Dynamic Adaptation of Hardware-Software Scheduling for Reconfigurable System-on-Chip

Fakhreddine Ghaffari ; Benoit Miramond ; François Verdier
ETIS Laboratory –UMR 8051 – ENSEA University of Cergy Pontoise – CNRS
6, Av. Du Ponceau,BP 44, F95014 Cergy-Pontoise cedex, France
EMAIL: {fakhreddine.ghaffari, miramond,verdier}@ensea.fr

Abstract

This paper presents an efficient run-time Hardware/Software scheduling approach. This scheduling heuristic consists in mapping on-line the different tasks of a highly dynamic application in such a way that the total execution time is minimized.Our approach takes advantage of the reconfiguration property of the considered architecture to adapt processing to the system dynamics. We compare our heuristic with another similar approach.

We present the results of our scheduling method on an image processing application. Our experiments include simulation and synthesis results on a Virtex2P based platform. This results show a better performance against existing methods.

1. Introduction

One of the main steps of the HW/SW codesign of a mixed electronic system (Software and Hardware) is the scheduling of the application tasks on the processing elements (PE) of the platform. The scheduling of an application formed by N tasks on M target processing units consists in finding the realizable partitioning in which the N tasks are launched onto their corresponding M units and an ordering on each PE for which the total execution time of the application meets the real time constraints. This problem of multiprocessor scheduling is known to be NP-complete [1] that's why we propose a heuristic approach.

Many applications, in particular in image processing (e.g. an intelligent embedded camera), have dependent data execution times according to the nature of the input to be processed. In this kind of application, the implementation is often stressed by real time constraints, which demand adaptive computation capabilities. In this case, according to the nature of the input data, the system must adapt its behaviour to the dynamics of the evolution of the data and continue to meet the variable needs of required calculation (in quantity and/or in type). Examples of applications where the processing need changes in quantity (the computation load is variable) comes from the intelligent image processing where the duration of the treatments can depend on the number of the objects in the image (motion detection, tracking…) or of the number of interest areas (contours detection, labelling…).

We can quote also the use in run-time of different filters according to the texture of the processed image (here it is the type of processing which is variable). Another example of the dynamic applications is video encoding where the run-length encoding (RLE) of frames depends on the information within frames.

For these dynamic applications, many implementation ways are possible. In this paper we consider an intelligent embedded camera for which we propose a new design approach compared to classical worst case implementations.

Our method consists in evaluating on-line the application context and adapting its implementation onto the different targeted processing units by launching a run-time partitioning algorithm. The on-line modification of the partitioning result can also be a solution of fault tolerance; by affecting in run time the tasks of the fault target unit on others operational targets [15]. This induces also to revise the scheduling strategy. In that context, the choice of the implementation of the scheduler is of major importance and depends on the heuristic complexity. Indeed, with our method the decisions taken on-line by our scheduler can be very time consuming. A software implementation of the proposed scheduling strategies will then delay the application tasks. For this reason, we propose in this work a hardware implementation for our scheduling heuristic.With this implementation, the scheduler takes only few clock cycles. So we can easily call the scheduler at run-time without penalty on the

978-0-7695-3180-9/08 $25.00 © 2008 IEEE
DOI 10.1109/RSP.2008.28

total execution time of the application. The implementation of our scheduler allows the system to adapt itself to the application context in real-time. We have simulated and synthesized our scheduler by targeting a FPGA (Xilinx Virtex) platform. We have tested the scheduling technique on an image processing application implemented onto a heterogeneous target architecture composed of two processors coupled with a configurable logic (FPGA).
The remainder of this paper is organized as follows. Section 2 presents related works on scheduling approaches. Section 3 introduces the framework of our scheduling problem. Section 4 presents the proposed approach. Section 5 shows the experimental results and finally the section 6 concludes this paper.

