CRC – Concepts and Evaluation of Processor-Like Reconfigurable Architectures

CRC – Konzepte und Bewertung prozessorartig rekonfigurierbarer Architekturen

Tobias Oppold, Thomas Schweizer, Julio Oliveira Filho, Sven Eisenhardt,
Wolfgang Rosenstiel, University of Tübingen

Summary The CRC project focuses on the utilization of fast reconfiguration to optimize area, performance, and power. The results are quantified by a synthesizable architecture model and by a commercial architecture. In order to assure good applicability of the research, a C-compiler is co-developed with the architecture. This article provides an overview of the optimization techniques and a summary of current evaluation results. ►►► **Zusammenfassung** Das CRC-Projekt beschäftigt sich mit der Nutzbarmachung schneller Rekonfiguration, um Flächenbedarf, Ausführungsgeschwindigkeit und Verlustleistung zu optimieren. Die Ergebnisse werden anhand eines synthetisierbaren Architekturmodells und einer kommerziellen Architektur beurteilt. Um die entwickelten Konzepte auch praktisch einsetzen zu können, wird ein C-Compiler gemeinsam mit der Architektur entwickelt. Dieser Beitrag liefert einen Überblick über die Optimierungstechniken und eine Zusammenfassung aktueller Bewertungsergebnisse.

KEYWORDS C.1 [Computer Systems Organization: Processor Architectures], C.3 [Computer Systems Organization: Special-Purpose and Application-Based Systems], dynamic hardware reconfiguration, coarse grained reconfigurable architectures, high-level compiler, voltage reconfiguration

1 Introduction

Reconfigurable systems provide the ability to reuse architectural resources over time. Typically, for statically reconfigurable architectures, this reuse happens rarely. For example, FPGAs are often used for prototyping purposes so that the reconfigurable fabric is reused in the range of minutes or even much longer. Reconfiguring such architectures at runtime of the system is mostly done in academic work trying to reuse the architecture, e. g., for various tasks of an embedded system. Due to the long reconfiguration times imposed by transferring the configuration data from outside the reconfigurable fabric, finding a good partitioning is highly application specific and hardly supported by commercial tools.

Newly developed architectures can be reconfigured within one clock cycle, allowing components of a device to be reused within a single application. We call such architectures *processor-like* reconfigurable architectures. They usually have a coarser granularity and require less configuration data than traditional FPGAs so that multiple configuration contexts can be stored inside the reconfigurable array and reconfiguration keeps pace with execution. This yields an additional degree of freedom, which we explore in a design environment that takes advantage of reconfiguration to optimize area, performance, and power. The optimizations pertain to architectural features and compiler techniques since the benefits of reconfigurability can only be exploited if applications can be mapped efficiently.

In the CRC project, we developed a modifiable architecture model (Configurable Reconfigurable Core, CRC) that can be adapted to the requirements of the compiler. Synthesizing instances of the CRC model enables a de-

tailed evaluation, which we cannot accomplish with commercial architectures. On the other hand, using the stable tool chain of commercial architectures enables the evaluation of complex applications more efficiently. Since NEC's DRP (Dynamically Reconfigurable Processor) architecture [6] is very similar to the CRC model with respect to runtime reconfiguration [8], we can use the silicon-proven DRP and its associated tools [12] to obtain additional evaluation data.

In the next section, a short discussion of related work is presented. The CRC model and NEC's DRP architecture are introduced in Section 3. Our optimization approach is presented in Section 4. Results based on the CRC model and the DRP are discussed in Section 5. In Section 6, we present a new approach for fast voltage reconfiguration.

2 Related Work

Early work on fast reconfiguration has been carried out by DeHon [4] and Trimberger et al. [13]. More recently, a growing number of commercial coarse grained architectures appeared on the market, e. g., NEC-DRP, PACT-XPP, IPFlex DAP/DNA, Silicon Hive, Elixent D-Fabrix, and MathStar FPOA, to name just a few. Some of these architectures can be reconfigured within one clock cycle.

Only for a minority of the architectures, a C-compiler that automatically partitions an application into several configurations is available. For example, Mei et al. [5] apply modulo scheduling to map loops of C-descriptions onto the ADRES architecture. NEC developed a C-compiler for the DRP based on their hardware synthesis tool Cyber [14].

