

# A Design Environment for Processor-Like Reconfigurable Hardware

T. Oppold, T. Schweizer, T. Kuhn, W. Rosenstiel  
University of Tuebingen  
Wilhelm-Schickard-Institute  
Sand 14, 72076 Tuebingen, Germany  
crc@informatik.uni-tuebingen.de

## Abstract

*There is a growing number of reconfigurable architectures that combine the advantages of a hardwired implementation (performance, power consumption) with the advantages of a software solution (flexibility, time to market). Today, there are devices on the market that can be dynamically reconfigured at run-time within one clock cycle. But the benefits of these architectures can only be utilized if applications can be mapped efficiently. In this paper we describe a design environment that takes into account the three aspects architecture, compiler, and applications, and we present the basic techniques that we use to realize the compiler.*

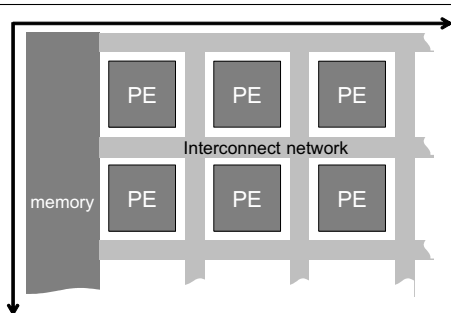
## 1. Introduction

A lot of research in the area of reconfigurable computing systems is focused on the re-use of devices like FPGAs for different applications. But compared to this, newly developed devices provide much more advantages. They allow to re-use the functional elements of a device for different operations within a single application. This is accomplished by extremely fast reconfiguration (in less than a clock cycle) during run-time. For these devices, frequent reconfiguration is part of the regular execution. The reconfiguration keeping pace with the execution yields an additional degree of freedom that constitutes a new principle of reconfiguration. We name this new principle processor-like reconfiguration. By means of reconfigurable datapaths, processor-like reconfiguration allows to instantiate and execute within one clock cycle exactly that part of a circuit that is needed in this cycle. Although for each of the new architectures distinct advantages have been presented, by today none of them was able to gain significant market shares in industry. From the technical point of view, our practical experiences have shown the following reasons for this: 1) the design tool to map applications onto the architecture (the com-

piler) is developed after the architecture was defined, 2) the language provided to users for application design is not appropriate, or 3) suitable applications are sought after the architecture was developed. This demonstrates that a good architecture alone does not provide a good solution. In fact, the benefits of reconfigurability can only be exploited if applications can be mapped efficiently onto the available architectures.

In this context we focus on two problems that need to be solved. The first problem is that different applications need different architectures. This is due to the fact that the area of *Reconfigurable Computing* mostly stresses the use of coarse grained architectures [7]. Accordingly, the commercial architectures are also coarse grained. For such architectures various properties (e. g. the datapath width) are fixed at fabrication time of the device. This means that only applications that fit this properties can be mapped onto this architectures without wasting a significant amount of resources. The second problem that we focus on is the availability of a compiler to map applications onto the architecture. Such a compiler must be able to automatically map applications that are described on high levels of abstraction, i. e., the algorithmic and system level. In our opinion application design at the register transfer (RT) level is not an option. In ASIC/FPGA design the implementation at the RT level already leads to a productivity gap between the number of gates provided by modern process technologies and the number of gates that can be implemented by application design teams. Reconfiguration yields a further degree of freedom that increases design complexity even more.

Our approach to solve these problems is a design environment for reconfigurable architectures that takes into account the three aspects architecture, compiler, and applications. The environment supports an iterative flow where the architecture and the compiler are refined concurrently. This refinement is driven by the analysis of applications from a given problem domain (e. g. encryption or graphical systems). The result of the iterative flow is an architecture together with a compiler which are optimized for the speci-



**Figure 1. CRC model.**

fied problem domain.

