

Euro DesignCon 2004

A New Design Approach for Processor-Like Reconfigurable Hardware

Tobias Oppold, Thomas Schweizer, Tommy Kuhn,
Wolfgang Rosenstiel
Tübingen University
Wilhelm-Schickard-Institute, Computer Engineering,
Sand 14, 72076 Tübingen, Germany
crc@informatik.uni-tuebingen.de

Abstract

There are 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 approach for reconfigurable architectures that takes into account the three aspects architecture, compiler, and applications. To realize the proposed design flow we developed a synthesizable architecture model. For this model we present area, speed, and power estimations.

Author(s) Biography

Tobias Oppold works as a research assistant in the Computer Engineering Department of the Tübingen University. There he is part of the Hardware Synthesis Group and his research topics are reconfigurable systems and object oriented and high level synthesis.

Professor Wolfgang Rosenstiel received his Ph.D. in 1984 from the Karlsruhe University. Since 1990 he is Professor (Chair for Computer Engineering) at the Wilhelm-Schickard-Institute for Informatics (WSI), Tübingen University, as well as Director of FZI Department "System Design in Microelectronics". He is member of the Editorial Board of several journals. He was general and programme chair of several international conferences like HLDTV, EURODAC, VLSI, EDAC, HLSW and others. He is member of the Executive Committee of DATE and ICCAD, and TPC member of many conferences and workshops. He is member of GI, IEEE, and IFIP 10.5. He is on the executive board of the German edacentrum.

His special interests are in electronic design automation, especially synthesis, co-design, verification, and modelling, computer architecture and artificial neural networks.

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 re-using 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 data paths, processor-like reconfiguration allows it to instantiate and to 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 compiler) 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 [1]. Accordingly, the commercial architectures are also coarse grained. For such architectures various properties (e.g. the data path width) are fixed at fabrication time of the device. This means that only applications that fit these properties can be mapped onto these 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 specified problem domain (see figure 1.)

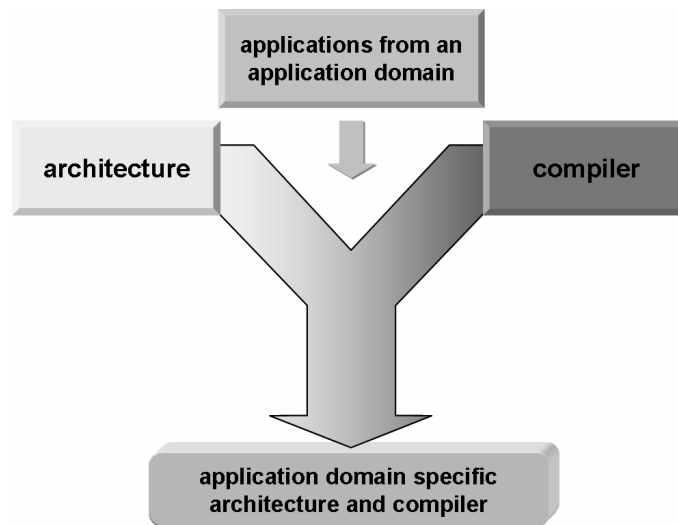


Figure 1: Concurrent refinement of architecture and compiler.

As the starting point for the architecture we have defined the CRC model depicted in figure 2. 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 (Configurable Reconfigurable Core). 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), an 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 application 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 [2].

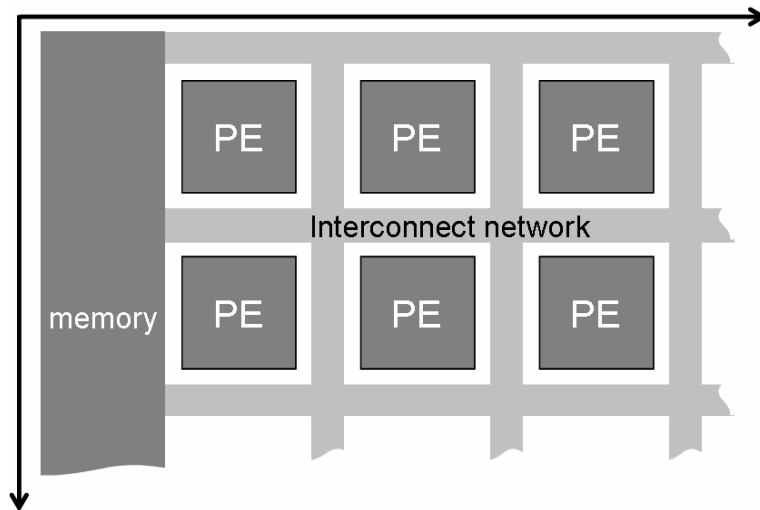


Figure 2: CRC model.

