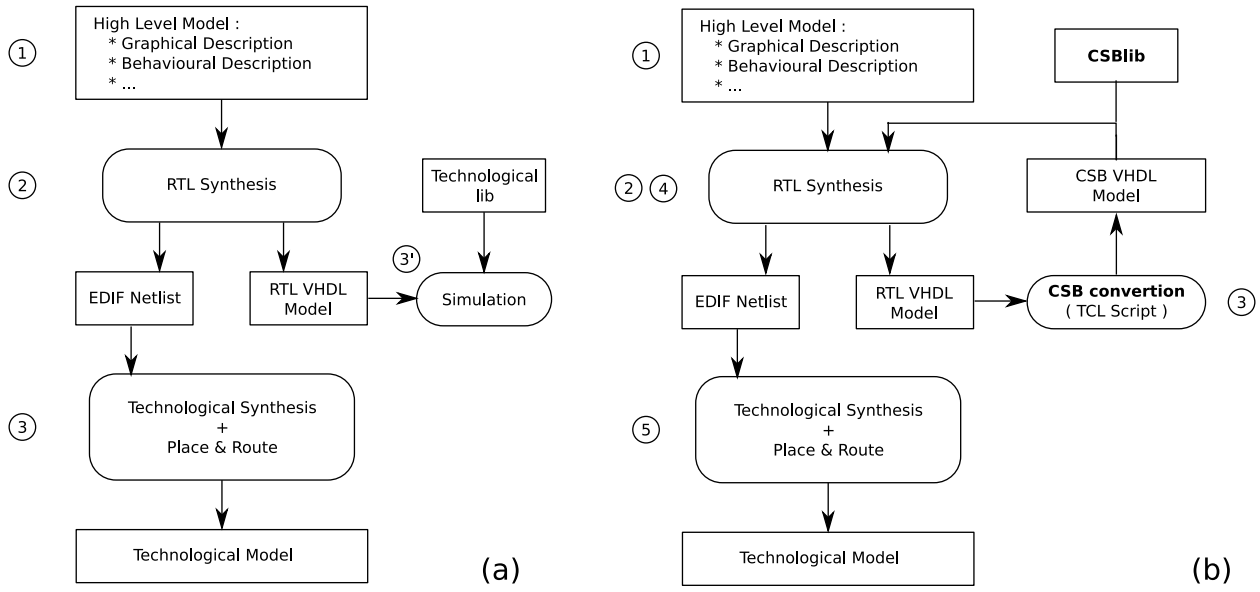

Fig. 3. CSB design flow

	Std	CSB	PCS8
Flipflops	128	180	176
LUT	60	228	166
run Fmax (MHz)	350	300	250
scan Fmax (MHz)	-	200	200
Context transfer time (ns)	-	640	80

TABLE II
FIR7 + CMU results

A. Area considerations

As expected, the use of both methods implies an overhead in term of FGDRA occupation. The main problem with area occupation on FGDRA is that all results highly depend on optimizations performed during the synthesis process. In table II we can see that the PCS8 method uses less LUT that CSB. But after further investigations it appears that due to the FIR architecture, seven 8 bits registers from the first pipeline stage are chained the same way in run mode or in scan mode. So that during synthesis process, data input multiplexers for those seven register (representing 56 flipflops) are actually not implemented.

Another optimization artifact is replica flipflops. As an example, the CSB based CMU alone require 37 flipflops and the FIR use 128, so the complete task including both should require 165 flipflops, but it actually requires 180. This is due to the fact that, for timing optimization purpose, the synthesizer creates replicas of some flipflops of the design. To study the real impact of our context management method, we then should not consider replicas as they are due to synthesis optimization, not to the context management method.

To conclude, we can give an estimation of area required for a given task when using CSB or PCS8 context transfer method and a CMU. Given $N_{ff_{STD}}$ and $N_{lut_{STD}}$ respectively the number of flipflops and LUT used for the standard version of the task, equations 1 and 2 gives the number of flipflops and LUT required for the CSB based preemptable version of this task.

$$N_{ff_{CSB}} = N_{ff_{STD}} + 37 \quad (1)$$

$$N_{lut_{CSB}} = N_{lut_{STD}} + N_{ff_{STD}} + 1 + 40 \quad (2)$$

Equation 3 and 4 give similar result for a PCS8 based preemptable task.

$$N_{ff_{PCS8}} = N_{ff_{STD}} + 33 \quad (3)$$

$$N_{lut_{PCS8}} = N_{lut_{STD}} + N_{ff_{STD}} + 1 + 35 \quad (4)$$

Note that those equation do not take synthesis optimization into account so that actual result can be sightly different. Memory used for the CMU is not mentioned here, when using a Virtex II pro, both PCS8 and CSB CMU use one BRAM in the actual configuration.