As the starting point for the architecture we have defined the CRC model (*Configurable Reconfigurable Core*) depicted in figure 1. The CRC model is a very general model for processor-like reconfigurable architectures that can be configured with a variety of parameters during the refinement process. This configuration and the reconfiguration of the architecture during run-time account for the name CRC. It is oriented towards commercially available architectures to take into account the technological feasibility but it leaves sufficient space for optimization by abstracting from the real architectures. The CRC model consists of processing elements (PE), a reconfigurable interconnect network, and memory. Each PE consists of a functional unit (FU) that is able to perform arithmetic and logic operations, a register set, and a context memory. An entry in the context memory determines the context of the current clock cycle. The context determines the operation performed by the FU, a register from the register set, and the connections between the PEs. The context can be switched on a cycle-by-cycle basis. Since the CRC model is firstly a theoretical model these resources are not constrained. Within the design environment instances of the CRC model are created. These instances are real architectures with limited resources.

In order to take advantage of reconfigurability a proper temporal partitioning of an application must be found. Research activities in the last years have shown that such a partitioning is hard to find if reconfiguration is expensive in terms of time, area, or power consumption. The properties of processor-like reconfigurable architectures allow us to make use of the well known techniques from C-based synthesis. To gain acceptance in industry the compiler must support an input language that is well adopted. Today, C is widely used for algorithm and embedded system design. Hence, we use C as the language for application design in our approach and consider application domains where compute intensive parts of an applications are moved from software to hardware to meet performance or power constraints. Object oriented languages like C++ and Java are extensions of C and therefore we are well prepared for a migration to

object oriented design that is part of our research on reconfigurable architectures [9].

The paper is organized as follows. In the next section we present work that is related to our project. The design environment is described in section 3, basic techniques that we apply to map applications are described in section 4. First instances of the CRC model are presented in section 5 along with characteristics in terms of speed, area, and power. Section 6 concludes this paper and provides an outlook on further work in the project.

## 2. Related Work

We see our design environment as an addition to the work done by industry and other academic research groups. We integrate features of a wide range of reconfigurable architectures and benchmark and compare these features. The results are used in our design flow to optimize existing architectures and to propose new architectural features.

For instance, NEC's DRP architecture is closely related to our architecture model. The DRP was architected based on a clear picture of how C code is compiled into hardware [11] and accordingly the compiler is based on NEC's C-based behavior synthesizer Cyber [15]. Other commercial architectures that influence our architecture model include PACT XPP [3], IPFlex DAP/DNA [8], PicoChip's picoArray [12], and Quicksilver's ACM [13]. Besides these relatively new architectures that are usually labeled as (dynamically) reconfigurable processors we also consider properties of traditional architectures like FPGAs, DSPs, and VLIW processors.

In the academic field, DeHon has done some fundamental work on optimizing area and performance of reconfigurable architectures [5]. He proposed a variety of highly reconfigurable architectures including DPGA and MATRIX that are very valuable for our approach. Morphosys [10] is another architecture that is related to our architecture model. An overview of reconfigurable devices that more or less relate to our work can be found, e. g., in [4].

All architectures mentioned above strongly influence the design of our CRC model. But we do not solely assess the architecture. We also take into account the compiler and the applications that are to be mapped onto the architecture. The concurrent refinement of architecture and compiler plays a prominent role in our approach. In that respect the Teramac project [2] is similar. The project targets logic simulation and to avoid compilation times of many days the architecture was designed according to the needs of the compiler. Teramac does not consider fast reconfiguration during run-time though.

Besides the usually well researched optimization of performance and area, power consumption is a major issue in many of today's and future designs. Reconfigurable devices

have the potential to provide an excellent performance to power ratio and power dissipation is an important criterion in our design environment. In [1], for instance, techniques to optimize power consumption of reconfigurable architectures are proposed. Today, power consumption is mostly considered at relatively low levels. For these levels numerous commercial analysis and estimation tools exist that support optimization of power consumption. In [14], power optimization techniques are considered at high levels of abstraction in hardware design. For microprocessors power consumption is also optimized mostly at the architectural level and compiler techniques are rather in the realm of academic research. In our design flow we consider power at the architectural level as well as in the compiler.