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. Speed, area, and power estimations for our architecture model are presented in section 4. Section 5 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 [3] and accordingly the compiler is based on NEC's C-based behavior synthesizer Cyber [4]. Other commercial architectures that influence our architecture model include PACT XPP [5], IPFflex DAP/DNA [6], PicoChip's picoArray [7], and Quicksilver's ACM [8]. Besides these relatively new architectures, which are usually labeled (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 [9]. 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 [11].

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 [12] 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 present 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 [13], for instance, techniques to optimize power consumption of reconfigurable architectures are proposed. The commercial tools available today for hardware design consider power consumption mostly at relatively low levels. 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

In this section we present the design environment that we use to realize the flow depicted in figure 1. Within the 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 data path. 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.

In the following four subsections we first describe the flow in the design environment and then provide the basic concepts of the different aspects in the design environment.

3.1. Design Flow

Figure 3 depicts the flow in the design environment. The first step is to select an application from an application domain. This application is 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 data path 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.

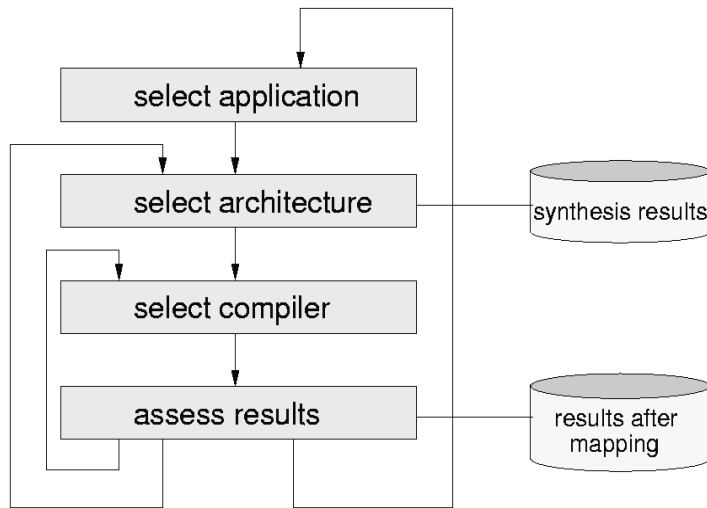


Figure 3: Flow in the design environment.

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 the compile strategies that are applied during the application mapping. Since the area is constrained by the selection of the architecture the compiler tries to find a mapping that yields the best performance or the lowest energy consumption. If optimization of the performance is desired the strategy of the compiler is to execute as many operations in parallel as possible. If not all resources are used up by the operations that can be executed in parallel operator chaining is performed. If all resources are used up or the delay of the data path reaches a limit that was set by the user a context switch is performed. If optimization of the energy consumption is desired the strategy is not as clear as in the case of performance. We perform high-level transformations like loop unrolling as well as different strategies for scheduling and placement of operations and routing of the signals. Eventually, our goal is to find transformations and strategies for the compiler that yield low energy consumption for processor-like reconfigurable architectures. Our

current approach is to simulate the architecture model with the application running on it with a commercial hardware simulator that evaluates the switching activity of the circuit. This allows benchmarking the different strategies in terms of energy consumption.

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 instance of the compiler 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.

3.2. Compiler

To overcome the problem of having a good architecture but not an appropriate compiler we assess the prospects for application mapping very early in the flow. To enable high productivity of application designers we focus on an automated application mapping of algorithmic descriptions. C-based synthesis transforms an algorithmic description into a temporally partitioned data path and a control unit. The different partitions of the data path can be mapped to different contexts and each of the contexts is able to implement exactly that part of the data path that is needed in this specific clock cycle in an ASIC-like fashion. Exploiting parallelism is a key technology to keep the clock frequency and power consumption low and to achieve a significant improvement over a sequential mapping as it is the case when a microprocessor is the target architecture. By using C-based synthesis as the starting point for the compiler we can deploy well known techniques to extract parallelism from high level descriptions. This allows us to focus on the aspect of reconfiguration that is not considered in conventional C-based synthesis. To perform the initial steps in our design flow we used a slightly modified version of our synthesis tool CADDY II [15].

