Non-preemptive Multitasking on FPGAs: Task Placement and Footprint Transform

Herbert Walder and Marco Platzner
Computer Engineering & Networks Lab
Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

Abstract ***Partial reconfiguration allows for mapping and executing several tasks on an FPGA during runtime. Multitasking on FPGAs rises a number of questions on the management of the reconfigurable resource, which leads to concepts of reconfigurable operating systems.***

This paper focuses on a major aspect of a reconfigurable operating system: task placement and transformation. We first discuss task characteristics and system models, and then concentrate on the execution of independent task sets on non-preemptive reconfigurable systems. We investigate placement techniques for non-rectangular, coarse-grained tasks and propose footprint transforms that change task shapes in order to find possible mappings. Finally, we discuss simulation experiments to evaluate these techniques.

Keywords: partial reconfiguration, task placement, footprint transform, operating system

1 Introduction

Custom computing machines map algorithms or parts thereof to a circuit which is configured and executed on an SRAM-based Field-Programmable Gate Array (FPGA). While early FPGAs were rather limited in their densities and reconfiguration capabilities, todays devices provide several millions of gates and allow for faster and partial reconfiguration.

FPGA circuits are increasingly designed and traded as intellectual property (IP) cores. Larger FPGAs can accommodate several such cores, which allows to prototype and implement complete systems-on-a-chip. To express their dynamic nature, we denote such cores as *tasks*. An application is basically a collection of tasks that are subject to timing, precedence, and resource constraints.

Partial reconfiguration allows to configure and execute a task onto an FPGA without affecting other currently running tasks. While this technique can increase device utilization, it also leads to complex allocation situations for dynamic task sets. This clearly asks for well-defined system services that help to efficiently design applications. Such a set of system services forms a *reconfigurable operating system*. From the designer's point of view, an operating system is an additional layer of abstraction in the design process that hides details of the underlying hardware. The benefits of using the additional layer are an increased design productivity, portability, and resource utilization. On the other hand, these benefits are paid for by overheads in the required area and computation time.

An important service of any reconfigurable operating system is *task placement*. The decision where a task is mapped to determines the fragmentation of the reconfigurable surface. A high fragmentation can lead to the undesirable situation that a task cannot be placed although there would be sufficient area available. Generally, we differentiate *proactive* and *reactive* strategies to combat fragmentation. An example for a proactive strategy is a placement technique that makes the successful placement of subsequent tasks more likely. Reactive strategies apply when a task cannot be placed due to fragmentation. One technique, that can be used proactively and reactively, is compaction or repacking. Currently running tasks are packed more closely in order to free larger contiguous areas. Compaction requires preemption, where running tasks are stopped and, at a different location, continued. With current FPGA technology, preemption is rather costly in terms of time. A different reactive strategy that does not require preemption is *footprint transform*. Footprint transform tries to change the shape of a task to fit it into an existing free area.

In this paper, we discuss a non-preemptive partially reconfigurable system that executes a set of independent tasks. We investigate two proactive placement techniques, first-fit and best-fit, and the feasibility of footprint transforms. The main contributions of this paper are:

- investigation of *placement* techniques for coarse-grained, *non-rectangular* tasks, based on a fragmentation metric
- investigation of *footprint transforms*
- *simulation experiments* to evaluate placement and footprint transform techniques

2 Related Work

A substantial body of work has been done in offline optimization of reconfigurable systems. Examples are temporal partitioning for non-partially reconfigurable systems, e.g., in [1], or 3D placement of tasks in time and space dimensions for partially reconfigurable devices, e.g., in [2] [3]. In an offline scenario, one can afford to spend the time to derive optimal or near-optimal solutions. An online scenario is discussed by Jean et al. [4]. They present a resource manager that schedules tasks online to a farm of non-partially reconfigurable FPGAs. However, as tasks occupy complete devices, placement is not an issue.