### 3. Design Environment

Within our design environment we not only explore different architectures but we vary the applications and the compiler as well. This leads to a very complex design space that needs to be searched. Of course, it would be desirable to have a system that reads in a set of applications and automatically generates an architecture along with a compiler that are optimized for the given applications. Given that processor-like reconfigurable architectures are relatively new and it is not clear yet, for example, what is the best way to map applications or what are the right application domains we see this as an unrealistic goal for the near future. We therefore require user interaction in the design flow. Especially in the early phase of the project the parameters by which the CRC model can be configured are not only scalar values like, e. g., the number of contexts or the width of the datapath. In fact, the design team of the compiler reports bottlenecks or proposes improvements of the architecture and this can require a new architecture model that incorporates entirely new features. For instance, if the compiler is not able to map loops efficiently due to missing wiring resources a new interconnect network with fast feedback wires is created.

Figure 2 depicts the flow in the design environment. The first step is to select an application from an application domain. We mainly consider applications in the area of mobile communication devices and graphical interactive systems. In these areas compute intensive applications can be found that are good candidates for an implementation on reconfigurable architectures. In a first approach we define application domains within these areas by the following criteria: datapath versus control oriented applications, applications that benefit from specialized operations (e. g. saturated arithmetic), and single-threaded versus multi-threaded applications. Support of multi-threaded applications is important for system level design and scheduling of multiple threads is a very good match for processor-like reconfigu-

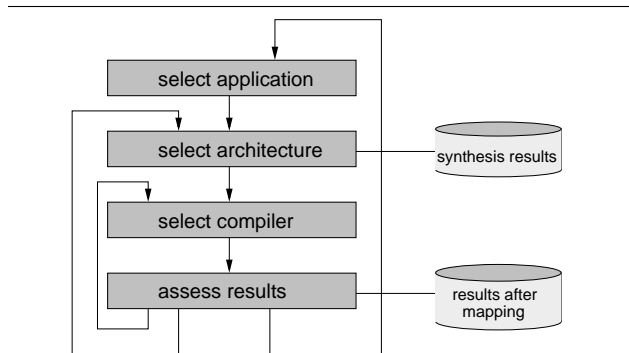


Figure 2. Flow in the design environment.

ration since threads can be scheduled on the basis of clock cycles with very little overhead.

The selected application is at first translated into an architecture independent format. This format unveils basic information about the application like the data types used or the memory accesses. Based on that information we manually select an instance of the CRC model that fits the application already to some extent because it provides the right datapath width etc. The delays of the various components of this particular instance that are needed for C-based application mapping as well as an area estimation are obtained from synthesizing the instance with a commercial tool. To avoid unnecessary synthesis runs we maintain a database with the results of all synthesized instances.

In the next step the application is mapped onto the architecture. This is done by selecting an instance of the compiler. An instance of the compiler is defined by the optimizations and compile strategies that are applied during the application mapping. Application mapping is further described in the following section.

After having performed the application mapping it is known at what clock speed the application can run and how many clock cycles are needed to execute it. The results for all properties (energy consumption, area, and performance) of the application mapped by this specific compile run onto this specific instance of the CRC model are kept in a database for comparison of the results. At this stage in the flow we also have detailed information about the utilization of the architecture's resources. Based on this information we try other compile strategies as well as other instances of the CRC model in order to optimize one of the properties while meeting constraints for the remaining properties.

After an architecture-compiler pair is found that satisfies the requirements for this specific application another application from the application domain is selected and the same steps are performed for this application. This time the initial selection of the architecture can be performed based on architectures found in previous iterations. After having processed a number of representative applications of a domain

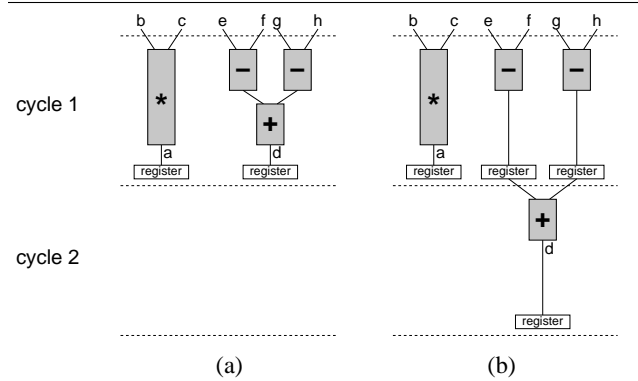
we assess the flexibility of each of the architecture-compiler pairs by mapping all applications again using this specific pair. If there is an architecture-compiler pair that is able to meet the constraints for all the applications this is the final result of the flow. If such a pair was not found new architecture-compiler pairs can be tried. If this also fails because the requirements of the applications are too diverse the application domain must be redefined.

