

Design of a Fine Grained Dynamically Reconfigurable Architecture for OS Support

Samuel GARCIA and Bertrand GRANADO
 ETIS, CNRS, ENSEA, Univ Cergy-Pontoise
 95014 CERGY, France
 email: samuel.garcia@ensea.fr,
 bertrand.granado@ensea.fr

Abstract—In the context of large versatile platform for embedded real time system on chip, a fine grained dynamically reconfigurable architecture (at a first approach an FPGA) can be used as one possible computational resource. In order to manage this resource in that kind of context we need an OS kernel able to manage such a resource. Both the history of micro-processor based system and our previous work based on currently available FPGA devices led us to think that not only an OS kernel must be defined to handle an FGDRAs but a FGDRAs must also be designed to handle this OS kernel. This article relate our original work in this direction. OLLAF¹, an original FGDRAs core that we have designed and modeled using VHDL will be presented as well as a more general view of our approach of a FGDRAs and its related OS kernel.

I. INTRODUCTION

This work takes place in the SMILE project. This project aims at providing a distributed middle layer to efficiently handle the complexity of a tomorrow's RSoC². This system may contain several computing units of different type. It will embed at less one or more General Purpose Processor (GPP), but also dynamically reconfigurable architectures (DRA) at different granularities. Tomorrow's computing systems has to comply with lots of constraints. Those constraints may be time related, to meet real time requirements, but also power consumption constraints, as it is, and will be more and more, one of the primary concern of electronical devices.

FGDRAs³ are today the platform of choice when it comes to handle tasks in a highly computational constrained context. In more general terms FGDRAs can achieves much better efficiency than GPP does, while offering the same versatility and, potentially, a very close flexibility. The counterpart is that it introduces a much greater complexity for application designers. This complexity could be lowered to an acceptable level in two ways. First by providing powerful CAD tool. Lots of research are thus led in the field of high level synthesis [1]. The second way is to abstract the system complexity by providing a middle layer, e.g an operating system, that abstracts the lower level of the system [2]. Moreover, an OS could manage new tasks at run time. This property is a feature of importance for

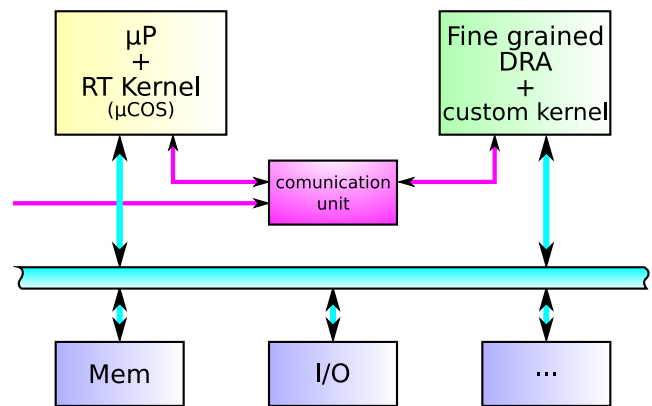


Fig. 1. Basic view of a RSoC in SMILE

DRA. For all those reasons, a specialized operating system is required for FGDRAs.

In our work we make a difference between a FGDRAs, which is a general term, and a FPGA which, for us, relate to an actual silicon device sold under this designation.

The SMILE project follows a distributed approach of the system. Each computing unit of a RSoC (GPP, DSP, DRA, ...) has its own real time kernel. This topology allows to use a specific custom made real time kernel for each computing unit. This enable to take into account every specificities of each computing unit. A message passing communication scheme, based on MPI⁴, ensure a consistent operation of the whole system. In this frame of mind, we developed a dedicated real time kernel for a FGDRAs.

Both the history of micro-processor based system and our previous work based on currently available FPGA devices led us to think that not only an OS kernel must be conceived to handle a FGDRAs, but a FGDRAs must also be designed to support efficiently this OS kernel. This article relate our original works in this direction. The FGDRAs core that we have designed will be presented as well as a more general view of our approach of a FGDRAs and its related OS kernel.