2. Related work

The field of study which tries to find an execution order for a set of tasks that meets system design objectives (e.g. minimize the total application execution time) has been widely covered in the literature. In [2] the problem of HW/SW scheduling for system-on-chip platforms with dynamically reconfigurable logic architecture is exhaustively studied. Moreover several works deal with scheduling algorithm implemented in hardware [3], [4], [5]. Scheduling in such systems is based on priorities. Therefore, an obvious solution is to implement priorities queues. Many hardware architectures for the queues have been proposed: binary tree comparators, FIFO queues plus a priority encoder, and a systolic array priority queue [3]. Nevertheless, all these approaches are based on a fixed priority static scheduling technique. Moreover most of the hardware proposed approaches addresses the implementation of only one scheduling algorithm (e.g., Earliest Deadline First) [5]. Hence they are inefficient and not appropriate for systems where the required scheduling behavior alters during runtime. Also, system performance for tasks with data dependent execution times should be improved by using dynamic schedulers instead of static (at compile time) scheduling techniques [6], [14].

The idea of dynamic partitioning/scheduling is based on the dynamic reconfiguration of the target architecture. Increasingly FPGA [16, 17] offer very attractive reconfiguration capabilities: partial or total, static or dynamic.

The reconfiguration latency of dynamically reconfigurable devices represents a major problem that must not be neglected. Several references can be found addressing temporal partitioning for reconfiguration latency minimization [7]. Moreover, configuration prefetching techniques are used to minimize reconfiguration overhead. A similar technique to lighten this overhead is developed in [8] and is integrated into an existing scheduling environment. In this paper, we focus only on the scheduling strategy and we assume that those reconfiguration aspects are taken into account in the HW/SW partitioning step (in the decision of task implementation). Furthermore we addressed this last step in our previous works [10].

3. Problem definition

3.1. Target architecture

The target architecture is depicted in Figure 1. It is a heterogeneous architecture, which contains two software processing units: a Master Processor and a Slave Processor. The platform also contains a hardware processing unit: RCU (Reconfigurable Computing Unit) and shared memory resources. The software processing units are Von-Neumann mono-processing systems and execute only a single task at a time.
Each hardware task (implemented on the RCU) occupies a tile on the reconfigurable area [9]. The size of the tile is the same for all the tasks to facilitate the placement and routing of the RCU. We choose for example the tile size of the task which uses the maximum of resources on the RCU (we designate by "resource" here the Logic Element used by the RCU to map any task).
The RCU unit can be reconfigured partially or totally. Each hardware task is represented by a partial bitstream. All the bitstreams are memorized in the contexts memory (the shared memory between the processors and the RCU in the figure 1). These bitstreams will be loaded in the RCU to reconfigure the FPGA according to run-time partitioning results. The HW/SW partitioning result can change at run-time according to temporal characteristics of tasks. In [10] we proposed a HW/SW partitioning approach based on HW→SW and SW→HW tasks migrations. The theory of tasks migrations consists in accelerating the task(s) which become critical by modifying their implementations from software units to hardware units and to decelerate the tasks which become noncritical by returning them to the software units.
After each new HW/SW partitioning result, the scheduler must provide an evaluation for this solution by providing the corresponding total execution time. Thus it presents a real time constraint since it will be launched at run-time. With this approach of dynamic

partitioning/scheduling the target architecture will be very flexible. It can self-adapt even with very dynamic applications.

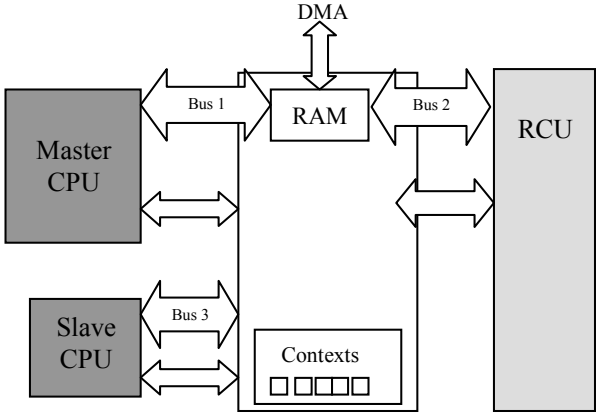


Figure 1: The target architecture

3.2. Application model

The considered applications are data flow oriented applications such as image processing, audio processing or video processing etc. To model this kind of applications we consider a DFG: Data Flow Graph (an example is depicted in figure 2) which is a directed acyclic graph where nodes are processing functions and edges describe communication between tasks (data dependencies between tasks). The size of the DFG depends on the functional partitioning of the application and then on the number of tasks and edges. We can notice that the structure of the DFG has a great effect on the execution time of the scheduling operations. A low granularity DFG makes the system easy to be predictable because tasks execution time does not vary considerably, thus limiting timing constraints violation. On the other hand, for a very low granularity DFG, the number of tasks in a DFG of great size explodes, and the communications between tasks become unmanageable.

