OLLAF : a Dual Plane Reconfigurable Architecture for OS Support

Samuel Garcia, Bertrand Granado

OLLAF : a Dual Plane Reconfigurable Architecture for OS Support

Samuel GARCIA and Bertrand GRANADO
ENSEA - ETIS
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 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 FGDRA[1] but a FGDRA must also be designed to handle this OS kernel. This article relate our original work in this direction. OLLAF[2], an original FGDRA 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 (see figure 1). This project aims at provide a distributed middle layer to efficiently handle the complexity of a tomorrow's RSoC[3]. 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 especialy FGDRA. 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 electronic 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 FGDRA. 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 FGDRA can achieve much better efficiency than GPP does, while offering the same versatility and, potentially, a very close flexibility. The counterpart is that

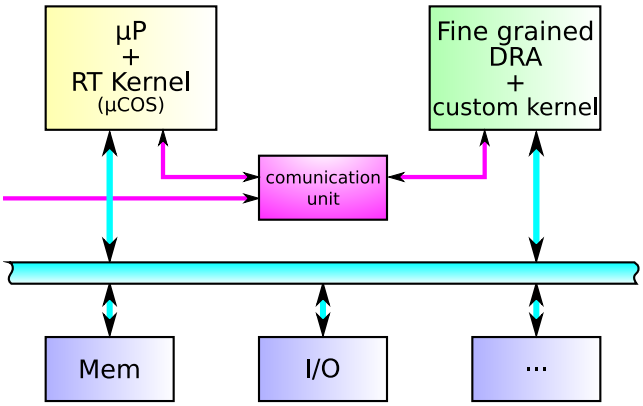

Fig. 1. Basic view of a RSoC in SMILE

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 FGDRA.

In our work we make a difference between a FGDRA, which is a general term, and a FPGA which, for us, relates 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. It then allows to take into account every specificities of each computing unit. A message passing communication scheme, based on MPI[4], ensure a consistent operation of the whole system. In this frame of mind, we developed a dedicated real time kernel for a FGDRA.

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

[1]Fine Grained Dynamically Reconfigurable Architecture
[2]Operating system enabled Low LAtency Fgdra
[3]Reconfigurable System on Chip
[4]Message Passing Interface

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

This paper is organized as follows. Section 2 discusses of related works in the field of OS for FGDRA. Section 3 explains our original FGDRA platform proposition named OLLAF. Section 4 discusses more precisely of the context management scheme and its extension to configuration management. Section 5 exposes a particular case study that covers the most common memory transfer case in OLLAF. Section 6 compares the transfer costs in OLLAF with the classical approach using this case study. Finally, conclusions are drawn in section 7.

II. Related work

A. OS for FGDRA

Several research have been led in the field of OS for FGDRA[3], [4], [5], [6]. All those studies present an OS more or less customized to enable specific FGDRA 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.

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 has 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]. In [11], authors also study the use of a configuration cache, this feature is provided to lower costly external transfers, witch is not mutch a problem in the case of a SoC. This paper hoever, show the advantages of coupling configuration caches, partial reconfiguration and multiple configuration planes.

More recently, in [12], 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 FGDRA.

B. previous work

Our first work on OS for FGDRA was related to preemption of hardware task on FPGA[13]. 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 introduces 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 FGDRA with OS support

We have designed a FGDRA 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 spends 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 FGDRA 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 works 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 FGDRA 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 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 tradeof 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 FGDRA 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 concerns the whole heterogeneous platform, not only the particular FGDRA computing unit. The second case is when two, or more, tasks run on top of the same FGDRA communicate together. This communication channel should remain the same wherever the task is placed on the FGDRA reconfigurable core and whatever state those

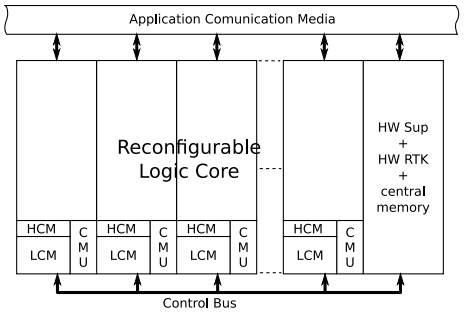

Fig. 2. Global view of the FGDRA

tasks are (running, pending, waiting, ...). That means that the FGDRA 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 FGDRA.

B. Proposed solutions

Figure 2 shows a global view of OLLAF, our original FGDRA designed to support OS sevices as they have just been specified.

In the center stand the reconfigurable logic core of the FGDRA. 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 has 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 are 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 contains a classic microprocessor which serves as a hardware supervisor. It runs a custom real time kernel specially adapted to handle FGDRA 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 configurations and eventual contexts of every task that may run on the FGDRA. 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 FGDRA they would be connected with this communication medium in the same way reconfigurable columns are.