The remainder of this section briefly lists a selection of related work in operating systems for partially reconfigurable FPGAs, focusing on task placement. Brebner [5] [6] proposed such an operating system based on swappable logic units (SLUs). SLUs are position independent tasks that are swapped in and out by the operating system. Merino et al. [7] [8] split the reconfigurable surface into a number of predefined subareas, so-called slots. The operating system schedules tasks to these slots, based on a task allocation table that keeps track of currently loaded tasks. Simmler et al. [9] discuss task switching in a preemptive operating system. The authors also underline the importance of critical sections, i.e., periods of time during which a task must not be preempted. Burns et al. [10] describe several operating system functions, including a transform manager that performs translation and rotation operations.

Diessel et al. [11] [12] tackle the fragmentation problem in partially reconfigurable FPGAs. They perform a task rearrangement by techniques denoted as local repacking and ordered compaction. Wigley and Kearney [13] identify services that must be offered by reconfigurable operating systems, including partitioning, allocation, placement and routing. They further indicate a method to quantify fragmentation. Compton et al. [14] discuss task relocations and transforms to reduce fragmentation. Task transforms consist of a series of rotation and flip operations. The authors also propose a novel FPGA architecture that supports efficient row-wise relocation. Barzagan et al. [3] investigate efficient data structures and algorithms for fast online placement. They conducted simulation experiments for variants of first-fit, best-fit and bottom-left bin-packing algorithms.

In contrast to related work, this paper focuses on tasks of coarse granularity that need not be of rectangular shape. We assume that such tasks consist of a number of subtasks, which results naturally from core-oriented design styles. We further discuss the feasibility of footprint transforms performed at subtask level.

3 Task and System Models

This section first discusses tasks characteristics for a reconfigurable operating system. Then, we outline the non-preemptive system model we use to experiment with in this paper. Finally, we comment on several scheduling scenarios for further investigation.

3.1 Task Characteristics

A task has several qualities. The functional quality captures the task behavior and is not visible to the operating system. We define following structural and timing qualities that the operating system needs to know for scheduling and placement:

Structural Qualities

- *Size:* Tasks have a certain area requirement, given in number of CLBs. We consider coarse-grained tasks with typical sizes as listed in Table 1.

core	*area* [CLB]
UART [15]	50
100-tap FIR filter [16] 12 bit data & coefficients	250
ADPCM [17]	250
DCT [18]	600
Triple-DES processor [16]	800
256 point complex FFT [16]	850
minimal protocol stack [15] Ethernet-MAC, IP, UDP	1050
Discrete Wavelet Transform [18]	1800
LEON Sparc-V8 core, 32 bit mem I/F [19] 2Kbit I-cache, 2Kbit D-cache	2000
MPEG2 video decoder [18]	3650

Table 1: Area requirements for typical FPGA cores.

- *Shape:* Related work often characterizes task shapes by rectangles including all CLBs and routing resources used by the task. With current design tools, rectangular task shapes are achieved by first constraining placement and then defining the smallest enclosing rectangle as final shape. While rectangular shapes simplify allocation, they also lead to internal fragmentation, i.e., unused and thus wasted subareas of the rectangle. We assume task shapes to be *polyominoes*. A polyomino is a connected subset of the square lattice (CLB) tiling of a plane (FPGA surface). We argue that such task shapes naturally result from a core-oriented design style. Figure 1 shows an example of a task that

implements a communication protocol stack composed of a number of cores. While the single cores can be characterized by rectangular shapes with a reasonably small fragmentation, the smallest enclosing rectangle for the overall task leads to a large fragmentation.

- *Relocatability:* Tasks that are non-relocatable must be placed exactly on their predefined positions, e.g., to access special resources such as block RAM. Relocatable tasks allow for translation, i.e., they can be placed at arbitrary positions with different row and column offsets. Current FPGA architectures show some limitations in the granularity of translation, as they are not totally homogeneous. Task relocation involves some difficulties and is not yet supported by FPGA design tools which produce only non-relocatable tasks. While intra-task wires are translated, wires running between different tasks or between tasks and I/O need to be re-routed dynamically. After re-routing, the delay of the new routes must be determined to guarantee the timing.