#### 4. C-Based Application Mapping

C-based synthesis translates a behavioral C description at the algorithmic level into a structural description at the register transfer level. The resulting structural description consists of a temporally partitioned datapath and a control unit. The datapath is a set of connected components and the control unit steers the functioning of these components. When an ASIC/FPGA is targeted the structural description is further processed by logic synthesis tools. For processor-like reconfigurable architectures the different partitions of the datapath can be mapped to different contexts and each of the contexts is able to implement exactly that part of the datapath that is needed in this specific clock cycle in an ASIC-like fashion.

In our application mapping process, the first step is the compilation of the C description into an intermediate format that is independent of the target architecture. This intermediate format is a simple three address code that is more suitable for the subsequent steps in the mapping process. Before any architecture specific transformations are applied several transformations on the intermediate format can be performed. These transformations comprise techniques that are also applied by software compilers, e. g., common sub-expression and dead code elimination, constant propagation, loop invariant code motion and loop unrolling. Depending on the property that is to be optimized some transformations may or may not lead to an improvement of the overall mapping. The architecture independent transformations therefore contribute to the attributes that distinguish the instances of the compiler in the design environment.

Processor-like reconfigurable architectures allow to map an application both temporally and spatially. In order to utilize the spatial features parallelism must be extracted from the application description. We currently focus on instruction level parallelism (ILP) since we consider loop level and thread level parallelism, which are both also in the scope of the CRC model, an extension to ILP. But parallel execution of instructions is only one dimension of spatial application mapping. The second dimension can be exploited by operator chaining which is not possible for superscalar microprocessors like VLIW processors. The effect of operator chaining is illustrated in figure 3. The two arithmetic expressions  $a = b * c$  and  $d = (e - f) + (g - h)$  can be executed in

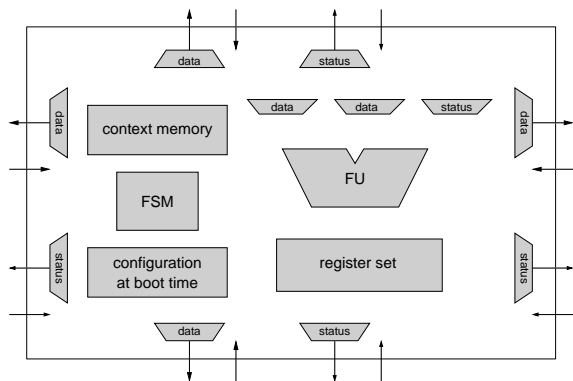


**Figure 3. Computation (a) with and (b) without operator chaining. The height of the operation boxes corresponds to the delay of the operation.**

one clock cycle if operator chaining is performed. Without chaining two cycles are needed.

Several scheduling algorithms that consider parallel execution and operator chaining have been proposed in the area of C-based synthesis. These scheduling algorithms take a data flow graph as input and are aware of the number of functional units and the time they need to complete an operation. Such a data flow graph can be found in the basic blocks of the intermediate format. Basic blocks are code blocks that do not have any control structures. But often, one basic block does not provide the high degree of parallelism that can be handled by reconfigurable architectures. To overcome this problem multiple basic blocks can be merged into one hyperblock enabling speculative parallel execution of different control branches. This usually yields a performance gain but also tends to increase power consumption and area. The scheduling algorithms that we implement in our compiler for processor-like reconfigurable architectures are based on our synthesis tool CADDY [6]. It is our goal to implement proven and also new scheduling algorithms in our compiler and to benchmark the results in particular in terms of power consumption.

If a schedule for the operations is found the operations must be assigned to PEs and the PEs must be properly connected using the reconfigurable interconnect network. At present, we do this manually after scheduling which is feasible for the small examples that we use to conduct first experiments. For a future implementation of the compiler we work on incorporating these steps into the scheduling process since these tasks can strongly influence each other.