Each node of the DFG represents a specific task in the application. For each task there can be up to three different implementations: Hardware implementations (HW) placed in the FPGA, Software implementations running on the master processor (MS), and another Software implementations running on the slave processor (SL).

Each node of the figure 2 is annotated with two informations: one about the implementation (MS or SL or HW) and the other is the execution time of the task.

Similarly each edge is annotated with the communication time between two nodes (two tasks).

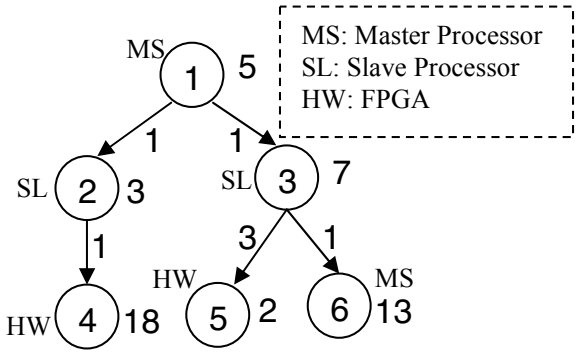


Figure 2: An Example of DFG

A task of the DFG is characterized by the following parameters: Texe (Execution time), Impl (Implementation on the RCU or on the master processor or on the slave processor), Nbpred (number of predecessor tasks) and the Nbsucc (number of successor tasks). All the tasks of a DFG are thus modeled identically and the only real time constraint is on the total execution time. At each scheduler invocation, this total execution time corresponds to the longest path in the mapped task graph. It then depends both on the application partitioning and on the chosen total order on processors.

4. Proposed approach

The applications are periodicals. In one period, all the tasks of the DFG must be executed. In the image processing the period is the execution time needed to process one image. The scheduling occurs only at the end of the execution of all tasks. Hence the result of partitioning/scheduling will be applied on the next period (next image). Our run-time scheduling policy is dynamic since the execution order of application tasks is decided at run-time. For the tasks implemented on the RCU, the only condition for launching their execution is the satisfaction of all data dependencies. A task may begin execution only after all its incoming edges have been executed. For the tasks implemented on the software processors, the conditions for launching are (1) the satisfaction of all data dependencies and (2) the discharge of the software unit. Hereby the task can have four different states: waiting, running, ready and stopped. The task is in the waiting state when it waits the end of execution of one or several predecessors. When a processing unit has finished the execution of a task, new tasks may become

ready for execution if all their dependencies have been completed. The task can be stopped in the case of preemption or after finishing its execution. The states of the processing units in our target architecture are: execution state, reconfiguration state or idle state. In the following, we will explain the principle of our approach as well as a hardware implementation of the proposed HW/SW scheduler.

4.1. Description of the scheduling algorithm

As explained on the figure 3, the basic idea of our heuristic of scheduling is to take decision of tasks priorities according to three criteria. The first criterion is the ASAP (As Soon As Possible) time. The task which has the shortest ASAP date will be launched first. The second criterion is the urgency time: the task which has the maximum of urgency will have priority to be launched before the others. This new criterion is based on the nature of the successors of the task. The urgency criterion is employed only if there is equality of the first criterion for at least two tasks. If there is still equality of this second criterion we compare the last criterion which is execution time of the tasks.