3 CRC Model

The CRC model was developed to represent a wide range of processor-like reconfigurable architectures. In its most general specification, only a few features are defined. As depicted in Fig. 1, it consists of a rectangular array of processing elements (PE) that are connected by a reconfigurable interconnect network. Each PE consists of a functional unit (FU) for word-wide arithmetic and logic operations, a register set, and a context memory that defines several configurations for the PE. At the beginning of each clock cycle, a context is selected by a control unit that can vary significantly for the various architectures. Since the interconnect network also varies, both are not further specified in the general CRC model.

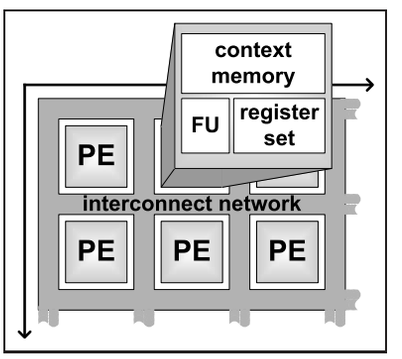

Figure 1 General specification of the CRC model.

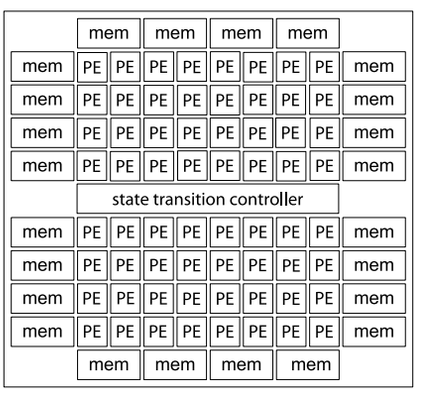

Figure 2 A tile of the DRP architecture [6].

Instances of the CRC model are specified by refinement of the general model. The instances are described as a transaction-level model in SystemC, and at the register-transfer level in Verilog. We use the SystemC implementations for system-level evaluations not further detailed in this article. The SystemC implementations are also used as target architectures by other researchers in the priority program introduced in the editorial of this issue, e. g., by the group of Prof. Merker [10]. The Verilog implementations are synthesized using commercial tools, and they are simulated and analyzed at the gate-level for detailed evaluations as described in the following sections.

NEC's DRP architecture can also be considered as instance of the CRC model. Its basic functionality complies with the refined CRC model presented in the context of our architecture optimizations in Section 4.3 (Fig. 6). The DRP's control unit ("state transition controller") is a sequencer that controls an array of 8×8 PEs, called a "tile" (Fig. 2). A DRP "core" is composed of one or more of such DRP tiles. The DRP-1 prototype core consists of 8 tiles, i. e., 512 PEs in total. The front-end of the DRP-compiler [12] is a special version of Cyber [14]. It maps the data path extracted from C-descriptions to multiple contexts and generates the code for the control unit.

4 Optimizations

The co-development of architecture and compiler provides a high degree of freedom for optimizations often neglected by architecture and compiler developers. However, it also constitutes a special challenge in addition to the partly conflicting optimization goals (Fig. 3). In the remainder of this section, our approach to address this complex of problems is described.

4.1 Optimization Goals

In order to restrict the enormous optimization space indicated by Fig. 3 reasonably, we use the approach depicted in Fig. 4. First, a performance constraint is set in terms of the throughput rate for a single streaming application or loop kernel. It is specified by the initiation interval, i. e., the number of clock cycles that are available to consume a set of input values, and the clock frequency. The acronym *II* is used in the following to denote the initiation interval.

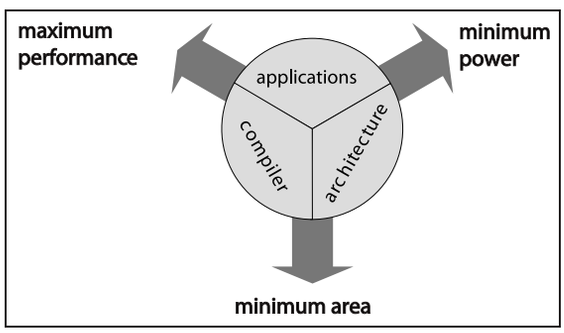

Figure 3 Goals for the co-development of architecture and compiler.