- *Transformability:* Complex cores often split up into several submodules, as shown in Figure 1. The figure shows a communication protocol stack, that implements Ethernet-MAC, IP (Internet Protocol) and UDP layers. The IP layer supports minimal ICMP (echo request) and ARP (address resolution) functionality. The protocol stack divides into subtasks or cores that handle the functions of the receive and send paths of the different protocol layers. Over time, the actual number of submodules that need to be instantiated can vary. Submodules express locality of operation. The number of wires connecting submodules is thus usually much smaller than the number of wires running exclusively inside the subtasks. In addition, wires between submodules are supposed to be less time critical than those inside. This motivates footprint transforms. A transformable task provides its inner structure, i.e., a list of relocatable subtasks, to the operating system.

- *Spatial critical sections:* A spatial critical section marks a contiguous subarea of a task. The operating system must not change the relative locations of the CLBs inside this area. An example are the fast carry chain resources.

Timing Qualities

- *Required cyles:* A task requires a certain number of cycles to execute, which might or might not be known in advance. The actual execution time is determined by the number of cycles and the clock rate at which the task runs.

- *Clock range:* This is the range of clock frequencies at which the task can run. Design tools usually report an upper bound for the clock rate. A task may, however, require a specific clock rate, for example to derive a timer object that relates events to physical time. A task might further require a clock rate in a certain interval to preserve timing requirements of I/O devices or memory. The clock rate is important for an operating system, as modern FPGAs have several clock nets and can run several tasks at different clock rates.

- *Time critical sections:* A critical section in time denotes a time period in which a task must not be interrupted. An example is the timing for I/O protocols.

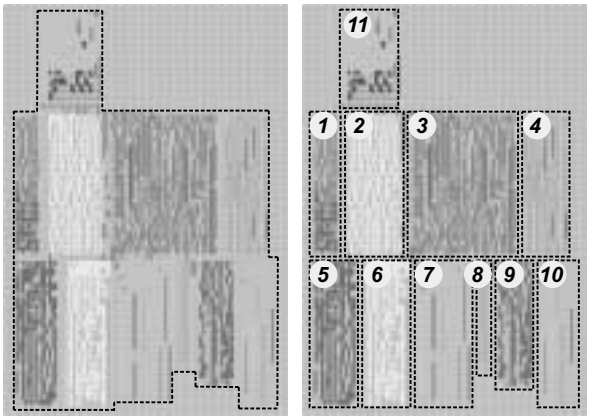

Figure 1: Floorplan of a communication protocol stack.

3.2 Non-preemptive Multitasking

The experiments described in this paper assume the system model shown in Figure 2. The tasks that are executed are independent, i.e., there are no precedence constraints, and there are no real-time constraints to satisfy. The asynchronous arrival times as well as the computation times are a-priori unknown. These characteristics reflect a general-purpose computing system.

We focus on non-preemptive partially reconfigurable FPGAs. Once a task is loaded onto the FPGA, it runs to termination. Formally, a task t_i arrives at time a_i, starts to execute at time s_i, and finishes at f_i. Thus, the task's waiting time is given by $w_i = s_i - a_i$ and the response

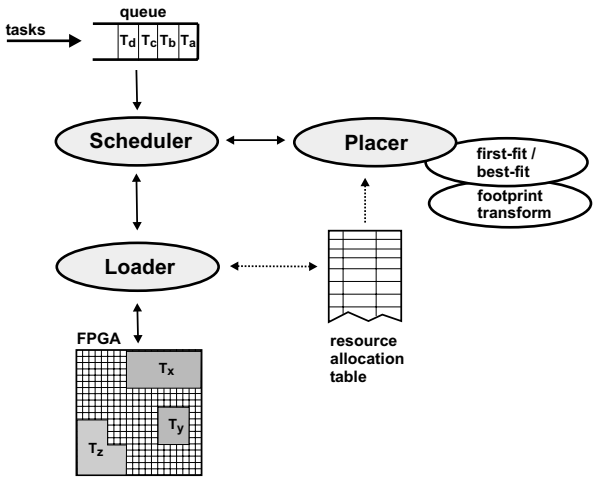

Figure 2: Non-preemptive system model

time is given by $r_i = f_i - a_i$. We set several system objectives:

- If n tasks belong to the same application, we are interested in minimizing the overall execution time of the task set:

$$f = \max_{i=1...n} \{f_i\}$$

- For n unrelated tasks that come from different applications, we want to minimize the average waiting time $\overline{w}$ and the average response time $\overline{r}$:

$$\overline{w} = \frac{1}{n} \sum_{i=1}^{n} w_i \qquad \overline{r} = \frac{1}{n} \sum_{i=1}^{n} r_i$$