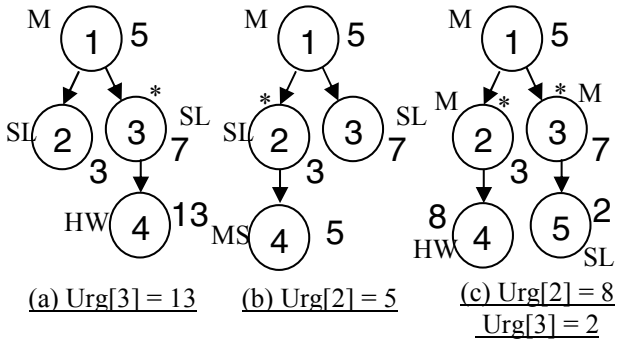
For all Software tasks do
- Compute ASAP
  - ✓ Task with minimum ASAP will be chosen
- If (Equality of ASAP)
- Compute Urgency
  - ✓ Task with maximum urgency will be chosen
- If (Equality of Urgency)
- Compare Execution time
  - ✓ Task with maximum execution time will be chosen

Figure 3: Principle of our scheduling policy

We choose the task which has the upper execution time to launch first. We use these criteria to choose between two or several software tasks (on the Master or on the Slave) for running.

The urgency criterion

The urgency criterion is based on the implementation of tasks and the implementations of theirs successors. A task is considered as urgent when it is implemented on the software unit (Master or slave) and has one or more successor tasks implemented on other units (hardware unit or software unit). The figure 4 shows three examples of DFG. In figure 4 (a), task 3 is implemented on the Slave processor and it is followed by task 4 which is implemented on the RCU. Thus the urgency of task 3 is the execution time of its successor (Urg (3) = 13). In the example (b) it is the task 2 which is followed by the task 4 implemented on a different unit (on the Master processor). In the last example (c) both task 2 and 3 are urgent but the task 2 is more urgent than 3 since its successor has an execution time upper than the execution time of the successor of task 3. When a task has several successors with different implementations, the urgency is the maximum of execution times of the successors.



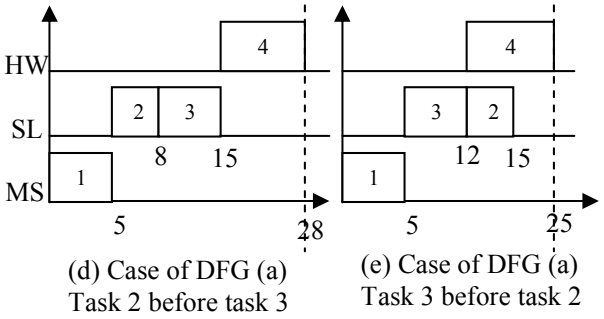


Figure 4: Case examples of urgency computing

In general case, when the direct successor of the task A has the same implementation as A and has a successor with a different implementation, then this last feedbacks the urgency to task A. We show the scheduling result for the case (a) when we respect the urgency criterion in the figure 4 (d) and for the other case in figure 4(e). We can notice for all the examples of DFG in figure 4 that the urgency criterion makes a best choice to obtain a minimum total execution time. The third criterion (the execution time) is an arbitrary choice and has very rarely impact on the total execution time. This scheduling strategy needs an on-line computation of several criterions for all software tasks in the DFG. A software implementation of this strategy is thus incompatible with real time constraints. We describe in the following an optimized hardware implementation of our scheduler.

4.2. Hardware Scheduler architecture

In this section, we describe the architecture of our scheduler. This architecture is shown in figure 5 for a DFG example of three tasks. It is divided in four main parts: (1) the DFG_IP_Sched, (2) The DFG_Update, (3) The MS_Manager and (4) the Slave_Manager.

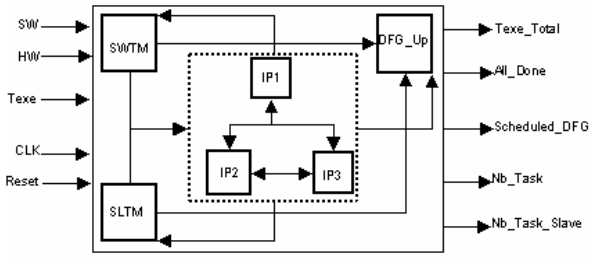


Figure 5: The scheduler architecture for a DFG of three tasks

The basic idea of this hardware architecture is to maximize the parallelism of processing tasks. So, at the most, we can schedule all the tasks of the DFG in parallel for infinite resources architecture. We associate to the application DFG a modified graph with the same structure composed of the IP nodes (each IP represents a task). Therefore in the best case we could schedule all the tasks in the DFG in only one clock cycle (if all tasks can be executed in parallel). To parallelize also the management of the software execution times, we associate for each software unit a hardware module (Master Task Manager and the Slave Task Manager) which manage the order of the tasks executions and compute the processor time for each one. In the following paragraphs, we will detail each part of this architecture.