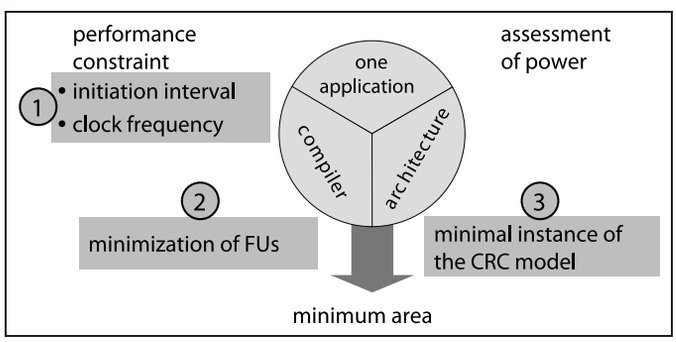

Figure 4 Optimization approach.

Second, the number of required FUs is minimized by the compiler as described in Section 4.2. Third, based on the results of this minimization, an instance of the CRC model with minimal resource requirements is defined as described by the architecture optimizations in Section 4.3.

By comparing the optimized architecture to instances of the CRC model with features related to processor-like reconfiguration (e. g. the number of contexts) being added or removed, the costs and benefits of processor-like reconfiguration are quantified. The results also provide the basis for defining architectures that are optimized for a number of applications rather than a single application [7].

4.2 Compiler Optimizations

The first step in the compiler is generating a combined control and data flow graph (CDFG) from a C-description of the application. The edges of the CFG represent branches in the control flow and its nodes contain the DFGs. The operations of the DFGs are scheduled for execution so that the number of FUs is minimized while meeting the *II* constraint. The corresponding techniques are discussed below. Further steps in the compiler include the binding of operations to FUs of the target architecture (placement) and the binding of the connections between the operations to components of the interconnect network (routing). In cooperation with the research group of Prof. Fekete (priority program project ReCoNodes), techniques for simultaneous scheduling, placement, and routing are developed on the basis of integer linear programming [2].

4.2.1 Implementing Data Flow

To demonstrate the techniques we apply to implement a single DFG being executed in a loop, the Cooley-Tukey algorithm for fast Fourier transform (FFT) is used in the following. Fig. 5a shows the DFG of an 8-point FFT. It is assumed that each of the operations (1–36 in Fig. 5) can be executed by a single FU.

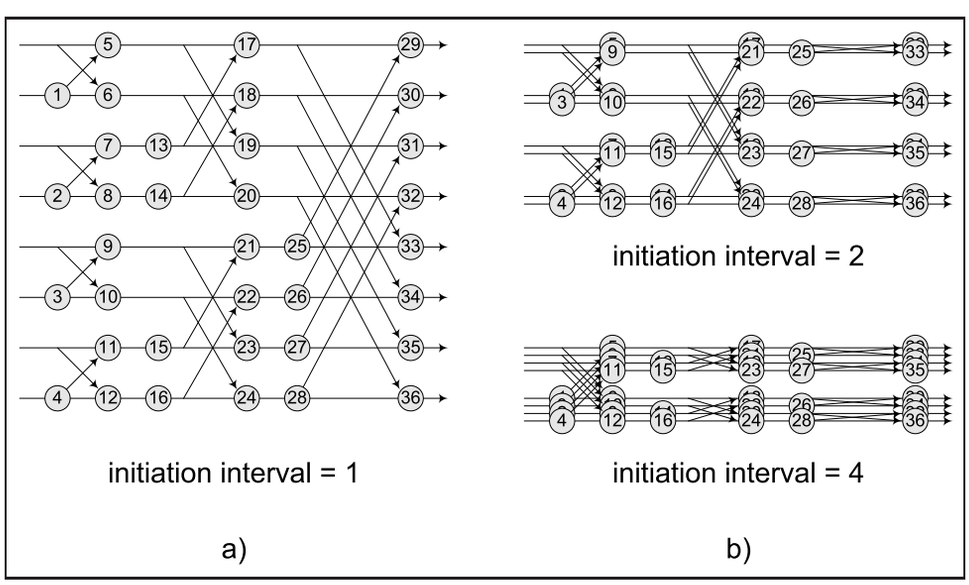

Figure 5 a) DFG of an 8-point FFT. b) Mapping for $II = 2$ and $II = 4$. Overlapping operations can be executed by a single FU in different contexts [3].