The system in Figure 2 enqueues arriving tasks in a pending list. The scheduler picks tasks from this queue and calls the placer to check whether a feasible position exists where the task could be mapped to. The placer first searches for a direct fit, i.e., a free area large enough to accommodate the task. We have experimented with first-fit and best-fit direct placement techniques. In case no direct placement is possible, the placer tries footprint transforms to modify the shape of the task. When a feasible position for a transformed task is found, the scheduler calls the loader that actually configures the task onto the FPGA. Placer and loader access the resource allocation table, a central data structure that reflects the FPGA surface. Many scheduling techniques are conceivable. In our current implementation, the scheduler searches the pending task queue for the first task that fits and places it. This process is repeated until no more tasks fit. The scheduler is invoked every time a new task arrives or an executing task terminates.

4 Placement Techniques

Online task placement on partially reconfigurable FPGAs and 2D bin-packing are related problems. Hence, online algorithms used for 2D bin-packing problems, such as first-fit, best-fit, or different bottom-left heuristics, have been considered for task placement [3]. Our work differs from these approaches, as tasks are polyominoes rather than rectangles. In this paper, we evaluate the performance of first-fit and best-fit in terms of placement quality rather than runtime. As the placement algorithms will eventually run on-line, the development of efficient algorithms and data structures [3] is an important next step.

4.1 First-fit

The surface of an FPGA with m rows and n columns is indexed with a rectangular, 2D coordinate system that assigns the coordinates $(0, 0)$ to the bottom-left CLB and $(m-1, n-1)$ to the top-right CLB. First-fit searches the CLB array row-wise from the bottom-left to the top-right corner. Overlapping the search positions with the bottom-left CLB of the task, first-fit tries to match the task shape with free CLBs. The task is mapped to the first matching position found. To derive efficient implementations of first-fit, the search must be directed to feasible regions of the CLB array.

4.2 Best-fit

The rationale behind best-fit is to place a task in a way that makes the successful placement of subsequent tasks more likely. It is reasonable to assume that this is the case when the residual areas on the surface form maximal large rectangles, see [13] [3]. To quantify allocation situations, we introduce a fragmentation metric, the fragmentation grade F. Best-fit determines the fragmentation grade for all possible placements and selects the position that minimizes the resulting fragmentation. For a given allocation, we place the largest possible rectangle into the residual area, note its dimensions and mark it as considered. This process is iteratively continued with the next-largest rectangle, until the complete free area has been marked. The result is a histogram of free rectangular areas. The histogram consists of i classes, each denoting n_i rectangles of size a_i. The fragmentation grade is given as:

$$F = 1 - \frac{\sqrt{\sum_i (n_i \cdot a_i^2)}}{\sum_i (n_i \cdot a_i)}$$

The fragmentation grade is bounded by $0 \leq F < 1$. The lower F is, the higher is the probability that a future task can be mapped. The fragmentation grade is

undefined for 100% utilized FPGAs. Contrary to the approach in [13], our fragmentation grade is normalized to the actual free area and thus device-independent. Possible improvements could add a more global view and prefer connected free rectangles over unconnected ones.

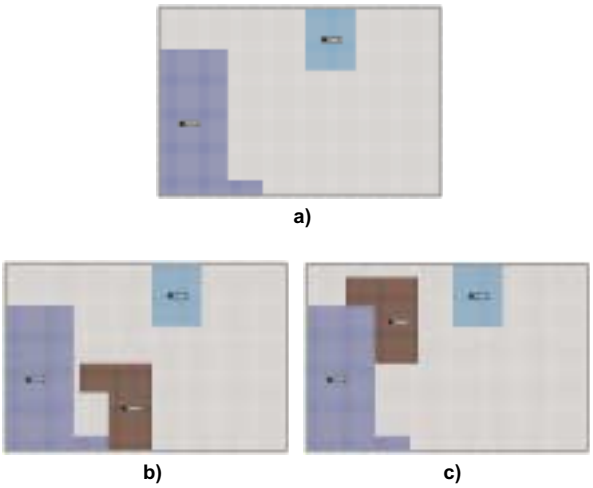

Figure 3: Screen shots from the simulation tool: initial allocation a), first-fit placement b), best-fit placement c).