The DFG_IP_Sched block

In this architecture there are N components (N is the number of tasks in the application).

For each task we associate an IP component which computes the intrinsic characteristics of this task (urgency, ASAP, Ready state…). It also computes the total execution time for the entire graph.

If the task is implemented on the RCU it will be launched as soon as all its predecessors will be done. For the software tasks (on the master or on the slave) the scheduling will take one clock cycle per task. Thus the computing time of the hardware scheduler only depends on the result of the HW/SW partitioning.

The DFG_Update block

When a DFG is scheduled the result modifies the DFG into a new structure. The DFG_Update block generates new edges (dependences between tasks) after scheduling in objective to give a total order of execution on each computing unit.

We represent dependences between tasks in the DFG by a matrix where the rows represent the successors and the columns represent the predecessors.

After scheduling, the resulting matrix is the update of the original one. It contains more dependences than this later. This is the role of the DFG_Update block.

The MS_Manager block

The objective of this module is to schedule the software tasks according to the algorithm given above. The input signal represents the ASAP times of all the tasks. The tasks which have the two criteria (ASAP and urgency) will be represented by a Tasks_Ready signal. A Task_Scheduled signal determines the only software task which will be scheduled. With this signal, it is possible to choose the good value of signal TEXE_SW and to give the new value of the SW_Total_Time signal thereafter. A single clock cycle is necessary to schedule a single software task. By analogy the Slave_Manager block has the same role as the SW_Manager block. From scheduling point of view there is no difference between the two processors

5. Experimental results

With the idea to cover a wide range of applications, we have tested our approach on a complex DFG which contains different classical structures (Fork, join, sequential). This DFG is depicted in Figure 6. It contains twenty tasks. Each task can be implemented on the Software computation unit (Master or Slave processor) or on the Reconfigurable RCU. The original DFG is the model of an image processing application: motion detection on a fixed image background. This application is composed of 10 sequential tasks (from ID 1 to ID 10). We added 10 others virtual tasks to obtain a complex DFG containing the different possible parallel structures. This type of parallel program paradigm (Fork, join …) arises in many application areas. All the execution times are in milliseconds (ms).

In order to test the presented scheduling approach, we have performed a large number of experiments. Several scenarios of HW/SW partitioning results are analyzed.

From figure 7, it may be concluded that when the result of partitioning changes at runtime, then the computation time needed for our scheduler to schedule all the DFG tasks is widely dependent on this modification of tasks implementations. So:

(1) There is a great impact of the partitioning result and the DFG structure on the scheduler computation time. (2) The longest sequential sequence of tasks

corresponds to the case where all tasks are on the Software (each task takes one clock cycle). This case corresponds to the maximum of schedule computation time. (3) The minimum schedule computation time depends on the DFG structure.