To achieve maximum throughput, i. e., $II = 1$ at the highest possible clock speed, the execution must be pipelined by inserting registers between sequential operations of the DFG. Processor-like reconfiguration is not applicable in this case. However, if only a part of the input vector is provided by the surrounding system in each clock cycle ($II > 1$), the FUs can be reused in the various clock cycles within different contexts as indicated in Fig. 5b. To meet the II constraints ($II = 2$ and $II = 4$, again at the highest possible clock speed), the multi-context execution must be combined with pipelining. The resulting multi-context pipelined execution is demonstrated for $II = 2$: in clock cycle $2 \cdot n$, operations 1 and 2 work on their parts of input vector n. At the same time, operations 5–8 work on input vector $n - 1$, operations 13 and 14 on input vector $n - 2$, etc. One clock cycle later, operations 3 and 4 work on their parts of input vector n and operations 9–12 work on input vector $n - 1$, etc.

As seen in Fig. 5, the number of FUs can be reduced significantly when the II is increased. In addition to this, processor-like reconfiguration allows to utilize reconfiguration as a third dimension for routing by redirecting communication through the time domain. By doing so, the number of interconnections, as well as their lengths, can be reduced. For deep sub-micron technologies, the interconnect delay contributes significantly to the overall delay of digital circuits. In addition to the metal wire delays, the reconfigurable routing switches contribute to the interconnect delay of reconfigurable architectures. Therefore, an increased clock speed compared to a statically reconfigurable architecture can be expected for the FFT example if the II constraint is above 1.

Another way to increase the clock speed is inserting additional registers between operations to pipeline communication over the interconnect network. However, as seen in Fig. 5a, pipelining the operations of the FFT example already requires more registers than FUs to keep the pipeline synchronized (e. g., two registers must be inserted between operations 5 and 17), and our experiments targeting NEC's DRP architecture were actually limited by registers rather than by FUs. Moreover, for applications with a feedback loop in the DFG, e. g., if the output of operation 17 were connected to the input of operation 5, pipelining the feedback loop usually decreases the throughput [1]. Therefore, we rather combine chaining of operations with pipelining and multi-context execution.

4.2.2 Implementing Control Flow

Branches in the control flow can be resolved by spatially multiplexing the data path. This transforms the control flow graph of an application into a pure data flow graph that can be further processed as described in Section 4.2.1. This approach is commonly used for reconfigurable architectures.

Alternatively, the branches can be assigned to different contexts, i. e., they are temporally multiplexed, to minimize the number of required FUs without impact on the performance since only one of the branches is actually needed at a time during execution. We call this technique multi-context control flow branches, which corresponds to the conditional jump instructions of von Neumann architectures.

As an extension of the previous technique, we use pipelined multi-context control flow branches. Using this technique, a pipeline stage may change its state and context depending on the results of a previous stage. If the target architecture features only one control unit for all PEs, an excessive number of states can be required to ensure a sustained throughput rate for all combinations of branches in the application. If there is a dedicated control unit for each pipeline stage, the number of states and contexts is minimal, since the possible combinations do not have to be considered at compile time.

4.3 Architecture Optimizations

The scheduling of operations as described in the previous section determines the execution of the application at the register-transfer (RT) level for the given II constraint. The ability to execute the application accordingly depends on the features of the target architecture. At this level of abstraction we therefore refer to the execution as an *execution scheme* [8] and define instances of the CRC model that enable the implementation of an application's execution scheme. The goal of the architecture optimizations is to find an instance that supports the execution scheme using a minimal number of resources.

Based on the execution schemes of several applications, we have refined the CRC model as depicted in Fig. 6. The interconnect network is subdivided into word-wide data channels (8, 16, or 32 bit) and 1-bit status signals not further detailed at this level of refinement. For the FU and the registers, the word-wide data flow is also separated from the 1-bit signals. The output of the FU can be stored in a register and fed into the interconnect network to execute two or more operations in a chain of FUs. For the control unit, the refined model merely specifies that it implements a finite state machine (FSM) controlled by the 1-bit status signals and that an entry of the context memory is selected by the FSM at the beginning of each clock cycle.

The requirements for the control unit and the interconnect network can vary significantly for different applications. Our ongoing research focuses on optimizing these features for a wide range of applications.