3.3. Architecture

The first instances of the CRC model that we implemented consist of uniform PEs (see figure 4) that are connected by a nearest neighbor interconnection network. The memory

is omitted in these first instances. To implement the control unit generated by the compiler efficiently each PE features a configurable finite state machine (FSM). 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. The FU has two data input ports and one input port for a 1-bit status signal. The output ports comprise one data and one status port. The status signal is used for carry signals and for the results of compare operations. The register set has separate registers for data and for status signals. Multiplexers are used to implement the nearest neighbor interconnection network and to select the data and status input signals for the FU. The data and status signals can be routed independently and a PE can simultaneously perform operations in the FU and route data and status signals from one neighbor to another. The data/status output of the FU can be stored in one of the data/status registers and it can be routed to one of the four neighboring PEs to perform operator chaining. Likewise, the data/status input for the FU can be selected from the register set and from the four neighboring PEs. The multiplexers and the selection of the operation to be performed by the FU are controlled by the content of the context memory. The FSM selects an entry of the context memory depending on the current state. The state transitions of the FSM are controlled by the status signals. To configure the context memory and the FSM from outside at boot time of the device a shift register and additional logic is part of each PE. A set of instances that only differ in scalar parameters that can be set during synthesis is called a type. To obtain one specific instance of the CRC model the type described above needs to be configured with the following parameters: operators supported by the FU, number of data and flag registers, width of the data path, number of contexts, and number of states.

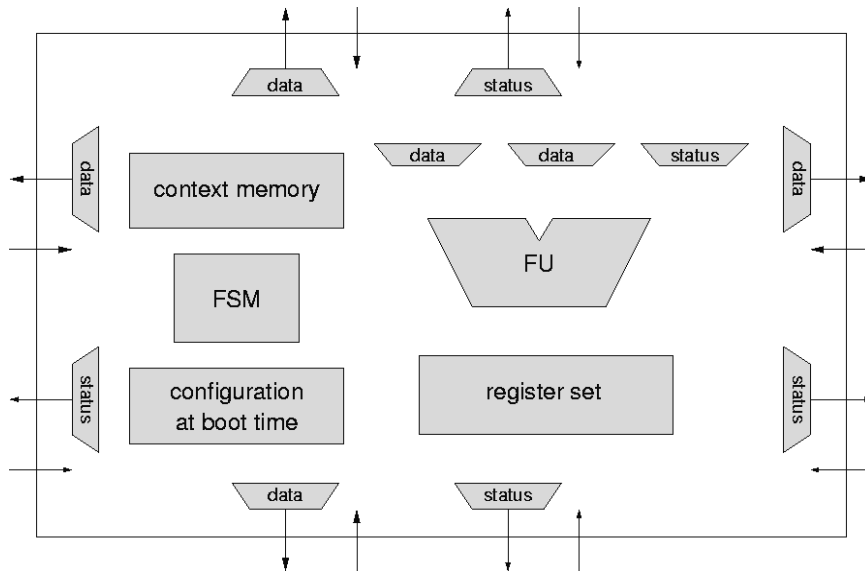


Figure 4: Overview of a PE of the first instances of the CRC model.

3.4. Application Domains

There is a variety of criteria by which application domains can be defined. 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: data path 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 reconfiguration since threads can be scheduled on the basis of clock cycles with very little overhead.

4. Experimental Results

In this section we present area, speed, and power estimations that we obtained from synthesizing components of various instances of the CRC model. All instances are of the type described in the previous section. 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 data path so that the result fits the data path width. This set of operators allows running 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 data path 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.

4.1. Area

Figure 5 depicts the area composition of a PE for a data path width of 8, 16, and 32 bits. The number of contexts as well as the number of states is set to 16 for all three instances. The arithmetic and logic operations excluding multiplication use a small area even for wide data paths. The multiplier allows only input operands of half the word length but still uses a lot more area than the remaining operations. It should therefore be well considered if multipliers are actually needed in a domain specific architecture. The twelve data and status registers also account for a noticeable portion of the area and should be set to a reasonable number to match the applications of a domain. The multiplexers are mostly used to implement the reconfigurable interconnect network. They not only make up a considerable amount of the area by themselves; they also affect the size of the context memory since the select signals for all multiplexers are stored for each context. Especially for a small data path width FSM and context memory together use a large portion of the area. This area as well as the area of the 64-bit shift register together with the control logic to configure the architecture from outside at boot time is independent of the data path width. The area of the FSM and context memory depending on the number of states and contexts is shown in figure 6. The number of states and contexts are set to the same value for all instances. For the instances of the CRC model analyzed in this paper it does not make sense to have more contexts than states or

significantly more states than contexts. Since the context memory uses by far more area than the FSM it is the best candidate if a reduction of area is desired. Overall, the reconfigurable interconnect network, i.e. the multiplexers together with the entries in the context memory, constitute the highest potential to optimize the area.

