Online Scheduling for Block-partitioned Reconfigurable Devices

Herbert Walder and Marco Platzner
Computer Engineering and Networks Lab
Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
walder@tik.ee.ethz.ch

Abstract

This paper presents our work toward an operating system that manages the resources of a reconfigurable device in a multitasking manner. We propose an online scheduling system that allocates tasks to a block-partitioned reconfigurable device. The blocks are statically-fixed but can have different widths, which allows to match the computational resources with the task requirements. We implement several non-preemptive and preemptive schedulers as well as different placement strategies. Finally, we present a simulation environment that allows to experimentally investigate the effects of specific partitioning, placement, and scheduling methods.

1 Introduction

Reconfigurable computers map algorithms to specialized hardware circuits which are configured and executed on, for example, SRAM-based Field-Programmable Gate Arrays (FPGAs). These computers are flexible through hardware reconfiguration. For many computationally intense applications, e.g., from the digital signal processing domain, reconfigurable systems have been shown to achieve a higher performance and an increased energy efficiency compared to processors.

While early FPGAs were rather limited in their densities and reconfiguration capabilities, todays devices provide several millions of gates and enable partial reconfiguration and readback. This allows to configure and execute a circuit onto the device without affecting other, currently running circuits. To express the dynamic nature of such circuits we denote them as *hardware tasks*. In many of the promising application domains for reconfigurable embedded systems, such as networked mobile systems [1] and wearable computing systems [2], the activation times and frequencies of the different tasks are only known at runtime. Task execution is triggered by user-generated events and changes in the environment.

Such highly dynamic situations ask for a well-defined set of system services that support an efficient design of reliable and portable applications and manage the reconfigurable resource at runtime. These system services are denoted as *reconfigurable operating system*. Reconfigurable operating systems are a rather new line of research. One of the first descriptions of *hardware multitasking* is due to Brebner [3]. More recently, Wigley et al. [4] discussed operating system services including device partitioning, allocation, placement, and routing. The preemption of hardware tasks was investigated by Simmler et al. [5].

A central issue in a reconfigurable operating system is the online scheduling of tasks to the partially reconfigurable resource. The difficulty of this problem stems from the fact that task and resource management are strongly coupled. A multitasked reconfigurable device can be seen as multiprocessor with an additional global resource constraint. As discussed below, *partitioning* of the reconfigurable surface relaxes this resource constraint to some extend and simplifies scheduling.

This paper presents our work toward an operating system layer for reconfigurable devices. Section 2 discusses different area models. Section 3 details our implementation of non-preemptive and preemptive online schedulers for 1D block-partitioned reconfigurable devices. A simulation environment and experimental results are discussed in Section 4. Section 5 reports on the status of our prototype. The main contributions of this paper are the development of:

- online schedulers for 1D block-partitioned reconfigurable devices
- a simulator that accurately models the device's configuration port with task-dependent configuration and readback times

2 Reconfigurable Resource Management

The reconfigurable resource is usually modeled by a rectangular array of configurable logic blocks, and tasks are modeled by smaller rectangles of such logic blocks. These

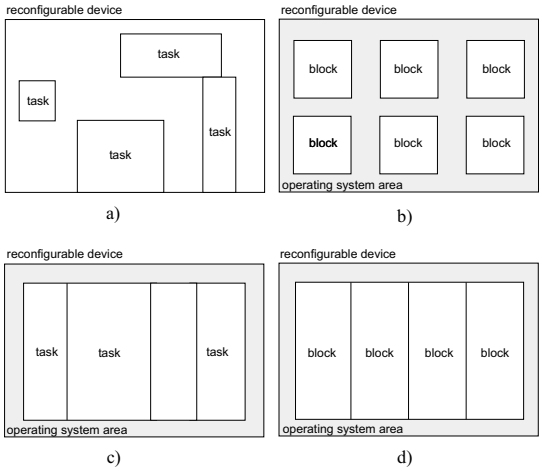

Figure 1. Reconfigurable resource models.

assumptions are reasonable as design tools allow to constrain the placement and, only recently, the routing of circuits to rectangular areas on the device.

The online scheduling problem includes the *placement* of tasks on the reconfigurable surface as an important subproblem. Given a specific allocation of tasks to the device, not every ready task might be placeable which strongly effects scheduling. The *area model* of the reconfigurable resource relates closely to the complexity of the scheduling and placement problems. In the following sections, we discuss several area models with their advantages and disadvantages.