B. Time consideration

Concerning time there are two important parameters, the first is context transfer time, directly proportional to the scan mode max frequency, and the second is the run mode max frequency.

When adding CSB or PCS8 preemption ability to a given task, the maximum application's working frequency is slowed down. This could be a serious problem and probably constitutes the main drawback of our system. This speed penalty is less important for CSB than for PCS8 but if the working frequency is not of a big issue here the penalty in term of overhead is largely compensated by the 8x gain due to parallelization.

Those results demonstrates that PCS8 has a clear advantage in term of overhead. But if the applications had to work at a frequency close to the FGDRA's limits, a CSB configuration should be considered.

VII. Conclusion

A complete hardware task context management method have been set up, based on a management unit, the CMU, and using two different context transfer architectures, one based on a simple scanpath, the CSB, and the second based on eight scanpath running in parallel, the PCS8. A quasi automatic designflow have also been proposed to build a preemptable hardware task, using our context management method, from a common VHDL description, using standard synthesis tool, a custom build library and a custom conversion script.

Our method has the particularity to be universal as it can use any synthesizable VHDL IP as input and can target any existing FPGA as long as it has sufficient ressources for the given applications. The most noticeable drawback of this method being that it slows down the application's maximum frequency.

Using this method, a hardware task can be preempted at any time. In some case a given task should not be preempted while performing a particular action, an access to an external pipelined memory is an example of such an action. For this purpose, the next version of the CMU will provide a special flag to indicate that the task is in a critical section.

At this time, we work on a midleware side of RTOS for PDR-SoC and the real time scheduling of hardware task on FGDRA. The second part of our work will concentrate on modifications of hardwired architecture of an FGDRA to make it more efficient for dynamical reconfiguration. We work on a new FGDRA architecture that directly implement hardwired PCS8 and CMU.

References

[1] G. Estrin, "Organization of computer systems, the fixed plus variable structure computer," Western Joint Computer Conference, pp. pp. 33–40, 1960.

[2] R. Hartenstein, "Reconfigurable computing : urging a revision of basic cs curricula," in WLSPD, 2002.

[3] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," ACM Computing Surveys, vol. 34, p. 171210, 2002.

[4] Cray, "Cray xd1 datasheet," Cray," Datasheet, 2005.

[5] J. Cambonie, S. Guerin, R. Keryell, L. Lagadec, B. Pottier, O. Sentieys, B. Weber, and S. Yazdani, "Compiler and system techniques for soc distributed reconfigurable accelerators," in SAMOS, 2004.

[6] B. Blodget, C. Bobda, M. Huebner, and A. Niyonkuru, "Partial and dynamically reconfiguration of xilinx virtex-ii fpgas," in FPL, 2004.

[7] M. Tommiska, "Reconfigurable computing in communications systems," Master's thesis, HELSINKI UNIVERSITY OF TECHNOLOGY, 1998.

[8] P. Sedcole, P. Cheung, G. Constantinides, and W. Luk, "Structured methodology for system-on-an-fpga design," in FPL, 2004.

[9] F. G. Moraes, D. Mesquita, J. C. Palma, L. Muller, and N. Calazans, "Development of a tool-set for remote and partial reconfiguration of fpgas," in DATE, 2003.

[10] Xilinx, "Two flows for partial reconfiguration: Module based or difference based," Xilinx," Application Note, 2004.

[11] R. David, D. Lavenier, and S. Pillement, "Du micro-processeur au circuit fpga, une analyse sous l'angle de la reconfiguration," in TSI, 2003.

[12] S. Pillement, R. David, and O. Sentieys, "Architectures reconfigurables : opportunites pour la faible consommation," in FTFC, 2003.

[13] J. LALLET, "Vers des architectures a reconfiguration dynamique preemptive," Master's thesis, Universite de Rennes 1, 2005.

[14] H. Simmler, L. Levinson, and R. Manner, "Multitasking on fpga coprocessors," in FPL, 2000.

[15] D. Koch and J. Teich, "Platform-independent methodology for partial reconfiguration." in Computing Frontiers, 2004.

[16] D. Koch, A. Ahmadinia, C. Bobda, and H. Kalt, "Fpga architecture extensions for preemptive multitasking and hardware defragmentation." in FPT, 2004.

[17] K. Danne, "Memory management to support multitasking on fpga based systems," in ReConFig, 2004.

[18] ——, "Operating systems for FPGA based computers and their memory management," in ARCS, 2004.

[19] MentorGraphics, "Dftadvisor reference manual," Mentor Graphics," Reference Manual, 1999.

[20] C. Aktouf, H. Fleury, and C. Robach, "Inserting scan at the behavioral level," IEEE Design & Test of Computers, vol. JulySeptember 2000, pp. 34–42, 2000.

[21] O. Laouamri and C. Aktouf, "Enhancing testability of system on chips using network management protocols," in DATE, 2004.