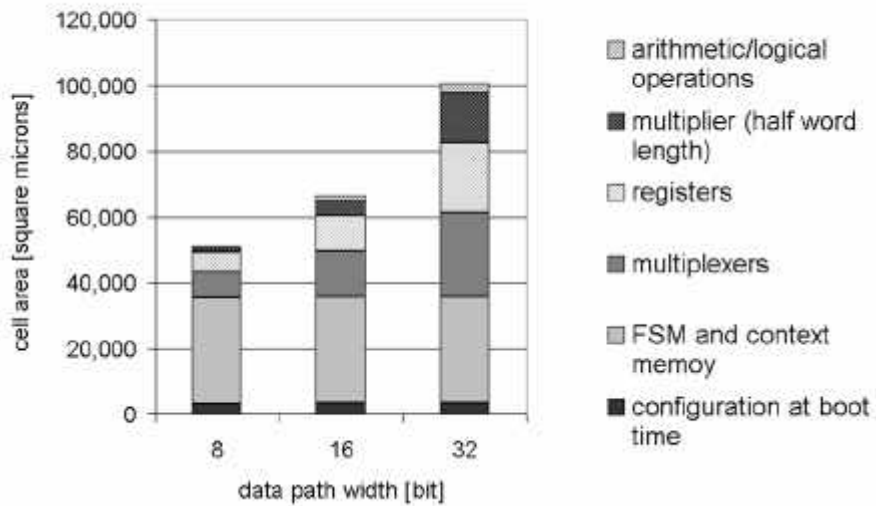


Figure 5: Area composition of a PE for different data path widths.

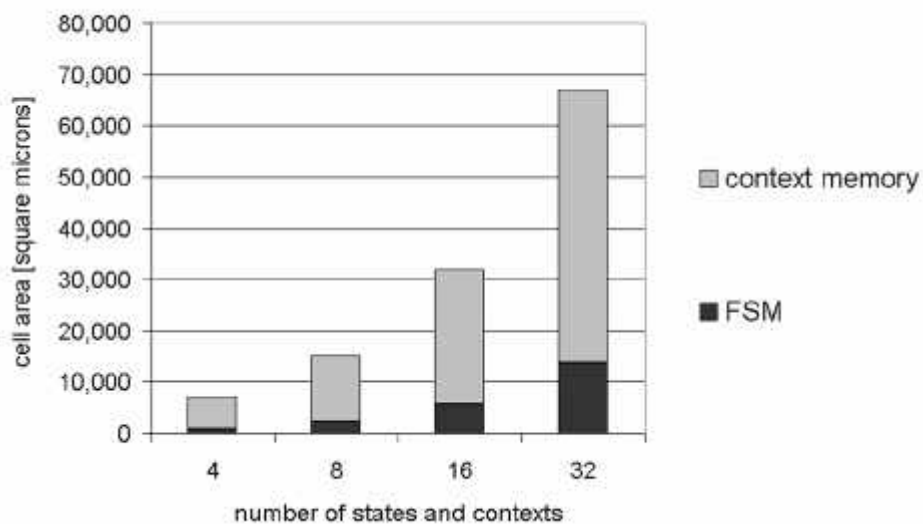


Figure 6: Area breakdown for FSM and context memory.

It should be noted that the FSM, the context memory, and the multiplexers are not entirely overhead imposed by the feature of processor-like reconfiguration. In an ASIC or FPGA implementation area is also needed to implement the FSM and to route signals in the data path depending on the state.

4.2. 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. Figure 7 shows a speed comparison between the CRC FU and a Xilinx Virtex-II FPGA. For the multipliers and the carry-look-ahead adders, the CRC FU is about ten times faster for different word lengths of the functions. For the arithmetic operations the relatively long delay of the functions dominates the delay of the function selection. For the logic AND the speed up is only small because the function selection dominates the delay of the very simple logic function. While the AND function implemented by the FPGA has the same delay for all word lengths the delay of the CRC FU implementation varies for different word lengths. We have analyzed the delay from the data/status inputs of the FU to the data/status output. The delay from the select input of the FU to the output is hidden by the delay of preceding components in the data path and is therefore not considered in the comparison. Since the synthesis tool chooses different components from the standard cell library when implementing the function selection for the different word lengths and we have not enforced to optimize only the data input-to-output delay, the 32-bit AND is actually faster than the 16-bit AND which in turn is faster than the 8-bit AND.