2.1 2D Area Models

Figure 1a) shows the most flexible area model that allows to allocate rectangular tasks anywhere on the 2D reconfigurable surface. This model has been used by many authors [3] [6] [7]. The advantage of this model is a high device utilization as tasks can be packed tightly. On the other hand, the high flexibility of this model makes scheduling and placement rather difficult. The development of online placement algorithms that find a good – or even feasible – allocation site for a task is not trivial. Bazargan et al. [6] address this problem and devise efficient data structures and heuristics. When tasks are placed on arbitrary positions the remaining free area is fragmented. This *external fragmentation* may prevent the placement of a new task although sufficient free resources are available. To combat this external fragmentation, Diessel et al. [7] investigate defragmentation techniques such local repacking and ordered compactation.

The area model in Figure 1a) includes some abstractions that are difficult to reduce to the practice of current FPGA technology (Xilinx Virtex). First, the tasks will require external wires to connect to other tasks and I/O pads. Related work either assumes that this communication is established inside the rectangular task area via configuration and readback (which is feasible but presumably inefficient) or proposes to leave some space between tasks for communication channels. Second, task connections and I/O must be dynamically rerouted and the timing must be reanalyzed which is not supported by current commercial design tools.

Figure 1b) shows a 2D partitioned model where the reconfigurable surface is split into a statically-fixed number of allocation sites, so-called blocks. Each block can accommodate one task at a time. Such a partitioning has been proposed by Merino et al. [8] and Marescaux et al. [9]. Partitioning the area simplifies scheduling and placement and makes a practical implementation on current technology more realistic. As the blocks have fixed positions, the remaining area can be made an operating system resource. Communication channels and I/O are provided exclusively by the operating system. With fixed interfaces between the tasks and the operating system there is no need for online rerouting and timing analysis [10]. The disadvantage of a partitioned area model is *internal fragmentation*, i.e., the area wasted when a task is smaller than a block.

2.2 1D Area Models

Currently available FPGA technology (Xilinx Virtex) is partially reconfigurable only in vertical chip-spanning columns. Hence, the configuration of a task potentially interferes with other tasks allocated to the same columns.

Figure 1c) shows a 1D area model where tasks can be allocated anywhere along the horizontal dimension; the vertical dimension is fixed. Such a model has been described by Brebner and Diessel [11], leads to simplified scheduling and placement, and is amenable to an implementation on Xilinx Virtex. However, the model suffers from both internal and external fragmentation which asks for defragmentation techniques. Figure 1d) finally shows a 1D block-partitioned area model which combines the simplified scheduling and placement of the model in Figure 1b) with the implementation advantages of the model in Figure 1c). Again, the disadvantage lies in the high internal fragmentation.

2.3 Our Resource Model

In this paper we focus on the architectural model shown in Figure 2. The reconfigurable device is controlled by a host CPU that runs the online scheduler, including the placer, and performs configuration and readback via a single config/readback port. The configuration and readback times depend on the tasks' widths. There are several implementations for this system. First, the host CPU can be externally attached to the reconfigurable device. Second, host CPU and the reconfigurable area can be integrated in a so-called configurable system on chip (CSoC). Finally, the

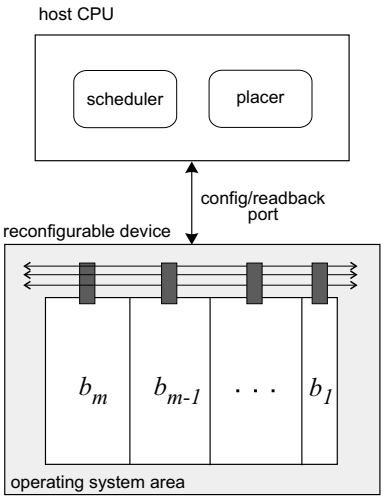

Figure 2. Architecture model.

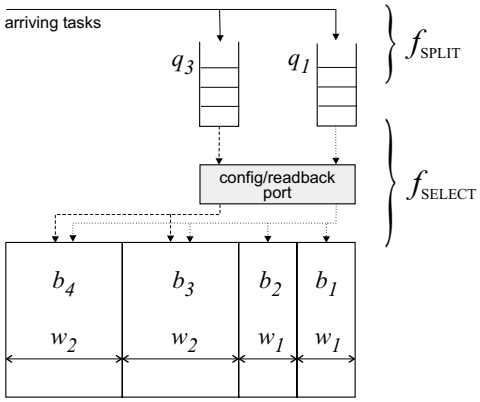

Figure 3. Structure of the online scheduler.

host CPU can be implemented in the operating system area of the reconfigurable device as a synthesizable soft core.