Figure 3a) shows an allocation with two tasks in an CLB array of size 64×96 (Xilinx XCV1000). The fragmentation grade is 35.6%. A third task is to be placed. Figure 4 presents the resulting fragmentation grades for the different positions of the task in the CLB array. In this plot, the grade is set to zero when the task cannot be placed. Due to the dimensions of the new task, only a subset of the array has to be checked. Figure 3b) displays the first-fit placement with a fragmentation grade of 43%; Figure 3c) shows the best-fit placement with a fragmentation grade of 35%.

5 Footprint Transform

A footprint transform changes the shape of a task. Compared to relocation, online footprint transforms are more complex as they require re-routing of a usually larger number of wires. We differentiate between two granularities of footprint transforms: the subtask level and the CLB level.

Coarse-granular tasks often consist of a set of subtasks, where the number of wires connecting different subtasks is much smaller than the number of intra-subtask connections. This motivates to translate individual subtasks in order to fit the overall task into the available FPGA area. We have experimented with a footprint transform that extends the first-fit placement technique. When no direct fit for a task is found, one of

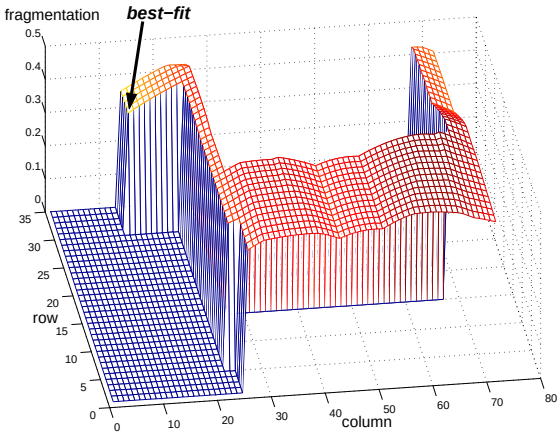

Figure 4: Fragmentation grade for the example in Figure 3 as a function of CLB positions in the XCV1000 CLB array.

its subtasks is translated to different positions with the restriction that the overall task area must remain contiguous. Figure 5 shows an example. The task in Figure 5b) is transformable and consists of three subtasks S_1, S_2 and S_3. Figure 5a) indicates the possible positions subtask S_3 can be translated to. When the transforms do no lead to a feasible placement, the next subtask is considered. Figures 5c) and 5d) show different transforms with translations of S_3; Figure 5e) displays a translation of S_1.

Footprint transforms at the CLB level are atomic. All other transforms, including flipping and rotation, can be expressed by a sequence of CLB translations. However, transforming many CLBs of a coarse-grained task does not seem to be realistic, because of the complex issues of re-routing and timing preservation.

6 Experiments and Discussion

To evaluate the placement techniques, we have conducted a series of simulation experiments. We simulated the system described in Section 3 with randomly generated task sets. Each task set contains 100 tasks. The arrival times are equally distributed in $[1, 100]$ time units, the computation times in $[5, 25]$ time units. We have generated five classes of task sets differing in task size. The classes are denoted by C_i and contain tasks of equally distributed size in the interval $[100, i]$ CLBs. We make a number of simplifying assumptions and neglect the times required for the placement algorithms, task translations, re-routing, re-calculation of delays, reconfiguration, and task I/O. We also do not model area overheads for task communication and I/O. In a practical environment, these issues may not be negligi-