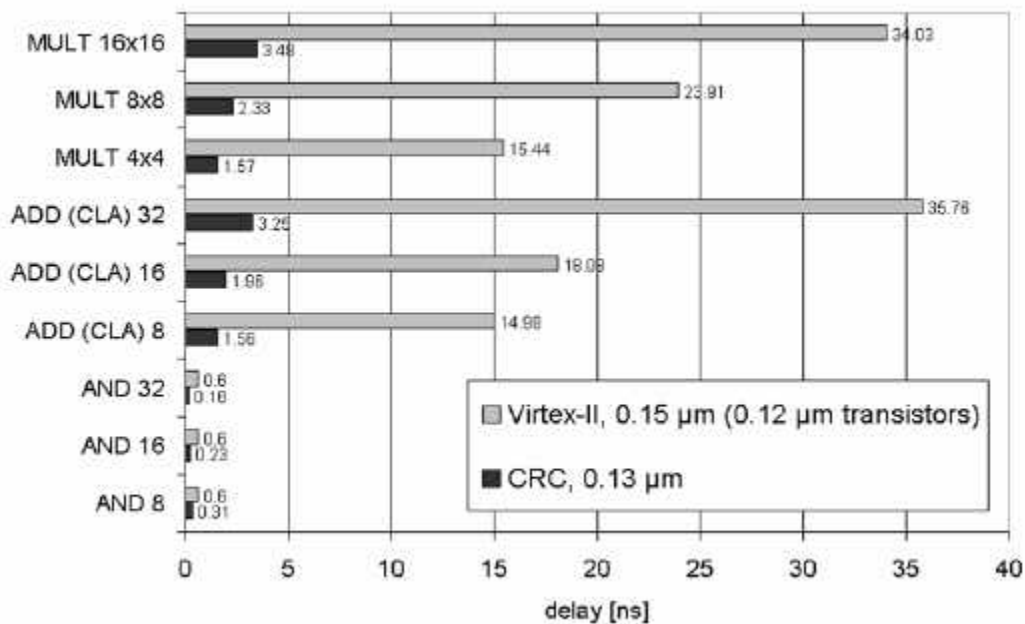


Figure 7: Speed comparison of the CRC FU with an FPGA.

The function selection is not the only part of the CRC model where a delay overhead arises from processor-like reconfiguration. The multiplexers in the reconfigurable data path also cause additional delay compared to an ASIC. Since the instances of the CRC model evaluated in this paper only provide nearest neighbor connections, the number of multiplexers between two FUs depends on the placement and routing of an application. Each multiplexer has a delay of about 0.1 to 0.3 ns which is subject to the data path width and the number of select inputs. The generation of the control signals for the reconfigurable data path also causes a delay overhead. Figure 8 shows the time needed to generate a new context. Starting from the register in the FSM that holds the current state, i.e. from the rising clock edge, about 0.5 to 1 ns are needed to select the context to be used in this state. This delay depends on the number of states and contexts. Independent of this number about 0.6 ns are needed to generate the control signals for the data path that are stored in the context memory. The so far accumulated delay of about 1.1 to 1.6 ns must not be added to the delay of each FU if operator chaining is applied. If, for instance, the first operator in the chain is an 8-bit addition the subsequent PE is already configured with the current context when the result from the addition arrives at this PE. The third component that adds to the time needed for generating a new context is the delay of the logic that determines the next state at the end of a clock cycle, i.e. from the input ports of the FSM to the next rising clock edge. This delay increases with the number of states almost up to 1 ns for 16 states. As in the case of the function selection in the FU the synthesis tool chooses different library components to implement the next-state logic for different numbers of states so that the delay for 32 states is actually shorter than for 16 states. The delay of the next-state logic can be hidden if the control signals for the FSM are available before other operations in the data path are finished.