The reconfigurable device contains blocks of fixed size with the same vertical dimension. We allow for different block widths, i.e., there are m blocks with $l, l \leq m$, different widths $w_1, \ldots, w_l$. We assume that communication takes place via preallocated channels in the operating system's area. Scheduling and placement is not constrained by communication requirements. Therefore, without loss of generality we arrange the blocks such that their widths decrease monotonically from left to right, i.e., $w_j \geq w_i$ for $j > i$. The area model in Figure 1d) is a special instance of our resource model.

The motivation of having differently-sized blocks is to achieve a better match between the resources and the tasks. Adapting block widths to task widths decreases internal fragmentation and leads to a higher average resource utilization. Although the block widths are fixed during the scheduling of a task set, we are still able to change the widths on a longer time scale. If, for example, the maximum task width for an upcoming task set is known, the device can be repartitioned to limit the maximum block width to the maximum task width and increase the number of available blocks.

3 Online Scheduling Methods

We assume task sets consisting of n unrelated tasks. Each task t_j is characterized by an a-priori unknown arrival time a_j, the area requirement expressed by its width w_j, the execution time requirement c_j and, optionally, a deadline d_j. Configuration and readback times are proportional to the task width.

We have implemented a number of non-preemptive and preemptive schedulers using policies well-known from single processor scheduling. All schedulers use the structure shown in Figure 3 which consists of a number of *queues* and the two functions f_{SPLIT} and f_{SELECT}. The number and positions of the queues depend on the device's block partitioning. A queue q_j is created and assigned to block b_j if the next block to the right, b_i, has a smaller width, $w_j > w_i$. The right-most block b_1 always gets a queue q_1 assigned. Function f_{SPLIT} works in two steps. First, it assigns an arriving task t_x with width w_x in the right-most queue corresponding to a block wide enough to accommodate t_x. Second, f_{SPLIT} inserts the task into that queue according to some sorting rule.

The function f_{SELECT} actually selects and places the task that is to be executed next. f_{SELECT} is invoked every time an executing tasks terminates, a configuration or readback process ends, or a new task arrives at the head of some queue. Among all queue heads, f_{SELECT} selects a task that can be allocated and configures it onto the smallest idle block able to accommodate the task. The selection is based on some selection rule.

The placer in f_{SELECT} can operate in two modes. In the *restrict* mode, tasks in queue q_i can only be placed into blocks that correspond to q_i. In the *prefer* mode, the placer can allocate a task to any block that is able to accommodate it. Consequently, tasks waiting in queue q_j can be allocated to blocks $b_j, \ldots, b_m$, but not to blocks $b_1, \ldots, b_i$. The example in Figure 3 indicates the prefer mode. Tasks from q_3 can be placed in b_4 and b_3, whereas tasks in q_1 can be placed in any block.

3.1 Non-preemptive Methods

The non-preemptive schedulers neither preempt tasks running on the reconfigurable device nor the configuration process itself. Once f_{SELECT} selects a task, the task is loaded

and run to termination. We have implemented the following non-preemptive schemes:

- *First Come First Serve (FCFS)*
  $f_{\text{SPLIT}}(\text{FCFS})$ assigns a timestamp to each arriving task and inserts it into the appropriate queue. The sorting rule is first-in first-out (FIFO). The selection rule of f_{SELECT} is to pick the task with the earliest arrival timestamp.

- *Shortest Job First (SJF)*
  $f_{\text{SPLIT}}(\text{SJF})$ sorts the queues according to the execution times of the tasks. In each queue, the head entry identifies the task with the smallest execution time. The selection rule for $f_{\text{SELECT}}(\text{SJF})$ is to pick the task with the smallest execution time.

3.2 Preemptive Methods

The preemptive schedulers preempt tasks running on the reconfigurable device to allocate a task with higher priority. Moreover, the configuration and readback processes can also be preempted (load abort and unload abort). Figure 4 shows the resulting task state diagram. Configuration processes are always aborted by higher-priority tasks. A readback process that unloads a block b_l is aborted only when the higher-priority task is to be loaded onto a block different from b_l. Otherwise, the readback is continued as b_l must be unloaded anyway. We have implemented the following preemptive schemes:

- *Shortest Remaining Processing Time (SRPT)*
  $f_{\text{SPLIT}}(\text{SRPT})$ sorts the queues according to the remaining execution times of the tasks. The selection rule for $f_{\text{SELECT}}(\text{SRPT})$ is to pick the task with the smallest remaining execution time.