The reminder of this paper is organized as follows. Section 2 discusses related work in the field of OS for FGDRAs. Section

¹Operating system enabled Low LATency Fgdra

²Reconfigurable System on Chip

³Fine Grained Dynamically Reconfigurable Architecture, FPGA being a case of FGDRAs

⁴Message Passing Interface

3 explains an original FGDR platform proposition called OLLAF what we have designed and modeled using VHDL. Section 4 discuss more precisely of the basic reconfigurable core in the point of view of a task designer and more precisely on the reconfiguration point of view. Section 5 explain how our architecture can affect different OS services. Finally, conclusion are drawn in section 6.

II. RELATED WORK

A. OS for FGDR

Several research have been led in the field of OS for FGDR[3], [4], [5], [6]. All those studies present an OS more or less customized to enable specific FGDR related services. Example of such services are : partial reconfiguration management, hardware task preemption or hardware task migration. They are all designed on top of a platform composed of a commercial FPGA and a micro-processor. This microprocessor may be a softcore processor, an embedded hardwired core or even an external processor.

In the 90's, some works have also been published about the design of a specific architecture for dynamical reconfiguration. In [7] authors discuss about the first multi-context reconfigurable device. This concept as been implemented by NEC on the Dynamically Reconfigurable Logic Engine (DRLE) [8]. At the same period, the concept of DPGA was introduced, it was also proposed in [9] to implement a DPGA in the same die as a classic microprocessor to form one of the first SoC including dynamically reconfigurable logic. In 1995, Xilinx even applied a patent on multi-context programmable device proposed as an XC4000E FPGA with multiple configuration planes [10].

More recently, in [11], authors propose to add special material to a DRA to support OS services, they worked on top of a classic FPGA. Some works as in [12] focus on technological improvement of FGDR like using a MRAM based extra configuration plane. Other focus more on high level generic modelization of a DRA of undetermined granularity [13].

The work presented in this paper try to take advantage of those previous work both about hardware reconfigurable platform and OS for FGDR.

B. previous work

Our first work on OS for FGDR was related to preemption of hardware task on FPGA[14]. For that purpose we explored the use of a scanpath at the task level. In order to accelerate the context transfer we explore the possibility of using multiple parallels scanpaths. We also provided the Context Management Unit or CMU, which is a small IP capable to manage the whole process of saving and restoring tasks contexts.

In that study both the CMU and the scanpath were build to be implemented on top of any available commercial FPGA. This approach showed number of limitations. They could be summarized in this way: implementing this kind of OS related material on top of the existing DRA introduce unacceptable overhead on both the task and the OS service. Differently said,

most of OS related material should be as much as possible hardwired into the platform's architecture.

III. OLLAF : GENERAL OVERVIEW

A. Specifications of a FGDR with OS support

We have designed a FGDR with OS support following those specifications. This FGDR called OLLAF has been designed as a synthesizable VHDL model.

It should first address the problem of the configuration speed of a task. This is one of the primary concerns because if the system spend more time configuring itself than actually running tasks, then its efficiency will be poor. The configuration speed will thus have a big impact on the scheduling strategy.

In order to enable more choice on scheduling scheme, and to match some real time requirement, our FGDR platform must also include preemption facilities. For the same reasons than configuration, the speed of context saving and restoring process will be one of our primary concerns. On this particular point, previous work we have discussed in section 2 will be adapted and reused.

Scheduling on a single GPP system is just a matter of time. The problem is to distribute the computation time between different tasks. In the case of a DRA the system must distribute both computation time and computation resources. Scheduling in such a system is then no more a one dimensional problem, but a three dimensional one. One dimension is the time and the two others are the surface of reconfigurable resources. Performing such a scheduling at run time with real time constraints is at this stage not conceivable. But the FGDR should help getting close to that goal. The primary concern on this subject is to ensure an easy task relocation. For that, the reconfigurable logic core should be splitted into several equivalent blocs. This will allow to move a task from a bloc to any another bloc or from a group of blocs to another group of blocs of the same size and the same form factor without any change on the configuration data. The size of those blocs would be a tradeoff between flexibility and scheduling efficiency.