For the evaluations in this article we used a parameterizable architecture template based on the refined CRC model that features a lookup-table-based control unit in each

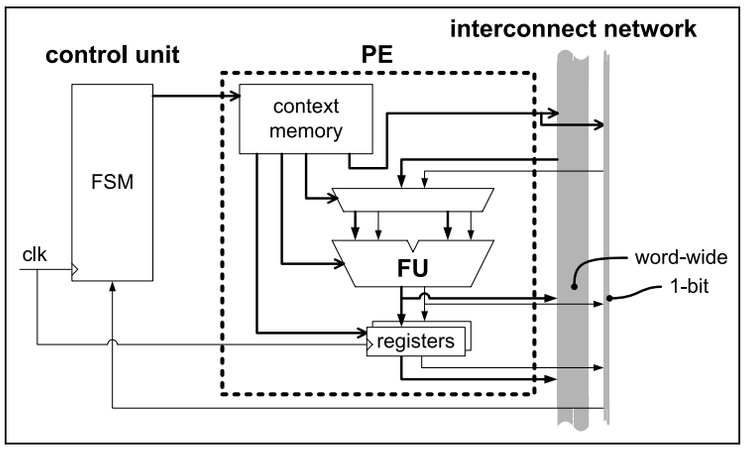

Figure 6 Refined CRC model.

PE and implements a nearest-neighbor (NN) interconnect network. By implementing these features within a single PE, arrays of arbitrary size can be created easily. The NN-network allows routing data between neighboring PEs independently of FU operation. For the applications we have analyzed so far, a NN-network with one status and two data channels that may buffer data independently of the register sets has turned out to yield a good FU to PE ratio. We denote this template as CRC-A since it is the first of a set of architecture templates implemented in Verilog.

For the CRC-A template we have implemented a number of RT-level components (FUs with and without multiplier, NN-networks with one or two data channels, etc.) that can be combined in various ways so that different architecture instances can be synthesized and analyzed in a highly automated flow with reasonable effort. To create architecture instances, in particular instances optimized for an execution scheme, the CRC-A template is configured by selecting RT-level components and setting parameters of the Verilog code (e. g., the word length and the number of contexts). Instances of the CRC model optimized for execution schemes that require no reconfiguration, i. e., statically reconfigurable architectures, are created by omitting the context memory and the control unit.

5 Evaluation

To quantify the costs and benefits of processor-like reconfiguration, area, performance, and power results for data flow applications are presented in the following. Details on control flow branches as well as the applicability of our execution schemes to NEC's DRP architecture are presented in [8].

5.1 Area

To quantify the area trade-off implied by processor-like reconfiguration, a simplified version of the 8-point FFT described in the previous section is discussed. While the Cooley-Tukey algorithm actually requires complex arithmetic, this simplified FFT uses integer arithmetic so that each operation of the DFG can be mapped to an FU of the CRC-A template that supports all operators of the C language except for division and modulo.

Fig. 7 shows area estimations for different instances of the CRC-A template targeting a 130 nm standard cell library. According to our optimization approach, a statically reconfigurable instance with 36 PEs represents the optimal architecture for $II = 1$ (routing is neglected for this area evaluation). Its area is observably smaller than that of processor-like reconfigurable architectures that also require 36 PEs to meet the II constraint.

For $II = 2$, the static architecture still requires 36 PEs while a processor-like reconfigurable instance requires only 20 PEs. Even if the number of contexts is doubled compared to the optimized architecture, the resulting area is smaller than that of the static architecture.

Likewise, an instance with 2 contexts still requires 20 PEs for $II = 4$, since processor-like reconfiguration can only be utilized at every other clock cycle. The optimized architecture with 4 contexts requires only 11 PEs and thereby only about

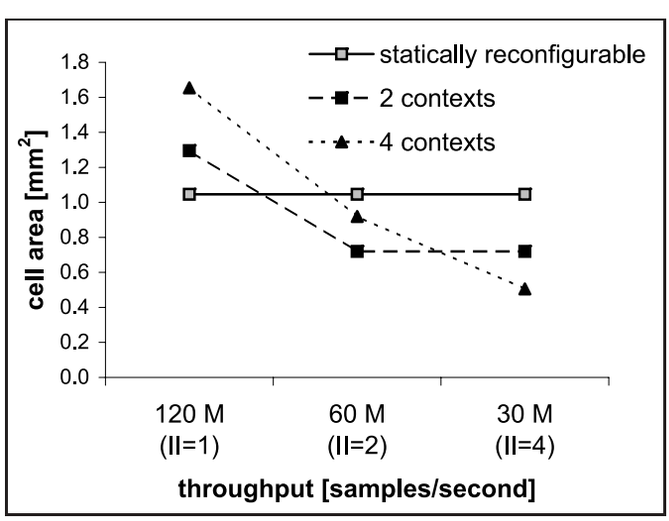

Figure 7 Area vs. performance trade-off for the simplified 8-point FFT and a 120 MHz clock constraint.

half the area of the static architecture.