- *Earliest Deadline First (EDF)*
  $f_{\text{SPLIT}}(\text{EDF})$ sorts the queues according to the task deadlines. In each queue, the head entry identifies the task with the earliest deadline. The selection rule of $f_{\text{SELECT}}(\text{EDF})$ picks the task with the earliest deadline.

When all blocks are of equal width, there is only one queue assigned to the right-most block. In this case, FCFS, SJF, SRPT, and EDF behave exactly as their single processor counterparts. Differently-sized blocks pose an additional resource constraint that might break the policy. For example, FCFS might schedule a later arrived smaller task before an earlier arrived bigger task.

3.3 Finding Good Partitionings

The partitioning determines the number and widths of the blocks. Finding a partitioning that yields good scheduling results, e.g., measured by the total execution time for a task set, is a non-trivial task and will likely involve experimentation.

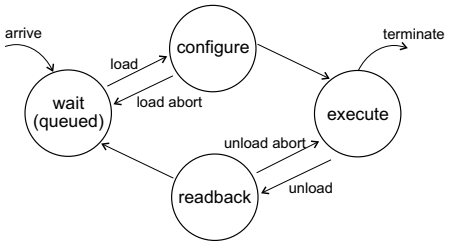

Figure 4. Task states for preemptive scheduling.

The following paragraph considers a rather simple scenario where a set of tasks has to be executed. The scheduling goal is to minimize the overall execution time. All tasks arrive at the same time, their widths are uniformly distributed in $[w_{min}, w_{max}]$, and their execution times are also uniformly distributed. The placer operates in the restrict mode. The device is partitioned into blocks of width $w_1, \ldots, w_l$, with a number of $m_1, \ldots, m_l$ blocks of each width. The restrict mode groups the tasks into classes. A class is defined by an interval of widths, e.g., $(w_{i-1}, w_i]$. A task falling into this interval finds m_i blocks to execute on. We define $\delta_i = w_i - w_{i-1}$ and $\Delta = w_{max} - w_{min}$. The percentage of tasks scheduled to $(w_{i-1}, w_i]$ is δ_i/Δ. This expression can be taken as measure for the overall execution time requirement (load) for this class. There are m_i blocks assigned to the class $(w_{i-1}, w_i]$, which gives $\delta_i/(\Delta \cdot m_i)$ as measure for the class' overall execution time. The task set's overall execution time is the maximum over all class execution times. Consequently, a good partitioning should select the parameters w_i and m_i in order to

$$\text{minimize:} \quad \max_{s=1\ldots l} \left\{ \frac{\delta_s}{\Delta \cdot m_s} \right\} \tag{1}$$

As an example, consider a device with $\overline{W} = 80$, $w_{min} = 4$ and $w_{max} = 20$. The partitioning $[3 \times 20, 2 \times 10]$ leads to an execution time metric of $\max\left\{\frac{7}{17 \cdot 2}, \frac{10}{17 \cdot 3}\right\} = 0.206$. The partitioning $[2 \times 20, 2 \times 15, 1 \times 10]$ leads to an execution time metric of $\max\left\{\frac{7}{17 \cdot 1}, \frac{5}{17 \cdot 2}, \frac{5}{17 \cdot 2}\right\} = 0.41$. The first partitioning can thus be expected to work better for the described scenario.

4 Simulation Experiments

We have implemented a simulation framework to experimentally investigate the behavior and the performance of the online schedulers. The parameters of the simulator include the dimensions of the reconfigurable device, the

configuration and readback times for one device column, and the block partitioning. In the current state the simulation neglects CPU runtimes required for i) the bitstream manipulations to relocate tasks to different allocation sites, ii) the context extraction and -insertion, and ii) the online schedulers. The simulation framework comprises the simulator module, a task generator, a module for data collection and statistical analysis including a Gantt chart viewer, and a graphical display of the allocation situation and queue loads. In the following, we report on a number of selected experiments conducted with this simulation environment.