Another aspect of an operating system is to provide inter task communication services. In our case we will distinguish two cases. First the case of a task running on top of our FGDR and communicating with another task running on a different computing unit, for example a GPP. This case will not be covered here as this problem concern the whole heterogeneous platform, not only the particular FGDR computing unit. The second case is when two, or more, tasks run on top of the same FGDR communicate together. This communication channel should remain the same wherever the task is placed on the FGDR reconfigurable core and whatever state those tasks are (running, pending, waiting, ...). That mean that the FGDR platform must provide a rationalized communication medium including some sort of exchange memories.

The same arguments could also be applied to inputs/outputs. Here again two cases exist. First the case of I/O being a global resource of the whole platform. Secondly the case of special I/O directly bound to the FGDR.

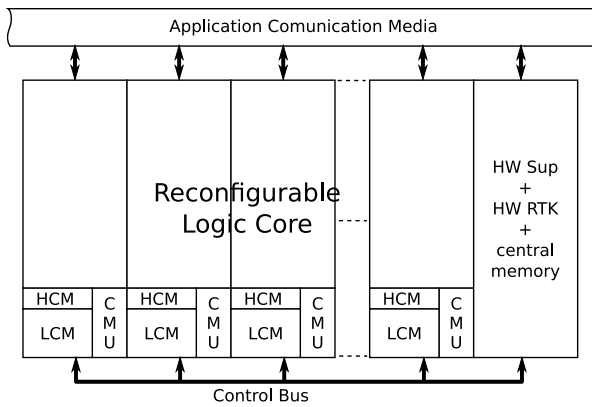


Fig. 2. Global view of the FGDR

B. Proposed solutions

Figure 2 show a global view of OLLAF, our original FGDR designed to support OS services as they have just been specified.

In the center stand the reconfigurable logic core of the FGDR. This core is organized in columns, each column can be reconfigured separately and offer the same set of services. That means that a task use an integer number of columns. This topology as been chosen for two reasons. First using a partial reconfiguration by column transform the scheduling problem into a two dimensional problem (time + 1D space) which will be easier to handle in real time situations. Secondly as every columns is the same and offer the same set of services, tasks can be moved from one column to another without any change on the configuration data.

In the figure, at the bottom of each column you can notice two hardware blocs called CMU and HCM. The CMU as said earlier is an IP able to manage automatically task's context saving and restoring. The HCM standing for Hardware Configuration Manager is pretty much the same but to handle configuration data also called bitstream. On each column a local configuration/context memory is added. This memory can be seen as a first level of cache memory to store contexts and configurations close to the column where it might most probably be required. The internal architecture of the core provides adequate materials to work with CMU and HCM. More about this will be discussed in the next section.

On the right of the figure stand a big bloc called "HW Sup + HW RTK + central memory". This bloc contain a classic microprocessor which serve as a hardware supervisor. It runs a custom real time kernel specially adapted to handle FGDR related OS services and platform level communication services. Along with this hardware supervisor a central memory is provided for OS use only. Basically this memory will store configuration and eventual context of every task that may run on the FGDR. This supervisor communicates with all columns using a dedicated control bus.

Finally, on top of the figure 2 you can see the application communication medium. This communication medium provides a communication port to each column. Those com-

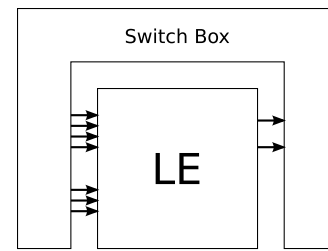


Fig. 3. Basic reconfigurable tile

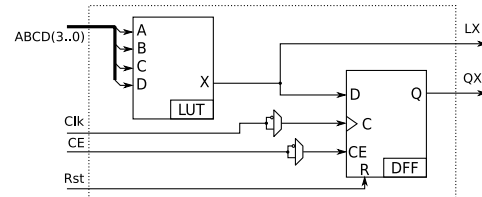


Fig. 4. Functional, task designer point of view of LE

munications ports will be directly bound to the reconfigurable interconnection matrix of the core. If I/O had to be bound to the FGDR they would be connected with this communication medium in the same way reconfigurable columns are.