The area improvements are achieved even though for $II = 2$ and $II = 4$ the operations are not perfectly distributed over the different contexts which actually would require only 18 and 9 PEs. We rather used exactly the schedules proposed by Fig. 5b, which also leads to a performance gain as discussed subsequently.

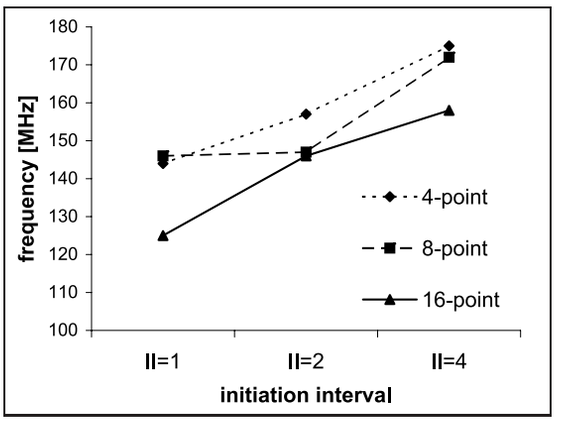

Figure 8 Performance estimations for the simplified FFT.

5.2 Performance

The CRC model allows a detailed comparison of processor-like and statically reconfigurable architectures based on execution schemes as described above. Due to currently restricted routing capabilities of our compiler, only small examples can be mapped completely to the CRC model. We therefore used NEC's DRP architecture to quantify the performance benefits of redirecting communication through the time domain.

Fig. 8 shows the clock frequencies achieved for simplified 4-, 8-, and 16-point FFTs at different *II*s as estimated by the DRP compiler after placement and routing.

The pipelined ($II = 1$) and multi-context pipelined ($II = 2$ and 4) execution was achieved by breaking up data dependencies in the C code of the FFT. However, we did not force a placement of the operations as proposed by Fig. 5. This allows the compiler to optimize the placement within one context, which in turn leads to a mapping of overlapping operations (Fig. 5b) to different FUs possibly far apart. Although this reduces the benefit of redirecting communication through the time domain, one can observe a significant speed-up. This speed-up is independent of placement and solely due to the reduction of routing complexity.

Since the DRP requires less than a nanosecond for selecting a context [12], it can even outperform a statically reconfigurable architecture for $II > 1$. However, it should be noted that for other applications we experienced that the clock frequency may actually decrease if the *II* is increased.

5.3 Power

The area optimizations are based on reusing architectural resources that are not needed concurrently. This reuse also prevents unnecessary circuit switching activities that consume power. In addition to that, the reduction of area reduces leakage currents that contribute significantly to the power consumption of 90 nm and below process technologies.

On the other hand, reconfiguration also consumes power. To evaluate the trade-off, we have mapped a smaller example ($\text{out} = (\text{c1} \cdot \text{in1} + \text{c2} \cdot \text{in2} + \text{c3} \cdot \text{in3} + \text{c4}) \gg \text{c5}$) onto instances of the CRC-A template including routing on a NN-network with one data channel. Fig. 9 shows the results for $II = 1$ and $II = 3$ estimated after the gate-level simulation of processing 100 samples at 200 MHz. The results are based on architecture instances with a minimized number of PEs, but the same results can be achieved for larger arrays by switching off unused PEs.