4.1 Evaluation of Partitionings and Placers

This section presents an experiment to investigate the influence of the partitioning and placement on the performance of the FCFS scheduler. We assume independent tasks. Hence, the scheduler performance is measured by the average response time for a task set. The response time r_i of a task t_i is given by $r_i = f_i - a_i$, whereas f_i is the task's finishing time and a_i is its arriving time. The simulation models the Xilinx Virtex XCV1000 where the configuration or readback of one column takes $159\mu s$. We assume that only 80 columns (out of 96 columns available) are usable for blocks; the remaining columns are occupied by the operating system. Since there are no benchmarks or statistical data available from real-world applications so far, we have to resort to randomly generated tasks. We have generated task sets with 100 task each. The tasks widths are uniformly distributed in $[4,20]$ columns, the execution times in $[2,200]ms$, and the arrival times in $[0.5,500]ms$. The resolution of the simulator, the duration of one clock tick, has been set to $500\mu s$.

Figure 5. Average response time for FCFS.

Figure 5 shows the results for both placer modes, restrict and prefer, and five different partitionings. For both placer modes, the partitioning $[1\times 20, 2\times 15, 3\times 10]$ performs best. In the restrict mode, the partitioning $[2\times 20, 2\times 15, 1\times 10]$ clearly shows a bottleneck for small tasks. The prefer mode achieves a much better result for this partitioning, as the smaller tasks can be allocated to the larger blocks as well. The partitioning $[1\times 20, 10\times 6]$ is a pathologic case because the 10 blocks with width 6 are quite ineffective. In this case, 88% of the total load is scheduled to the one large block. Consequently, the average response time increases dramatically.

4.2 Influence of Configuration and Readback

This section discusses an experiment to analyze the effects of configuration and readback on the system performance. For this, we have simulated the SRPT scheduler in the restrict placement mode with and without modeling the configuration port. The scheduler performance is measured by the overall execution time for a task set. We have generated task sets with 25 tasks each. The arrival times are distributed in $[0.1,10]ms$ to eliminate the influence of late arriving tasks on the overall execution time. All other parameters are identical to the previous section which results in configuration and readback times of $[0.6,3.1]ms$. The results show an increase in the overall execution time of 1.2% to 7.3% when configuration and readback times are included. We believe that in the targeted application domains, such as wearable computing [2], typical task execution times will be higher than the times assumed in this experiment. Consequently, the presented partially reconfigurable system will not suffer from a bottleneck formed by the single configuration port.

4.3 EDF Scheduling

Figure 6 displays a screenshot of the Gantt chart viewer. The reconfigurable surface is partitioned into two blocks (b_1,b_2) of widths $(w_1,w_2) = (5,10)$. The scheduler runs in EDF mode; the placer in the prefer mode. The example details the scheduling of a task set with following four tasks $t_x(a_x,w_x,c_x,d_x)$:
$t_1(1,5,8,26),\ t_2(2,5,8,24),\ t_3(8,5,4,23),\ t_4(9,8,4,20)$

The configuration and readback times for t_1, t_2 and t_3 are 2, for task t_4 3 time units. At time 1, t_1 arrives and starts to configure onto b_1. At time 2, t_2 arrives and gets a higher priority than t_1. The placer allocates t_2 to b_2 (prefer mode). The configuration of t_1 is preempted (configuration abort) and t_2 is configured onto b_2.

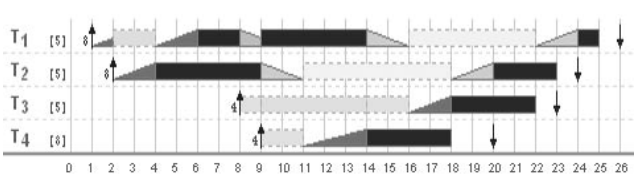

Figure 6. Gantt chart of an EDF schedule.

At time 4, the config/readback port is released by t_2. t_1 starts again configuration onto b_1. At time 8, t_3 arrives with a higher priority than t_1 and t_2. As both blocks are already in use, the scheduler selects t_1 to be preempted due to its long deadline. Readback of t_1 from b_1 starts. At time 9, t_4 arrives with the highest priority among all tasks. t_4 can only be placed in b_2. Consequently, t_2 must be preempted. The readback of t_1 which is currently in progress is stopped (unload abort) and the readback of t_2 is started instead. t_1 resumes execution on b_1. At time 11, readback of t_2 has finished and t_4 is loaded onto b_1.