**Figure 4. Overview of a PE of the first instances of the CRC model.**

## 5. Results

As mentioned in the previous section the compiler generates a temporally partitioned datapath and a control unit. While the different partitions of the datapath can be implemented by the FUs inside the PEs of the CRC model efficiently, this general model does not provide a special mechanism to realize the control unit. It is an important feature of our design environment to modify the architecture according to the characteristics of the compiler. We therefore defined a set of instances of the CRC model that feature a configurable finite state machine (FSM) to implement the control unit. This set of instances is realized by a synthesizable Verilog model that consists of uniform PEs (see figure 4) that are connected by a nearest neighbor interconnection network. The memory is omitted in this first instances and we decided to integrate a separate FSM within each PE. We experienced that an FSM that is shared by multiple PEs is actually more convenient for application mapping but the absence of features that are shared among multiple PEs makes our first estimations more scalable.

A mechanism to configure the context memory and the FSM from outside at boot time of the device, the separation of data and status signals, and multiplexers to select the data and status input signals for the FU are further features that are not defined in the general CRC model but are implemented in the presented set of instances. To obtain a specific instance of the CRC model the Verilog model must be configured with the following parameters: operators supported by the FU, number of data and status registers, width of the datapath, number of contexts, and number of states.

To perform the architecture specific tasks in the compilation process it is mandatory to have detailed cost functions for the target architecture. In the following we present results from area, speed, and power estimations that we obtained by synthesizing components of instances of the CRC

model described above. The operators supported by the FU are  $*$ ,  $+$ ,  $==$ ,  $<$ , AND, OR, NOT, and SEL. SEL selects one of the data inputs of the FU depending on the status input. The multiplier supports input operands with half the word length of the datapath so that the result fits the datapath width. This set of operators allows to run a few simple applications to functionally verify the model. Other C operators like  $-$ ,  $<=$ , XOR etc. can be derived from the operators or can be added without significantly affecting the cost functions. The number of data and status registers is set to twelve for all instances. The datapath width, the number of contexts, and the number of states varies for the different instances. The synthesis was done using a commercial logic synthesis tool and a 0.13 micron standard cell technology library.

**Area:** To perform the area estimations we have set the number of contexts and the number of states to 16 which we consider reasonable. For this setup the arithmetic and logic operations excluding multiplication use only a small area even for a relatively wide datapath (32 bits). The multiplier allows only input operands of half the word length but still uses a lot more area than the remaining operations together. It should therefore be well considered if multipliers are actually needed in a domain specific architecture.

Area estimations for varying numbers of states and contexts and different datapath widths have shown that the reconfigurable interconnect network, i. e. the multiplexers together with the entries in the context memory that are needed to control the multiplexers, constitute the highest potential to optimize the area.

**Speed:** The arithmetic and logic functions of the FU by itself provide full ASIC speed. But the FU also contains logic to select different functions in the different contexts. Taking into account both factors we have experienced that for arithmetic operations the CRC architecture is about ten times faster than a Xilinx Virtex-II FPGA. E. g., a 32-bit addition is done within 3.25 ns by the CRC architecture while it takes 35.76 ns on the FPGA. For logic operations the speed up compared to the FPGA is only small because the function selection dominates the delay of the very simple logic functions. Another factor that must be considered is the time needed to generate a new context. It is composed of the time needed to determine the next state, the time to select a context according to the new state, and the delay of the context memory. This time totals to 1.82 ns to 2.45 ns depending on the number of states and contexts.

**Power:** For the power estimations the datapath is set to a width of 8 bits. The number of states and contexts is set to 16. The figures for power dissipation that we present are obtained from the static estimations of the logic synthesis tool. Power dissipation highly depends on the switching activity of a circuit. Based on the synthesis tool's default values for switching activity a PE dissipates 1.34 mW when running

at a clock frequency of 100 MHz. For a clock frequency of 50 MHz a power dissipation of 0.68 mW is estimated. The power dissipation at 50 MHz is a little more than half the power dissipation at 100 MHz. This is due to the static leakage power of 0.034 mW which is independent of the clock frequency. The mechanism to configure the context memory and the FSM from outside at boot time is estimated to dissipate almost one third of the total power. But this part of the circuit is actually not needed during normal operation. We therefore work on instances of the CRC model where switching activity for that part is eliminated after booting the device in order to significantly reduce power dissipation. Besides that, the multiplexers together with the context memory, i. e. the reconfigurable interconnect network, provides the highest potential to reduce power dissipation. This complies with the measures to reduce the area.