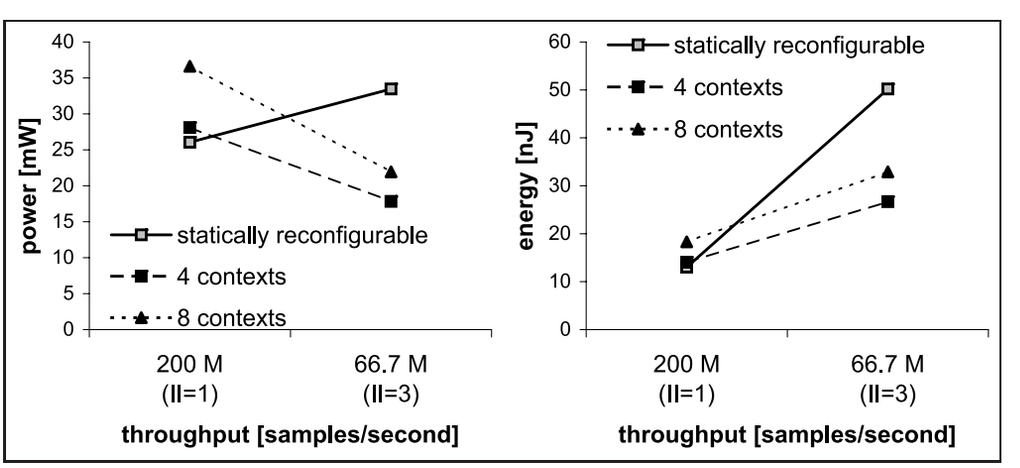

Figure 9 Power and energy estimations for a smaller example processing 100 input samples at 200 MHz (130 nm, 1.2 V).

A statically reconfigurable instance requires 9 PEs (including routing) for both *II*s. For $II = 3$, power increases because data must be buffered for synchronization and, due to the missing control unit, unnecessary switching activity cannot be avoided. Energy consumption increases even more since the execution time increases from 0.5 to 1.5 µs.

An optimized processor-like reconfigurable instance requires only 3 PEs and 3 contexts for $II = 3$, and even instances with 4 or 8 contexts dissipate significantly less power than the static architecture. Since switching contexts consumes power, energy consumption for $II = 3$ is higher than for $II = 1$ but still much lower than for a statically reconfigurable architecture at $II = 3$.

Similar to the area evaluations, such power evaluations require the CRC model since a statically reconfigurable variant of the DRP architecture is not available.

6 Voltage Reconfiguration

If we assume that, for example, the delay of operation 3 in the DFG of Fig. 5 is less than the delay of operation 1, either due to the FU or the interconnect delay, we can exploit the slack time between the two operations to optimize the power consumption by reducing the supply voltage of the faster operation. Although this reduction also increases the delay of the faster operation, it does not affect the overall performance if the delay increase is limited to the delay of the slower operation. We extend the concept of processor-like reconfiguration by applying reconfiguration not only to the function of FU and interconnect network, but also to their supply voltages [11]. This fast voltage reconfiguration mechanism allows the assignment of a desired voltage to each functional unit regardless of spatial position or time. In order to illustrate this idea, we synthesized instances of the CRC-A template considering two voltage levels. Table 1 describes the execution delay of the FU as a function of the operation type and the supply voltage. The slower operation (multiplication) at higher supply voltage determines the minimal clock period (4 ns). Suppose that a multiplication and an addition are executed in the same context and in two different FUs. The FU executing the muliplication must be supplied with high (1.2 V) operating voltage to keep the timing constraint. However, the FU executing the addition can be reconfigured to low (1.0 V) voltage because it takes only 3.5 ns and therefore, it remains within the pre-defined clock period.

Table 1 Delay and power consumption of a FU.

	1.2 V	1.0 V
·	4.0 ns	5.5 ns
+	2.7 ns	3.5 ns
P	3.7 mW	2.3 mW

We see that, if the timing constraint is set between 4.0 and 5.5 ns, it is possible to reduce the total power consumption from 7.4 mW (both FUs function at 1.2 V) to 6.0 mW (one FU functions at 1.2 V, the other at 1.0 V), which is a reduction of 19 percent. The reduction of power consumption is thus accomplished by extending the execution time of the faster operation in the range of the defined timing constraint. Delay and power consumption have been evaluated using commercial synthesis and power analysis tools.

In contrast to other voltage scaling approaches [9], this example shows that it is possible to reduce the power consumption without modifying the clock frequency.

7 Conclusions and Further Work

Processor-like reconfiguration enables to reuse architectural resources efficiently at each clock cycle. Due to I/O constraints or branches in the control flow, it is often not necessary to implement all operations of an application concurrently so that the reuse can be exploited for a wide range of applications and realized by a high-level compiler.

The experiments demonstrate that the overhead imposed by processor-like reconfiguration is low enough to achieve an overall reduction of area and power. By redirecting communication through the time domain, even the performance can be increased compared to a statically reconfigurable architecture. Furthermore, the concept of processor-like reconfiguration cannot only be applied to the functionality of the architectural resources, but also to their supply voltage to optimize power.