IV. RECONFIGURABLE LOGIC CORE

FGDR core could be considered with two points of view. The first one is the functional point of view, it consists on the information that a task designer may have to know in order to design the task's architecture. The second point of view is the configuration point of view, it consists on information about reconfiguration plane. As one of the main goals of the OS is to abstract configuration management, this point of view could be seen as the OS point of view.

A. conceiver point of view

The FGDR core is composed of several basic tiles (figure 3). Each tile consist on a classic Logic Element and a Switch Box. The switch box is in fact an elementary part of the reconfiguration matrix. In other words the reconfiguration matrix is build by connecting adjacent tile's switch box. This allow to easily generate a reconfigurable core of any desired size.

Internal architecture of a LE in the functional point of view can be seen on figure 4. This architecture integrates elements that compose a classic Logic Element of FGDR. The OS point of view don't impact the functional point of view and if we want to modify the functional architecture, it should not change our conclusion on OS point of view.

B. configuration point of view

In the configuration point of view only the configuration plane will be discussed.

Each configuration memory point is at first modelized as a flip flop. In order to perform reconfiguration they are chained to form a scanpath. As configuration speed have to be increased as much as possible our reconfigurable logic

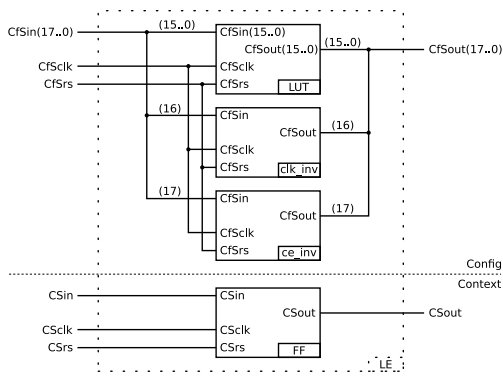


Fig. 5. Configuration point of view of LE

core will use multiple parallel scanpath. As an example the 18 configuration memory points of one of our LE will be each part of a different scanpath (16 for the 4bit LUT and 2 configurable inverter). That mean 18 parallel scanpaths for each LE columns and then a reconfiguration time 18 times shorter than with one unique scanpath. Those scanpaths are not independent, they all share the same clock and control signals. For that reason in the remaining of this section it will be referred as a unique configuration scanpath, the path being now a word, not a bit.

When a task is preempted, the context of this task must be saved. During this process only the information stored in the LE's flipflops are required. For this reason LE's flipflops are connected to another totally independent scanpath. This scanpath use a different clock and control signals.

Both configuration and context scanpath use equivalent signals. Signals for the configuration scanpath use the Cfs prefix and context scanpath ones use the CS prefix. For each scanpath there is basically three signal, one clock signal *clk*, a mode selection signal *rs*, and the chain signal named *in* at the input of a flip flop or a bloc and *out* at the output.

In order to even lower the configuration and context managing overhead, our reconfigurable logic core use a double memory plane. A configuration memory point is not one flipflop but two flipflop with some switching material. Architecture of this memory point can be seen on figure 7. *Run* and *scan* are then no more two working mode but two parallel plane which can be swapped as will. With this topology, configuration and context of a task can be shifted while the previous task is still running. The effective task switching overhead is then taken down to one clock cycle.