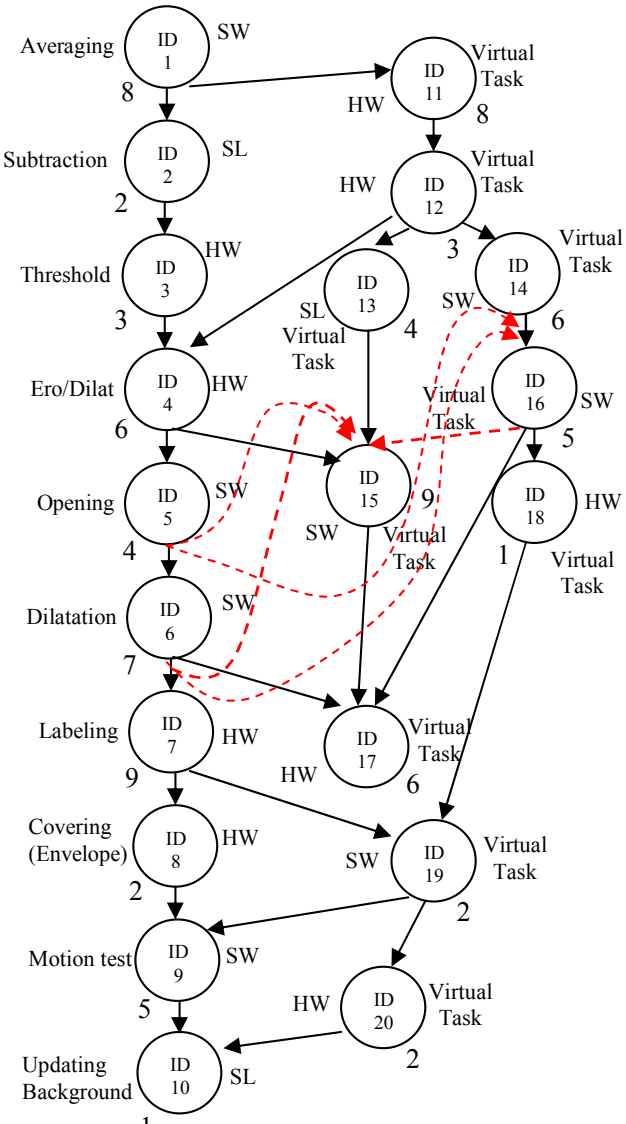


Figure 6 : DFG application

The longest sequential sequence of tasks when all tasks are on the hardware (each task takes one clock cycle). In our case (Figure 6) this sequence is formed by 10 tasks, so the minimum schedule computation time is equal to 10 clock cycles.

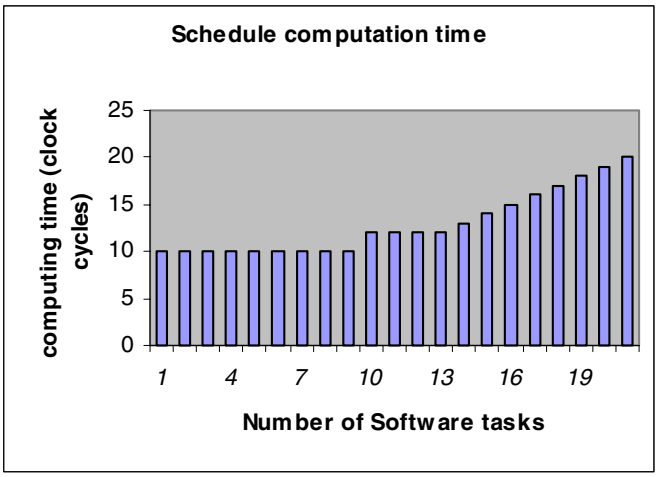


Figure 7: Variation of scheduling computation time according to tasks implementations

5.1. Comparison results

Throughout our experiments, we compared the result of our scheduler with the one given by the HCP algorithm (Heterogeneous Critical Path) developed by Madsen [12]. This algorithm represents an approach of scheduling on a heterogeneous multiprocessor architecture. It starts with the calculation of priorities for each task associated with a processor. A task is chosen depending on the length of its critical path (CPL). The task which has the largest minimum CPL will have the highest priority. We compared with this method because it is shown better than several other approaches (MD, MCP, PC ...) [13]. Our scheduling method provides consistent results significantly better than HCP method. For example the HCP method returns a total execution time of the DFG shown in figure 6 equal to 69ms whereas our method returns only 58ms for the same DFG.

5.2. Synthesis Results

We have synthesized our scheduler architecture with a FPGA target platform (Virtex2P, device XC2VP100) [11] for the RCU of figure 1. The table below (figure 8) shows the device utilization summary. We noticed that for this complex DFG, our scheduler use only 12% of the device slices which is reasonable.

