

OLLAF : a Fine Grained Dynamically Reconfigurable Architecture for OS Support

Samuel GARCIA and Bertrand GRANADO

ENSEA - ETIS

95014 CEREGY, 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 could be used as one possible computational resource. In order to manage efficiently this resource we need a specific OS kernel able to manage such a hardware adaptable architecture. 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 FGDR but a FGDR must also be designed to handle this OS kernel. This article relate our original work in this direction. OLLAF¹, an original FGDR core that we have designed will be presented. A comparison with other methods used today using commercially available FPGA is also presented concerning the particular preemption service.

I. INTRODUCTION

This work takes place in the SMILE project. This project aims at provide a distributed middle layer to efficiently handle the complexity of a tomorrow's RSoC². This system may contains several computing units of different types. It will embed at least one or more General Purpose Processor (GPP), but also dynamically reconfigurable architectures (DRA) at different granularities and especially FGDR³. 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.

By fine grained, we here means an architecture which is reconfigurable at the bit level. A dynamically reconfigurable architecture, using single bit LUT and flipflop, and providing a bit level reconfigurable interconnection matrix, as the one presented here, or basic logic fabric of most commercial FPGA, are examples of FGDR. Those kind of architecture can be adapted to any application more optimally than a coarser grain DRA. This feature make them today the platform of choice when it comes to handle computational tasks in a highly constrained context.

In more general terms FGDR can achieves much better efficiency than GPP does, while offering the same versatility and, potentially, a very close flexibility. The counterpart is that

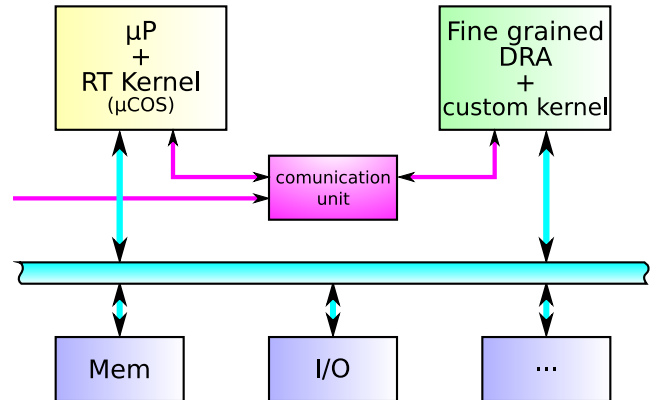


Fig. 1. Basic view of a RSoC in SMILE

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

In our work we make a difference between a FGDR, which is a general term, and a FPGA which, for us, relate to an actual silicon device sold under this designation and which can be used as a FGDR but is actually not designed especially thor that purpose.

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. It then allows 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 FGDR.

This kernel is an adaptation to FGDR of an abstract OS

¹Operating system enabled Low Latency Fgdra

²Reconfigurable System on Chip

³Fine Grained Dynamically Reconfigurable Architecture

⁴Message Passing Interface

model which could be described as follow :

- it manages the execution of a set of task on a given versatile computational resource. More concretely it will run periodically a special algorithm to evaluate where and when to run each tasks. This period is called Tick and is a tradeoff between efficiency and flexibility, a typical value in classical OS is tens of milliseconds.
- it offers an abstracted view of the platform to the task designer. In other terms, each task can be designed without worrying about other tasks and sometimes even about the platform. It then offers a standardized set of services such as communications or synchronizations between tasks.

This model slightly differs from most OS implementation proposed for FPGA management even if the overall idea remain the same.

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 FGDR, but a FGDR must also be designed to support efficiently this OS kernel. This article relate our original works in this direction. The FGDR core that we have designed will be presented as well as a more general view of our approach of a FGDR and its related OS kernel.

This paper is organized as follows. Section 2 discuss of related works in the field of OS for FGDR. Section 3 explains our original FGDR platform proposition named OLLAF. Section 4 discuss more precisely of the context management scheme and its extension to configuration management. Section 5 explains how our architecture can affect different OS services. Section 6 exposes an analytic comparison between OLLAF and other methods used today in terms of preemption overhead and efficiency. Finally, conclusions are drawn in section 7.

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.

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 [12]. 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 parallel 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.