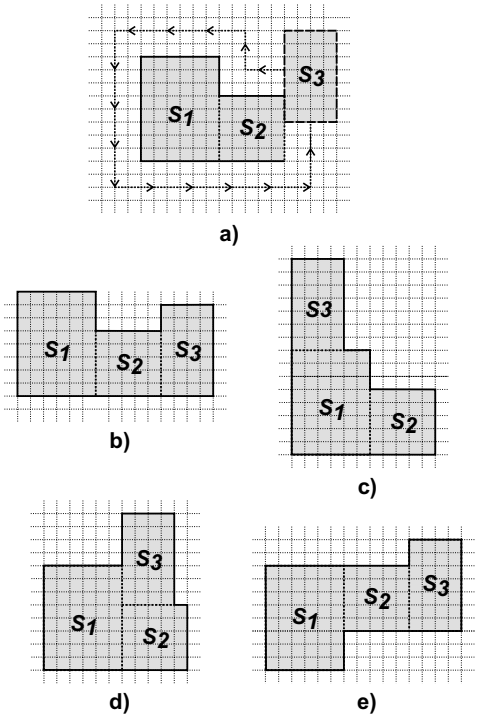

Figure 5: Transformable task b), possible transforms for task S_3 a), transformed tasks c)-d).

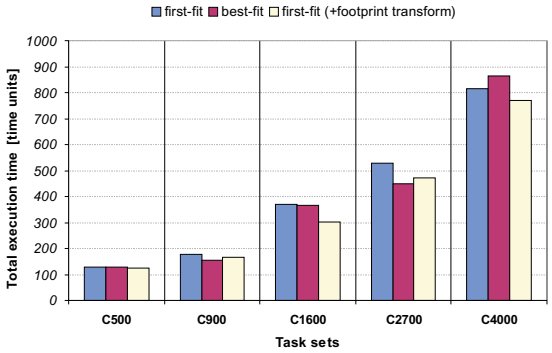

Figure 6: Execution times for different placement techniques and task sets.

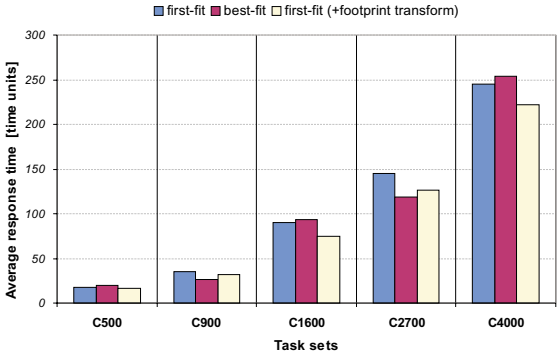

Figure 7: Average response times for different placement techniques and task sets.

ble. Thus, we plan to extend our model to include these effects.

Figures 6 and 7 present the simulation results for the five task classes. The figures show the total execution time and the average response time, respectively, for the placement techniques first-fit, best-fit, and first-fit plus footprint transform. In each class, the data has been averaged over three runs with different task sets. For our specific task sets, following observations can be made:

- Best-fit performs sometimes better than first-fit, sometimes worse. The differences can be significant, e.g., for C_{2700} best-fit outperforms first-fit by 14.9% in the total execution time. On average, best-fit improved the execution time by only 4%.
- Footprint transform turns out to be quite beneficial. On average, 25% of the tasks that could be placed were footprint transformed. If tasks are small compared to the overall FPGA area (C_{500}), footprint transform is not that often applied as many tasks have direct fits. For large tasks, on the other hand, even footprint transform is often unsuccessful. The biggest improvement was achieved for task set C_{1600} with 18.4%. On average, footprint transform improved execution time by 8.7% over first-fit.
- Figure 8 displays the reduction in execution time using best-fit and footprint transform. Although we have not concentrated on efficient implementations of the placement techniques, best-fit and footprint transform are clearly more complex than first-fit. From the simulation result, it could be concluded that time spent for a complex placement technique is best invested in footprint transform, as this technique gives sound and substantial improvements.

7 Conclusion and Further Work

In this paper, we have discussed placement and footprint transform techniques for coarse-grained, non-rectangular tasks on partially reconfigurable FPGAs. We have conducted experiments to evaluate first-fit and best-fit placements, as well as footprint transforms. We plan further work along the following lines:

- We will refine our simulation model to include re-

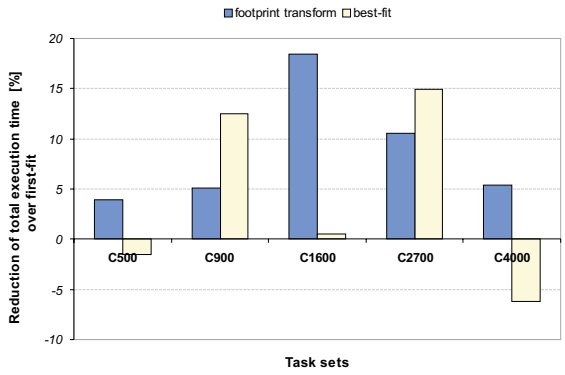

Figure 8: Reductions in execution time for best-fit and footprint transform over first-fit.

configuration times, runtimes for the placement algorithms, etc.