C. configuration, preemption and OS interaction

In previous sections an architectural view of our FGDRA 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 FGDRA.

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 means 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 implies 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 exists more than one valid context for each task in the entire FGDRA. 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 FGDRA).

IV. Context Management Scheme

In [13] 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 stages

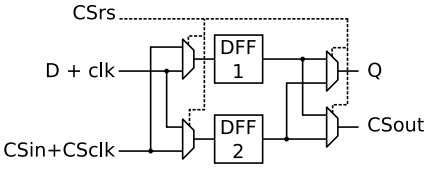

Fig. 3. Dual plane configuration memory

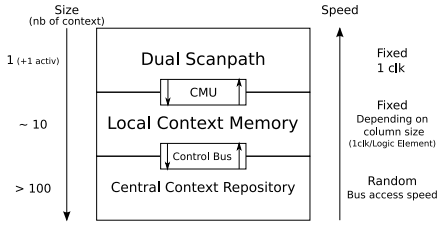

Fig. 4. Context memories hierarchy

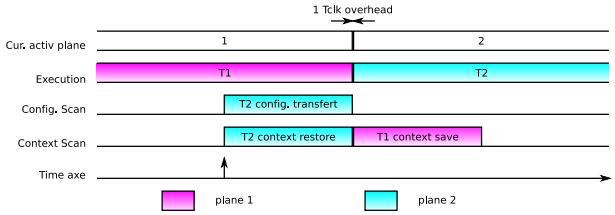

Fig. 5. Typical preemption scenario

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 splited differently. As the Programable 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 uses 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 3. *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 5.

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 usefull when a task uses 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.

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 keeps track of this version number and also increments 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 that a configuration does 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 2 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 decides 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
- T1's context is updated as soon as possible in the CCR

This scenario is illustrated in figure 5.

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.

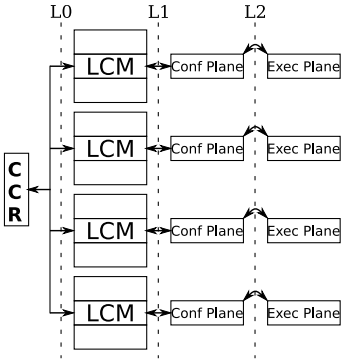

Fig. 6. Memory view of the considered implementation of OLLAF

V. Case study

We will here expose the execution of a given already scheduled set of tasks. We consider here a particular implementation of the OLLAF architecture including 4 columns of 1024 logic elements each (4x256LE per columns), a 32bits control bus width, and local configuration/context memories (LCM) size of three task so three configuration and three context. This mean a configuration bitstream size of 87Kbits and a context of 1Kbit per column. LCMs would then be 264Kbits memories.

As stated earlier OLLAF architecture provide three level of configuration memory in addition to the working configuration plane. Those memory levels are respectively the CCR, LCMs and the second configuration plane (the plane not currently working). Transfer from the CCR to an LCM using the control bus will be referred at as L0 transfer, those from an LCM to the CCR as L0'. Transfer from LCM to the non working configuration plane will be L1 and the backward L1'. Swapping between the two configuration plane could alos be modelized as memory transfer referred at by L2, note that this particular transfer always occurs in both directions at the same time. This memory view of the platform is shown on figure 6.

In this study, as the architecture is all synchronous, all times will be considered in an integer number of clock period, referred at by Tclk. Task scheduling will be dictated by the operating system clock also called Tick, which is not to be confused with the logical clock above-mentioned. The period of this operating system clock have usually a typical value comprised between 1ms and 100ms. In this study we concider a random logical clock frequency of 100MHz and a Tick period of 5ms as it is the default Tick period used in μC/OS[14], a well known real time kernel that we intend to use in our works. That mean a Tick period of 500K Tclk. Scheduling can also happend in the case where an interuption occure, but this case is not covered in this study.

Table I give transfer times at the differents stages for the considered implementation of the platform.