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 splited into several equivalent blocks. This will allow to move a task from a block

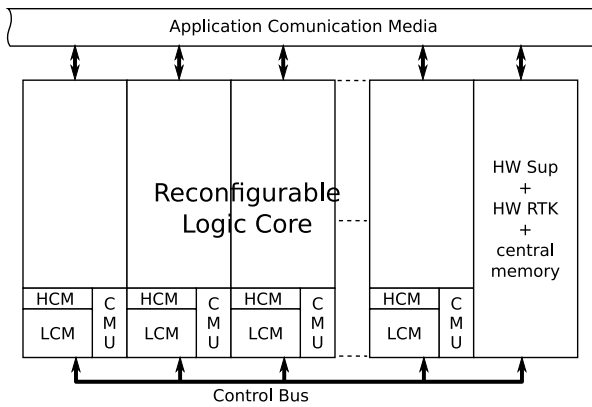


Fig. 2. Global view of the FGDR

to any another block or from a group of blocks to another group of blocks of the same size and the same form factor without any change on the configuration data. The size of those blocks 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.

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 uses an integer number of columns. This topology as been chosen for two reasons. First using a partial reconfiguration by column transforms 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 offers 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 blocks 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 stands a big block called "HW Sup + HW RTK + central memory". This block contain a classic microprocessor which serves 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 communications 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.

C. Logic core overview

In order to make the description of the FGDR core more understandable, we will here split its functionalities between 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 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.

Internal architecture of a LE in the functional point of view can be seen on figure 3. This architecture integrates elements that compose a classic Logic Element of FGDR. If we want to improve functional architecture, it should not change our conclusion on configuration point of view.

A multiplexor based interconnect as been chosen instead of the passing MOS transistor used in most commercial FPGA. In this way we can lower the number of configuration bit required to allow the same connection flexibility. In this last interconnection scheme, the number of configuration bit grow linearly with interconnection possibility while using multiplexor makes it grow as a log2 function.

At first, configuration memory points are modellized as a D flip-flop. This allow us to rapidly apply our works on context management to configuration management. However, configuration and context management remains two separate path, a context swap can be performed without any change

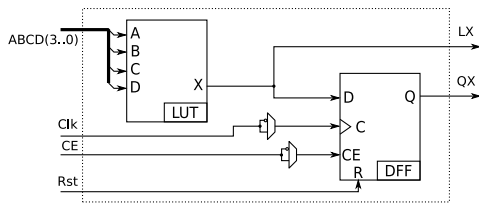


Fig. 3. Functional, task designer point of view of LE

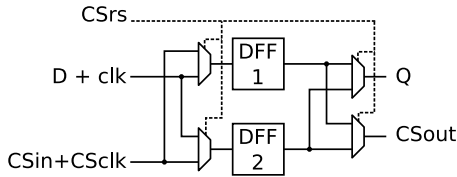


Fig. 4. Dual plane memory point

in configuration. This can be interesting for checkpointing or when more than one instance of the same task running.

IV. CONTEXT MANAGEMENT SCHEME

In [12] we proposed a context management scheme based on a scanpath, a local context memory and the CMU which is a small IP capable of managing automatically context transfer between the scanpath and the local memory. The context management scheme in OLLAF is slightly different in two ways. First, every context management related material is hard wired into the platform. Secondly, we added two more stage in order to even lower preemption overhead and to ensure the consistency of the system.

As context management materials are added at platform level and no more at task level, it needed to be splitted differently. As the Programmable Logic Core is column based, it was then natural to implement context management at columns level. A CMU and a local memory have then been added to each column, and one scanpath is provided for each column's set of flipflops.

In order to lower preemption overhead, our reconfigurable logic core use a double memory plane. Flipflops used in LE are thus replaced with two FF with switching material. Architecture of this double plane FF can be seen on figure 4. *Run* and *scan* are then no more two working modes but two parallel planes which can be swapped as will. With this topology, the context of a task can be shifted in while the previous task is still running and shifted out while the next one is already running. The effective task switching overhead is then taken down to one clock cycle as illustrated in figure 6.

Contexts are transfered by the CMU into Local Context Memories using this hidden scanpath. Because the context of every column can be transfered in parallel, Local Context Memories are placed at column level. It is particularly useful when task use more than one column. Those memories can contain at this stage 10 contexts. They can be seen as local cache memories to optimize access to a bigger memory called the Central Context Repository.