- We will run a larger number of simulations and evaluate other promising placement heuristics, such as combining best-fit with footprint transform.

- For the most promising placement techniques, we plan to devise efficient on-line algorithms.

- We want to demonstrate the feasibility of a reconfigurable operating system by experiments on a prototyping platform using Xilinx Virtex and JBits. Our current status is that we are able to generate relocatable tasks from standard design tool flows. We can translate the tasks and partially configure them onto the FPGA with JBits. The rerouting of external connections and delay calculation for routes is not yet fully operational due to current limitations of JBits.

8 Acknowledgments

This work was supported by the Swiss National Science Foundation (SNF) under grant number 2100-59274.99. We would further like to thank Christian Plessl for many fruitful discussions, and Michael Lerjen for providing his protocol stack IP core and experiments with JBits on Xilinx Virtex.

References

[1] Purna, K. and Bhatia, D. Temporal Partitioning and Scheduling Data Flow Graphs for Reconfigurable Computers. *IEEE Transactions on Computers*, 48(6):556–564, June 1999.

[2] Sandor Fekete, Ekkehard Köhler, and Jürgen Teich. Optimal FPGA Module Placement with Temporal Precedence Constraints. In *Proceedings of Design Automation and Test in Europe (DATE)*, pages 658–665, 2001.

[3] 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.

[4] Jack S.N. Jean, Karen Tomko, Vikram Yavagal, Jignesh Shah, and Robert Cook. Dynamic Reconfiguration to Support Concurrent Applications. *IEEE Transactions on Computers*, 48(6):591–602, June 1999.

[5] Gordon Brebner. A Virtual Hardware Operating System for the Xilinx XC6200. In *Proceedings of the 6th International Workshop on Field-Programmable Logic and Applications (FPL)*, pages 327–336. Springer, 1996.

[6] Gordon Brebner. The Swappable Logic Unit: a Paradigm for Virtual Hardware. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*. IEEE CS Press, 1997.

[7] Pedro Merino, Juan Carlos Lopez, and Margarida Jacome. A Hardware Operating System for Dynamic Reconfiguration of FPGAs. In *Proceedings of the 8th International Workshop on Field Programmable Gate Arrays (FPL)*, pages 431–435. Springer, September 1998.

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

[9] H. Simmler, L. Levinson, and R. Männer. Multitasking on FPGA Coprocessors. In *Proceedings of the 10th International Workshop on Field Programmable Gate Arrays (FPL)*, pages 121–130. Springer, 2000.

[10] Jim Burns, Adam Donlin, Jonathan Hogg, Satnam Singh, and Mark de Wit. A Dynamic Reconfiguration Run-Time System. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, pages 66–75. IEEE CS Press, 1997.

[11] Oliver Diessel and Hossam ElGindy. On Scheduling Dynamic FPGA Reconfigurations. In *Proceedings of the 5th Australasian Conference on Parallel and Real-Time Systems (PART)*, pages 191–200, 1998.

[12] 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.

[13] Grant Wigley and David Kearney. The Development of an Operating System for Reconfigurable Computing. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*. IEEE CS Press, April 2001.

[14] Katherine Compton, James Cooley, Stephen Knol, and Scott Hauck. Configuration Relocation and Defragmentation for Reconfigurable Computing. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*. IEEE CS Press, April 2001.

[15] Michael Lerjen and C. Zbinden. Reconfigurable BlueTooth-Ethernet Bridge. Master's thesis, ETH Zurich, 2002.

[16] Xilinx Inc., Virtex Core Generator.

[17] Matthias Dyer and Marco Wirz. Reconfigurable System on FPGA. Master's thesis, ETH Zurich, 2002.

[18] Amphion Semiconductor Ltd., `www.amphion.com`.

[19] Gaisler Research, `www.gaisler.com`.