5 Prototype Implementation

The prototype is work in progress. The implementation is done on a PC-based XESS XSV-800 board equipped with a Virtex XCV-800 FPGA. The tasks are designed in VHDL and synthesized to full FPGA bitstreams with commercial design tools (Xilinx Foundation). From these full bitstreams we extract partial bitstreams [10]. Then we localize and store the positions of the status bits within the partial bitstreams. These status bits define the tasks' contexts. We have created an operating system frame on the FPGA and partitioned the remaining area into a number of blocks. Each block is provided with a reset signal. The host PC can execute following operating system functions:

- *task load:* The task is relocated by modifying the frame addresses in the partial bitstream. Then, the partial bitstream is written to the FPGA. Finally, the block's reset signal is enabled which loads the initial state and starts task execution.
- *task preemption:* The running task's state is captured by activating the capture signal on the FPGA. Then, the task with its state is read back. Finally, the context is extracted and saved by accessing the state bits in the readback bitstream.
- *task resume:* This is basically identical with task load. The only difference lies in the insertion of the previously extracted state into the partial bitstream before loading.

We have created rather simple test tasks in the complexity of counters. For debugging purposes, we allow the blocks access to I/O pins that connect to external LEDs. We have been successful in loading, preempting, and resuming such tasks on the FPGA. Although the correct function has been verified, we must note that the tasks have limited sizes. Future work includes the implementation and test of more complex tasks.

6 Conclusion and Further Work

In this paper we discussed area models for reconfigurable devices. We introduced a 1D block-partitioned model and devised an online scheduling system that schedules tasks according to several non-preemptive and preemptive policies. We presented a simulation framework that allows to evaluate specific partitionings, placements, and scheduling methods.

Further work includes the investigation of novel scheduling techniques that adapt the resources by a repartitioning of the device and the implementation of a more complex prototype for wearable computing [2]. We also intend to look at application scenarios with related tasks and tasks with hard deadlines.

7 Acknowledgments

This work was supported by the Swiss National Science Foundation (SNF) under grant number 2100-59274.99.

References

[1] Rudy Lauwereins. Creating a World of Smart Re-configurable Devices. In *Proc. 12th Int'l Conf. on Field-Programmable Logic and Applications (FPL)*, pages 790–794, 2002.

[2] Christian Plessl, Rolf Enzler, Herbert Walder, Jan Beutel, Marco Platzner, and Lothar Thiele. Reconfigurable Hardware in Wearable Computing Nodes. In *Proc. 6th Int'l Symposium on Wearable Computers (ISWC)*, 2002.

[3] Gordon Brebner. A Virtual Hardware Operating System for the Xilinx XC6200. In *Proc. 6th Int'l Workshop on Field-Programmable Logic and Applications (FPL)*, pages 327–336, 1996.

[4] Grant Wigley and David Kearney. The Development of an Operating System for Reconfigurable Computing. In *Proc. IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, 2001.

[5] H. Simmler, L. Levinson, and R. Männer. Multitasking on FPGA Coprocessors. In *Proc. 10th Int'l Workshop on Field Programmable Gate Arrays (FPL)*, pages 121–130, 2000.

[6] Kiarash Bazargan, Ryan Kastner, and Majid Sarrafzadeh. Fast Template Placement for Reconfigurable Computing Systems. In *IEEE Design and Test of Computers*, volume 17, pages 68–83, 2000.

[7] O. Diessel, H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt. Dynamic scheduling of tasks on partially reconfigurable FPGAs. In *IEE Proceedings on Computers and Digital Techniques*, volume 147, pages 181–188, May 2000.

[8] Pedro Merino, Margarida Jacome, and Juan Carlos Lopez. A Methodology for Task Based Partitioning and Scheduling of Dynamically Reconfigurable Systems. In *Proc. IEEE Symopsium on FPGAs for Custom Computing Machines (FCCM)*, pages 324–325, 1998.

[9] Théodore Marescaux, Andrei Bartic, Verkest Dideriek, Serge Vernalde, and Rudy Lauwereins. Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs. In *Proc. 12th Int'l Conf. on Field-Programmable Logic and Applications (FPL)*, pages 795–805, 2002.

[10] Matthias Dyer, Christian Plessl, and Marco Platzner. Partially Reconfigurable Cores for Xilinx Virtex. In *Proc. 12th Int'l Conference on Field-Programmable Logic and Applications (FPL)*, pages 292–301. Springer, September 2002.

[11] Gordon Brebner and Oliver Diessel. Chip-Based Reconfigurable Task Management. In *Proc. 11th Int'l Workshop on Field Programmable Gate Arrays (FPL)*, pages 182–191, 2001.