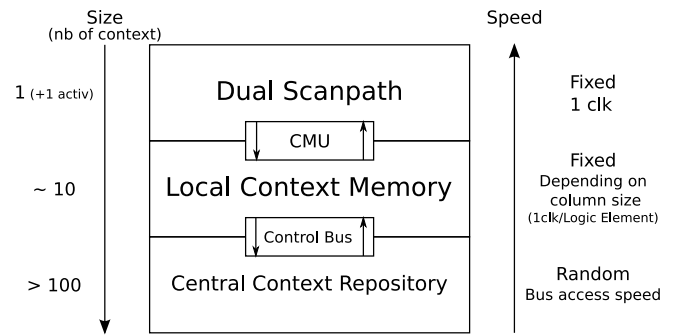


Fig. 5. Context memories hierarchy

The Central Context Repository is a large memory space storing the context of each task instance run by the system. Local Context Memories should then store contexts of tasks who are most likely to be the next to be ran on the corresponding column.

After a preemption of the corresponding task, a context can be stored in more than one LCM in addition to the copy stored in the Central Context Repository. In such situation, care must be taken to ensure the consistency of the task execution. For that purpose, contexts are tagged by the CMU each time a context saving is performed with a version number. The operating system keep tracks of this version number and also increment it each time a context saving is performed. In this way the system can then check for the validity of a context before a context restoration. The system must also try to update the context copy in the CCR as short as possible after a context saving is performed.

Dual Plan Scanpath, Local Context Memory and Central Context Repository form a complex memory hierarchy specially designed to optimize preemption overhead. The same memory scheme is also used for configuration management except configuration do not change during execution so it does not need to be saved and then no versioning control is required here. The programmable logic core use a dual configuration plane equivalent to the Dual Plane Scanpath used for context. Each column has a Hardware Configuration Manager which is a simplified version of the CMU (without saving mechanism). A Local Configuration Memory is provided beside Local Context Memory, the name LCM is used as in figure 3 to relate to both those memories. In the same way, the CCR can refer to Central Context/Configuration Repository.

In best case, preemption overhead can then be bound to one clock cycle.

A scenario of a typical preemption is presented here. In this scenario we consider the case where context and configuration of both task are already stored into the right LCM. 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

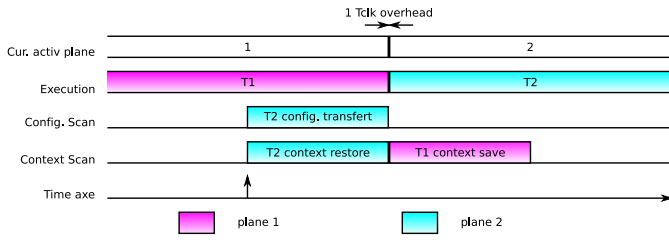


Fig. 6. Typical preemption scenario

- 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
- T1's context is updated as soon as possible in the CCR

This scenario is illustrated in figure 6.

This is the case when both context and configuration of T2 are already stored into LCM. That means that, in order to have this favorable case, we need an anticipated scheduling to manage our Context/Configuration Memories Hierarchy as a smart cache.

V. CONFIGURATION, PREEMPTION AND OS INTERACTION

In previous sections an architectural view of our FGDR has been exposed. In this section, we discuss about the impact of this architecture on OS services. We will here consider the three services 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 an intelligent cache to prefetch task configuration on the columns where it might most probably be placed. In order to do so, an 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 valid 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. PREEMPTION COST COMPARISON

This section present an analytic comparison of preemption efficiency in OLLAF and other solution from past works or literature. We will here consider six methods :

- XIL
a solution based on the xilinx XAPP290 [13] using ICAP to transfer both context and configuration and using the readback bitstream for context extraction.
- Scan
a solution using a simple scanpath for context transfer as described in both [14] and [12], and using ICAP interface for configuration.
- PCS8
is similar to Scan solution but using 8 parallel scanpath as described in [12].
- DPScan
use a dual plane scanpath similar to the one used in OLLAF for context and ICAP for configuration. This method is also studied in [14], referred as a shadow Scan Chain.
- MM
use once again ICAP for configuration and the memory mapped solution proposed in [14].
- OLLAF
this last solution being the use of separate, column distributed, dual plane scanpath for configuration and context as proposed in this article.

In this study we consider two parameter. The preemption overhead H is the cost of a preemption for the system in terms of time. The efficiency of preemption process λ is then $\lambda = 1 - \frac{H}{P}$ with P is the minimum period at which preemption occurs so in our case P is the clock tick of the operating system. In this study we use a typical clock tick of 10ms. In order to focus on the architectural view only all times will be expressed and estimated in number of clock cycle. Assuming a typical clock frequency of 100MHz the OS tick is 10^6tclk . Task sizes will be expressed as n , the number of flipflop used. The time cost of a preemption take into account two context transfers and one configuration transfer.