Logic Utilization	Used	Available	Utilization
Number of Slices	5351	44096	12%
Number of Slice Flip Flops	813	88192	0%
Number of 4 input LUTs	10074	88192	11%
Number of bonded IOBs	673	1164	57%

Figure 8: Device utilization summary after synthesis

6. Conclusions

In this paper, we presented a complete runtime hardware-software scheduling approach. Results of our experiments show the efficiency of the adaptation of the scheduling to a dynamic change of the partitioning that can be due to a new mode of a dynamic application or to fault detection. As developed in this paper, a dynamic HW/SW Scheduling approach has many advantages over static traditional approaches. In addition to that, the efficiency of our hardware implementation gives to our scheduler a minimal overhead in on line execution context. Our future works consist in integrating our scheduling approach among the services of an RTOS for dynamically reconfigurable systems.

7. References

[1] Michael Garey and David Johnson: Computers and Intractability - A Guide to the Theory of NP-completeness; Freeman, 1979.

[2] J. Noguera and R. M. Badia. Dynamic runtime hw/sw scheduling techniques for reconfigurable architectures. In CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign, pages 205–210, New York,NY, USA, 2002. ACM Press.

[3] S. Moon, J. Rexford and K. Shin, "Scalable hardware priority queue architectures for high-speed packet switches," IEEE Transactions on Computer, vol. 49, no.11, pp.1215 – 1227, November 2000.

[4] D. Picker and R. Fellman, "A VLSI priority packet queue with inheritance and overwrite," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 3 no. 2, pp. 245–253, June 1995.

[5] B. Kim and K. Shin, "Scalable hardware earliest deadline-first scheduler for ATM switching networks," Proceedings of the Real-time Systems Symposium, pp. 210-218, December 1997.

[6] S. Fekete, J. van der Veen, J. Angermeier, D. Göhringer, M. Majer and J. Teich. Scheduling and communication-aware mapping of HW-SW modules for dynamically and partially reconfigurable SoC architectures. In Proceedings of the Dynamically Reconfigurable Systems Workshop (DRS 2007), Zürich, Switzerland, March 15, 2007.

[7] K. Puma, D. Bhatia, "Temporal Partitioning and Scheduling Data Flow Graphs for Re-configurable Computers", IEEE Trans. on Computers, vol 48, No. 6. June 1999.

[8] J. Resano and D. Mozos. Specific scheduling support to minimize the reconfiguration overhead of dynamically reconfigurable hardware. In DAC '04: Proceedings of the 41st annual conference on Design automation, pages 119–124, New York, NY, USA, 2004. ACM Press.

[9] J-Y. Mignolet, V. Nollet, P.Coene, D.Verkest, S.Vernalde, R. Lauwereins "Infrastructure for Design and management of Relocatable Tasks in a Heterogeneous Reconfigurable System-on-chip". In Proc. of the DATE 2003 Conference. Messe Munich, Germany March 3-7, 2003.

[10] Fakhreddine Ghaffari, Michel Auguin, Mohamed Abid, and Maher Ben Jemaa, " Dynamic and On-Line Design Space Exploration for Reconfigurable Architectures", P. Stenstr¨om (Ed.): Transactions on HiPEAC I, LNCS 4050, pp. 179–193, 2007. c_Springer-Verlag Berlin Heidelberg 2007

[11] Virtex II Pro, Xilinx Corp., http://www.xilinx.com.

[12] P. Bjorn Jorgensen, J. Madsen, "Critical path driven cosynthesis for heterogeneous target architectures" 5th Inter. Workshop on Hardware/Software Codesign (Codes/CASHE'97) March 24-26, 1997.

[13] YU-Kwong Kwok and Ishfaq Ahmad "Static Scheduling for allocating Directed Task Graphs to Multiprocessors" ACM Computing Surveys, vol, 31, No. 4, Decembre 1999.

[14] B. Miramond and J-M. Delosme. Design Space Exploration for Dynamically Reconfigurable Architectures. DATE 2005, 366-371, 2005.

[15] Christian Haubelt, Dirk Koch, Jürgen Teich: Basic OS Support for Distributed Reconfigurable Hardware. SAMOS 2004: 30-38.

[16] Altera Corp., http://www.altera.com.

[17] Xilinx Corp., http://www.xilinx.com.