The estimated total power dissipation indicates that an array of tens to hundreds of PEs would have a power dissipation that is comparable to an embedded low-power microprocessor. But we are aware that the static power estimation is only a rough estimation. We currently work on incorporating the switching activity that actually occurs when an application is running on the architecture. This more accurate modeling is not only necessary to optimize the power dissipation of the architecture. It is also an important prerequisite to explore power saving techniques in the compiler.

## 6. Conclusions and Further Work

One of the first steps in our design flow was the implementation of a synthesizable architecture model that fits the basic features of C-based application mapping. In contrast to other projects that are engaged in the design of reconfigurable architectures we optimize the architecture according to the capabilities of a high level compiler and the requirements of the applications right from the start. With the area, speed, and power estimations from a commercial logic synthesis tool we can carry on with the development of the compiler that needs realistic values for the characteristics of the architecture. The speed estimations demonstrate that the overhead imposed by processor-like reconfiguration is reasonable and that for applications that feature a high rate of arithmetic operations a significant speed up compared to FPGAs can be achieved. This kind of applications can be found in many application domains that we target. In order to also take into account the wire delays that are an important factor for deep sub micron technologies we extend our current synthesis flow to the steps of place and route. Further work also includes enhancements of the architecture like integration of memory blocks and more sophisticated interconnect networks.

## 7. Acknowledgments

This work is funded by DFG project No. RO-1030/13 within the "Priority Program 1148" which is focused on reconfigurable computing systems.

## References

- [1] A. Abnous and J. Rabaey. Ultra-low-power domain-specific multimedia processors. In *IEEE VLSI Signal Processing Workshop*, 1996.
- [2] R. Amerson, R. Carter, B. Culbertson, P. Kuekes, and G. Snider. Teramac—configurable custom computing. In *IEEE Workshop on FPGAs for Custom Computing Machines*, 1995.
- [3] V. Baumgarte, F. May, A. Nüchel, M. Vorbach, and M. Weinhardt. PACT XPP - a self-reconfigurable data processing architecture. In *International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2001.
- [4] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, June 2002.
- [5] A. DeHon. Reconfigurable architectures for general-purpose computing. Technical Report AITR-1586, MIT Artificial Intelligence Laboratory, Sept. 2 1996.
- [6] P. Gutberlet and W. Rosenstiel. Scheduling between basic blocks in the CADDY synthesis system. In *European Conference on Design Automation*. IEEE Computer Society Press, 1992.
- [7] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Design Automation and Test in Europe*, 2001.
- [8] IPFlex, Inc. DAP/DNA Overview. <http://www.ipflex.com/english/product/index.html>.
- [9] T. Kuhn, T. Oppold, M. Winterholer, W. Rosenstiel, M. Edwards, and Y. Kashai. A framework for object oriented hardware specification, verification, and synthesis. In *Design Automation Conference*, 2001.
- [10] G. Lu, H. Singh, M. Lee, N. Bagherzadeh, F. Kurdahi, E. Filho, and V. Castro-Alves. The morphosys dynamically reconfigurable system-on-chip. In *The First NASA/DoD Workshop on Evolvable Hardware*, 1999.
- [11] M. Motomura. A dynamically reconfigurable processor architecture. In *Microprocessor Forum*, 2002.
- [12] picoChip Designs Limited. <http://www.picochip.com>.
- [13] QuickSilver Technology, Inc. <http://www.qstech.com>.
- [14] A. Stammermann, L. Kruse, W. Nebel, A. Pratsch, E. Schmidt, M. Schulte, and A. Schulz. System level optimization and design space exploration for low power. In *International Symposium on System Synthesis*, 2001.
- [15] K. Wakabayashi. C-based synthesis experiences with a behavior synthesizer, "Cyber". In *Design, Automation and Test in Europe*, 1999.