Analytic expression of H for each case are estimated as follow :

- XIL
In [14] authors estimate that bitstream contain 20 times more data than context related data so the bitstream of a task of size n is approximately $21n$. Assuming that it use a 32bits width access bus, the ICAP interface can transfers 32bits per clock cycle. In the same article, authors estimate that it takes 20clock cycles to extract each context bit from the readback bitstream.

$$H = \frac{21n}{32} + \frac{21n}{32} + 20n \simeq 21.3n \quad (1)$$

	XIL	Scan	PCS8	DPScan	MM	OLLAF
H (tclk)	15188	1897	642	472	492	1
λ	98.4%	99.8%	99.94%	99.95%	99.95%	$1 - 10^{-6} \simeq 100\%$

TABLE I
COMPARISON OF TASK PREEMPTION OVERHEAD AND EFFICIENCY FOR 713FF TASK

	XIL	Scan	PCS8	DPScan	MM	OLLAF
H (tclk)	21.3×10^6	2.66×10^6	900×10^3	660×10^3	690×10^3	1
λ	-2030%	-166%	10%	34%	31%	$\simeq 100\%$

TABLE II
COMPARISON OF TASK PREEMPTION OVERHEAD AND EFFICIENCY FOR A WHOLE 1MFF FGDR

- Scan
Using a simple scanpath for context transfer requires 1 clock cycle per flipflop for each transfer.

$$H = \frac{21n}{32} + 2n \simeq 2.66n \quad (2)$$

- PCS8
Using 8 parallel scanpath it requires 1 clock cycle for 8 flipflops.

$$H = \frac{21n}{32} + \frac{2n}{8} \simeq 0.9n \quad (3)$$

- DPScan
Using a double plane scanpath, the context transfers can be hidden, the cost of those transfer is then always 1 clock cycle.

$$H = \frac{21n}{32} + 1 \simeq 0.66n + 1 \quad (4)$$

- MM
Using 32 bits memory access, this case is similar to the PCS8 but using 32 parallel paths instead of 8.

$$H = \frac{21n}{32} + \frac{2n}{32} \simeq 0.69n \quad (5)$$

- OLLAF
In OLLAF, both context and configuration transfer are hidden so the total cost of the preemption is always 1 clock cycle whatever the size of the task.

$$H = 1 \quad (6)$$

In order to make a concrete case comparison, we will consider two task T1 and T2. We consider a DES56 cryptographic IP that requires 862 flipflops, and a 16tap FIR filter that requires 563 flipflop. Both of those IPs can be found in www.opencores.org. To ease the computation we will consider two task using the average number of flipflop of the two considered IP. So for T1 and T2 we got $n = \frac{862+563}{2} \simeq 713$. Table I show the overhead H and the efficiency λ for each method presented.

Those results show that in this case, using our method leads to a preemption overhead around 500 times smaller than the bests others cases.

If we now consider that not only one task is preempted but the whole FGDR, assuming a 1 Million LE's logic core, estimation of overhead and efficiency for each method are shown in table II. Those results show clearly the benefit of OLLAF platform over actual FPGA concerning preemption. Using actual methods, preemption overhead is linearly dependant on the size of the task. In OLLAF, this overhead do not depends on the size of the task and is always of only one clock cycle.

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 in any case 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.

VII. CONCLUSION AND PERSPECTIVES

A global view of OLLAF, a 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.

In this paper we presented in more details our context management scheme and its extention to configuration management. It has been shown that this scheme permit a far better preemption efficiency than other methods in use today.

Today, the reconfigurable logic core have been designed and is being tested by several simulations. The rest of the FGDR is also in progress. The dedicated custom OS services are 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 (ICESS), 2005, pp. 116–121.
- [3] H. Simmler, L. Levinson, and R. Männer, "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 reconfirme: 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 virtuel 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 cmputing 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] 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.
- [13] Xilinx, "Two flows for partial reconfiguration: Module based or difference based," Xilinx, Application Note, 2004, application Note: Virtex, Virtex-E, Virtex-II, Virtex-II Pro Families XAPP290 (v1.2) September 9, 2004.
- [14] D. Koch, C. Haubelt, and J. Teich, "Efficient hardware checkpointing - concepts, overhead analysis, and implementation," in Field Programmable Logic and its Applications (FPL), 2007.