Let's consider that a task T1 is preempted to run another task T2, scenario of task preemption is then as follow :

- T1 is running and the scheduler decide to preempt it to run T2 instead
- T2's configuration and eventually context is shifted on the second configuration plane
- once the transfer is completed the two configurations planes are switched
- now T2 is running and T1's context can be shifted out to be saved

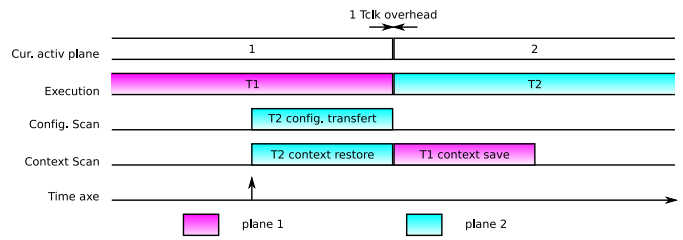


Fig. 6. Typical preemption scenario

This scenario is illustrated in figure 6.

The only constraint of this system is that scheduling decision have to be taken in advance in order to take advantage of the dual configuration plane.

In OLLAF, both context and configuration transfers are hidden due to the use of a dual configuration plane. The latency L between the moment a preemption is asked and the moment the new task effectively begin to run can also being studied. This latency only depends on the size of the columns. That means that for a given platform, it will be a constant. In the worst case this latency will be far shorter than the OS tick period. OS tick period being the shortest time in which the system can respond to an event, we can consider that this latency will not affect the system at all under normal operation. In the case of an interruption, would then be perceptible. It hasn't been measured yet but we can say that this latency will be fully determined for a given platform. In an extreme case, a task that should respond immediately to a specific event should be run as a critical section so that it will be always active.

C. Results

During our past works on preemption for FGDR, we used a fixed coefficient seventh order FIR as test application. We then want to compare the estimated time overhead of a task preemption using our new FGDR with the one obtained using our old context management scheme and a Xilinx ICAP interface. In order to make the comparison possible we will consider a 128x1 tiles column, our FIR architecture using 128 LEs. Each tile use 100bits of configuration memory. To reconfigure a 128 tiles column we thus need 12800bits. The context of the task is composed by 128bits. To perform the preemption we need to save the context of the preempted task and then to restore the one of the new task we want to run. We thus have to transfer 256 context bits. The Xilinx ICAP interface have a throughput of 32bits per clock cycle while using CSB for context transfer a throughput of one bit per clock cycle is performed. Using our FGDR both operation can be performed at the same time and the switching time cost is always of one clock cycle. Using those estimated data, the comparison between those two solution are shown in the table I.

V. CONFIGURATION, PREEMPTION AND OS INTERACTION

In previous section an architectural view of our FGDR has been exposed. In this section, impacts this architecture will

	ICAP + CSB	our FGDR
Context transfer	128×2	1
Configuration	400	-
Total task switching	656	1

TABLE I

ESTIMATED COMPARISON OF TASK PREEMPTION OVERHEAD USING OUR NEW FGDR OR A XILINX ICAP INTERFACE AND OUR OLD CSB CONTEXT TRANSFER METHOD. (GIVEN IN CLOCK CYCLE)

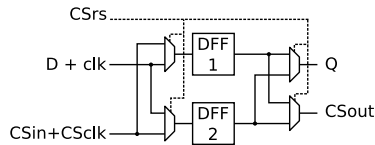


Fig. 7. Dual plane memory point

have on OS services will be discussed. We will here consider the three service most specifically related to the FGDR.

First, the configuration management service. On the hardware side, each column provides a hardware configuration manager and an associated local memory. As stated earlier that mean that configurations have to be placed in advance in the local configuration memory. The associated service running on the hardware supervisor micro-processor will thus need to take that into account. That imply that this service must manage some sort of intelligent cache to prefetch task configuration on the columns where it might most probably be placed. In order to do so, some sort of anticipated scheduling must be performed.

Secondly, the preemption service. The same principle must be applicable here as those applied for configuration management. Except that contexts also have to be saved. The context management service must ensure that it never exist more than one context for each task in the entire FGDR. Context must thus be transferred as soon as possible from local context memory to the centralized global memory of the hardware supervisor. This service will also have a big impact on the scheduling service as the ability to perform preemption with a very low overhead allow the use of more flexible scheduling algorithms.