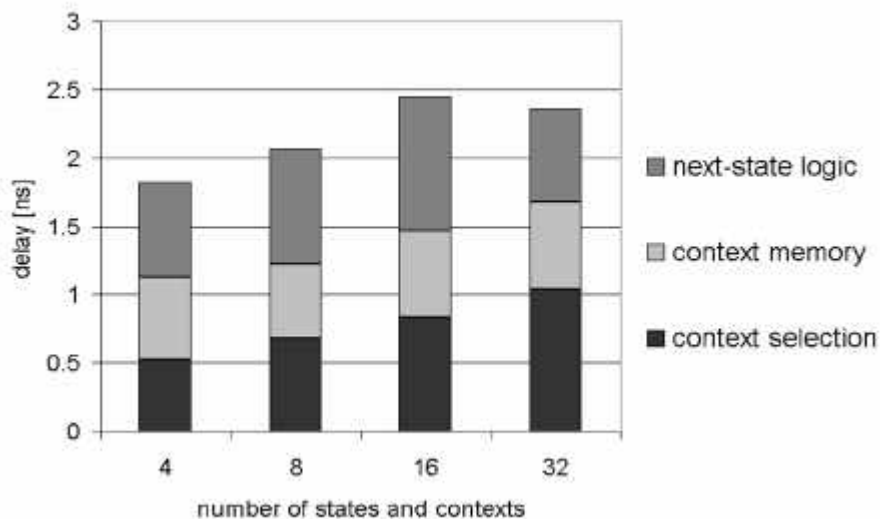


Figure 8: Timing for generating a new context.

Similar to what we have pointed out in our area evaluation, the components involved in processor-like reconfiguration might also be present in a circuit that is implemented by an ASIC and in particular in a statically reconfigurable architecture.

4.3. Power

For the power estimations the data path 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. Figure 8 shows the power dissipation of the different modules of a PE in percent of the total power dissipation. The percent values are based on a clock frequency of 100 MHz and are only slightly different for 50 MHz. The shift register together with the control logic to configure the architecture 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. This shows that the estimation of the total power dissipation is rather pessimistic. For future instances of the CRC model we will eliminate switching activity during normal operation for this part of a PE completely. 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.

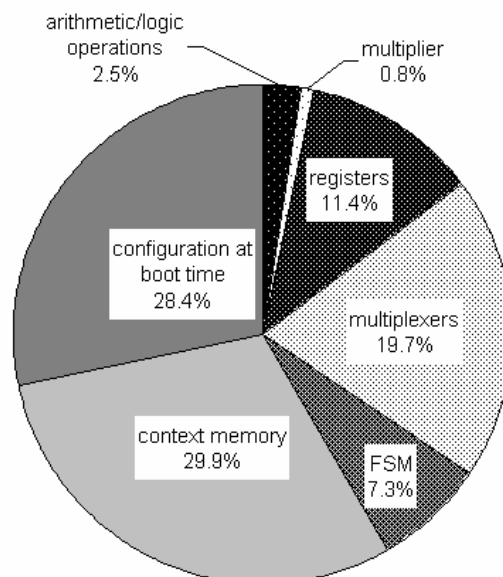


Figure 9: Breakdown of power dissipation of a PE running at 100 MHz.

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.

5. 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 applications 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.

6. Acknowledgements

This work is funded by DFG under RO-1030/13 within the ‘DFG Priority Program 1148’ which is focused on reconfigurable computing systems.

7. References

- [1] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Design, Automation and Test in Europe*, 2001.
- [2] T. Kuhn, T. Oppold, M. Winterholer, W. Rosenstiel, M. Edwards, and Y. Kashi. A framework for object oriented hardware specification, verification, and synthesis. In *Design Automation Conference*, 2001.
- [3] M. Motomura. A dynamically reconfigurable processor architecture. In *Microprocessor Forum*, 2002.
- [4] K. Wakabayashi. C-based synthesis experiences with a behavior synthesizer, “Cyber”. In *Design, Automation and Test in Europe*, 1999.
- [5] 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.

- [6] IPFlex, Inc. <http://www.ipflex.com>.
- [7] picoChip Designs Limited. <http://www.picochip.com>.
- [8] QuickSilver Technology, Inc. <http://www.qstech.com>.
- [9] A. DeHon. Reconfigurable architectures for general-purpose computing. Technical Report AITR-1586, MIT Artificial Intelligence Laboratory, Sept. 2 1996.
- [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] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, June 2002.
- [12] 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.
- [13] A. Abnous and J. Rabaey. Ultra-low-power domain-specific multimedia processors. In *IEEE VLSI Signal Processing Workshop*, 1996.
- [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] P. Gutberlet and W. Rosenstiel. Scheduling between basic blocks in the CADDY synthesis system. In *European Conference on Design Automation*, 1992.