Our ongoing work focuses on exploring architectures optimized for an application domain rather than a single application [7], since reconfigurable architectures are usually not intended to be used for just a single application. Thereby we also consider the integration of processor-like reconfigurable cores in a system-level environment (System-on-a-Chip) where, e. g., contexts are reloaded from outside the array to set up the execution of a new application while another application is still being executed using the entire array.

Another core theme of our ongoing research is the co-development of architecture and compiler with respect to the interconnect network. On the compiler side, we develop techniques that combine scheduling, placement, and routing, taking into account the interconnect delay which contributes significantly to the overall performance. The modifiable CRC model provides the required delay estimations and allows us to develop networks that can be utilized efficiently by the compiler.

References

[1] S. Alexander and R. Stewart: The effects of pipelining feedback loops in high speed DSP systems. In: IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing (ICASSP), 2005.

[2] J. Brenner, J. van der Veen, S. Fekete, J. Oliveira Filho, and W. Rosenstiel: Simultaneous scheduling, binding and routing for processor-like reconfigurable architectures. In: Int'l Conf. on Field Programmable Logic and Applications (FPL), 2006.

[3] H. Claßen: Evaluation of the FFT on CRC. Student research project, University of Tübingen (WSI/TI), 2005.

[4] A. DeHon: DPGA utilization and application. In: Int'l Symp. on Field-Programmable Gate Arrays (FPGA), 1996.

[5] B. Mei, S. Vernalde, D. Verkest, H. DeMan, and R. Lauwereins: Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. In: Design, Automation and Test in Europe (DATE), 2003.

[6] M. Motomura: A dynamically reconfigurable processor architecture. In: Microprocessor Forum (MPF), 2002.

[7] J. Oliveira Filho, T. Schweizer, T. Oppold, T. Kuhn, and W. Rosenstiel: Tuning coarse-grained reconfigurable architectures towards an application domain. In: Int'l Conf. on Reconfigurable Computing and FPGAs (ReConFig), 2006.

[8] T. Oppold, S. Eisenhardt, and W. Rosenstiel: Design and validation of execution schemes for dynamically reconfigurable architectures. In: Int'l Conf. on Field Programmable Technology (FPT), 2006.

[9] T. Pering, T. Burd, and R. Brodersen: Voltage scheduling in the lpARM microprocessor system. In: Int'l Symp. on Low-Power Electronics and Design (ISLPED), 2000.

[10] M. Rullmann, S. Siegel, R. Merker, J. Oliveira Filho, T. Schweizer, T. Oppold, and W. Rosenstiel: Efficient mapping and functional verification of parallel algorithms on a multi-context reconfigurable architecture. In: Architecture of Computing Systems (ARCS), 2007.

[11] T. Schweizer, J. Oliveira Filho, T. Oppold, T. Kuhn, and W. Rosenstiel: Evaluation of temporal-spatial voltage scaling for processor-like reconfigurable architectures. In: Euro DesignCon, 2005.

[12] T. Toi, N. Nakamura, L. Jing, Y. Kato, T. Awashima, and K. Wakabayashi: High-level synthesis challenges and solutions for a dynamically reconfigurable processor. In: Int'l Conf. on Computer-Aided Design (ICCAD), 2006.

[13] S. Trimberger, D. Carberry, A. Johnson, and J. Wong: A time-multiplexed FPGA. In: IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM), 1997.

[14] K. Wakabayashi and T. Okamoto: C-based SoC design flow and EDA tools: An ASIC and system vendor perspective. In: IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 19, no. 12, pp. 1507–1522, 2000.

Shown from left to right:
Thomas Schweizer, Julio Oliveira Filho, Wolfgang Rosenstiel, Sven Eisenhardt, Tobias Oppold

1 Dipl.-Inform. Sven Eisenhardt, Julio Oliveira Filho, MSc. in Computer Science, Dipl.-Inform. Tobias Oppold, Prof. Dr. Wolfgang Rosenstiel, Dipl.-Inform. Thomas Schweizer The authors are members of the Department of Computer Engineering at the University of Tübingen. Their research interests are in the field of embedded systems and electronic design automation with special focus on reconfigurable systems, verification, synthesis, parallel computing, artificial neural networks, and organic computing. Address: Eberhard-Karls-Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, Sand 13, 72076 Tübingen, Tel.: +49-7071-2978956, Fax: +49-7071-295062, E-Mail: {eisenhar, oliveira, oppold, rosenstiel, tschweiz}@informatik.uni-tuebingen.de