And last the scheduling service and in particular the space management part of the scheduling. It takes advantage of the column topology and of the centralized communication scheme. As stated, fewer computing power will be required to manage a one dimensional space at run time. The problem is here similar to memory management in classical GPP based system. The reconfigurable resource could then be managed as a virtual infinite space containing an undetermined number of columns. The job is then to dynamically map the required set of columns (task) into the real space (the actual reconfigurable logic core of the FGDR).

VI. CONCLUSION AND PERSPECTIVES

A global view of OLLAF, an original FGDR that enhance OS service support has been presented, and in more details

its reconfigurable logic core. We claim that OS and platform must be closely linked to each others in order to perform as optimally as possible.

We showed that this architecture can enhance the preemption efficiency and ease task relocation. Thanks to that, both time and space scheduling complexity can be drastically reduced. The operating system should then be able to run those services efficiently at run time under real time constraints.

Today, the reconfigurable logic core have been modeled using VHDL and is being tested by several simulations. The rest of the FGDR is also in progress. The dedicated custom OS service are being written as an extension of $\mu C/OS-II$, a well proven real time OS. We are also working on the distributed management of the whole heterogeneous system including, at least, one of our FGDR and its dedicated real time kernel, and one GPP.

REFERENCES

- [1] P. Coussy, G. Corre, P. Bomel, E. Senn, and E. Martin, "High-level synthesis under i/o timing and memory constraints," in *Proceeding of IEEE International Symposium on Circuits and Systems (ISCAS)*, 2005.
- [2] Q. Deng, S. Wei, H. Xu, Y. Han, and G. Yu, "A Reconfigurable RTOS with HW/SW Co-scheduling for SOPC," in *International Conference on Embedded Software and Systems (ICESSE)*, 2005, pp. 116–121.
- [3] H. Simmler, L. Levinson, and R. Manner, "Multitasking on FPGA Coprocessors," in *Field Programmable Logic and its Applications (FPL)*, ser. Lecture Notes in Computer Science, no. 1896, 2000, pp. 121–130.
- [4] G. Chen, M. Kandemir, and U. Sezer, "Configuration-Sensitive Process Scheduling for FPGA-Based Computing Platforms," in *Design Automation and Test in Europe (DATE)*, 2004, pp. 486–493.
- [5] H. Walder and M. Platzner, "Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations," in *Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2003, pp. 284–287.
- [6] G. Wigley, D. Kearney, and D. Warren, "Introducing reconfigure: An operating system for reconfigurable computing," in *Conference on Field Programmable Logic and Application*, September 2-4 2002.
- [7] X. ping Ling and H. Amano, "Wasmii : a data driven computer on virtual hardware," in *IEEE workshop on FPGAs for custom computing machines*, 1993.
- [8] Y. Shibata and al., "A virtual hardware system on a dynamically reconfigurable logic device," in *IEEE symposium on FPGAs for custom computing machines*, 2000.
- [9] A. DeHon, "Dpga-coupled microprocessors : Commodity ics for the early 21st century," in *IEEE Workshop on FPGAs for custom computing machines*, 1994.
- [10] Xilinx, "Time multiplexed programmable logic device," Patent no.5646545, 1997.
- [11] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Designing an Operating System for a Heterogeneous Reconfigurable SoC," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2003, p. 174a.
- [12] W. Zhao, E. Belhaire, B. Dieny, G. Prenat, and C. Chappert, "Tas-mram based non-volatile fpga logic circuit," in *International Conference on Field-Programmable Technology*, 2007.
- [13] A. Kupriyanov, F. Hannig, D. Kissler, J. Teich, J. Lallet, O. Sentieys, and S. Pillement, "Modeling of interconnection networks in massively parallel processor architectures," in *International Conference on Architecture of Computing Systems (ARCS)*, ser. Lecture Notes in Computer Sciences, vol. 4415, Zurich, Mar. 2007, pp. 268–282.
- [14] S. Garcia, J. Prevotet, and B. Granado, "Hardware task context management for fine grained dynamically reconfigurable architecture," in *Workshop on Design and Architectures for Signal and Image Processing (DASIP)*, 2007.