The considered set of tasks consists on five tasks, T1, T2, T3, T4 and T5, we consider that they have already been scheduled as shown on figure7 and table II. Note that T1 and T1' are two different instance of the same task, that mean they share the same configuration but not the same context.

	l0	l0'	l1	l1'	l2
Transfer time (tclk)	2816	32	1024	1024	1

TABLE I

Transfer times for 4x256 column based OLLAF platform

	T1	T1'	T2	T3	T4	T5
Size (Column)	1	1	2	1	4	1
Duration (Tick)	2,33	2	5	3	1	2
Begin time (Tick)	0	6	0	2	3	7

TABLE II

Considered tasks descriptions

When a task execution in asked on a particular columns, the context and configuration of the task must be transferred to one configuration plane of this column and then the configuration planes must be swapped in order to actually run the task. If another task was running on this column it will be preempted. Depending on the place where the context and configuration data of the task are already present, and on the state of the different memory stages, there can be several different cases of transfers prior of execution. As an example, a very unfavorable case is when we want to run a task on a columns of which the LCM contain three other context that has not already been saved in the CCR, we then need to save at least one of those contexts before we can write into the considered LCM and then transfer the configuration of the task we want to run into one of the memory planes. Memory transfers in this case will lead to a minimal latency of 4865Tclk. Another example is when both context and configuration of the task are still in the second configuration plane, the latency could then be as low as one clock cycle. Those task and this scheduling are chosen to meet the most commons of those cases given the particular implementation considered.

VI. Results

Using the sequence and parameters given in previous section, we here study the transfers cost in terms of time. We compare this cost using three differsnt platforms. The first (Shared Bus) is a platform using a simple 32bit shared bus as it is the case when using a classical FGDRA platform such as a Xilinx VirtexIIpro with Icap interface. The second (OLLAF

	Shared Bus	OLLAF simple	OLLAF
Total Transfer time (tclk)	36608	36897	9509

TABLE III

Transfer times comparison for the first execution

	Shared Bus	OLLAF simple	OLLAF
Total Transfer time (tclk)	36608	11846	1062

TABLE IV

Transfer times comparison for periodic execution of the sequence

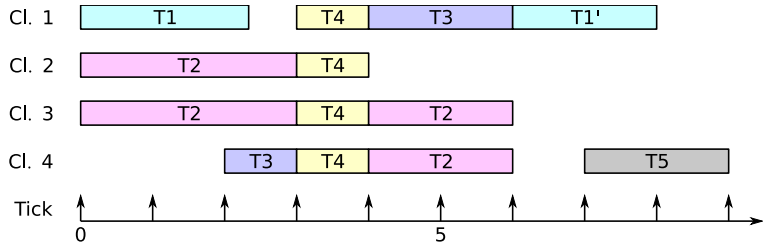

Fig. 7. Considered tasks and scheduling

simple) is a simple plane version of OLLAF where transfers times can not be masked by the execution of another task. And the last (OLLAF) is the OLLAF architecture as described earlier in this paper. As we want to focus this study on the memory transfers, we will here consider that no anticipations are made. The total transfer times during one execution of this sequence using those three platforms are shown in Table III.

Those results shows that the total transfer time for OLLAF-simple is quiet the same than for the Shared bus solution, this can be explained as the gain due to parallelisme of some transfers is weighed by the fact that transfers are more numerous. But it also shows being able to overlap execution and context transfers, as with OLLAF, can achieve a gain close to x4 in such a sequence.

If we now concider this sequence as periodical, results in Table III are now only true for the first execution of the sequence. Table IV show results for every next iteration of the sequence.

Those results, regarding of OLLAFsimple versus Shared bus, show the utility of a cache stage, which is no more to be demonstrated. We can note that the result in OLLAF is not as low as we could at first expect due to a cache miss. The interesting result is that in the case of OLLAF, it permits a gain of 34.5 compared with the classical approach in spite of this cache miss.

If we were to concider either a big platform or short Tick period, this would have a big impact on the efficiency of the platform.

We should also note that with a very simple anticipation mechanism we could avoid this cache miss, resulting in a transfer cost of only 6Tclk using OLLAF or a gain over 6000 comparing to the classical approach.

VII. Conclusion and perspectives

A global view of OLLAF, a FGDRA that enhance OS service support has been presented. 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 showed a case study that demonstrate that thanks to a dedicated configuration and context memory hierarchy, the OLLAF architecture can perform a greater efficiency than the one performed using a traditional commercial FPGA.

This paper shows result of the preliminary study of the project, it demonstrate the efficiency of the methodes used in terms of time overhead. Further work will now be led in three directions. First, the development of a generic VHDL description of the OLLAF platform should permit to study area cost of those methodes. Secondly the OS itself will be further studied and more particularly the scheduling and ressources management algorithmes, we should as an exemple try to study in more details an anticipated scheduling mechanism in order to be able to prefetch configurations and context and then to bring the best out of the OLLAF architecture. And last, the Application Comunication Media should also be further studied, including data exchanges, application memories and I/O management.

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 reconfigme: 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 symposiunm 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] Z. Li, K. Compton, and S. Hauck, "Configuration caching techniques for fpga," in IEEE Symposium onFPGA for Custom Computing Machines (FCCM), 2000.

[12] 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.

[13] 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.

[14] J. J. Labrosse, MicroC/OS-II The Real-Time Kernel. CMPBooks, 2002.