Design-Space Exploration of Low Power Coarse Grained Reconfigurable Datapath Array Architectures

R. Hartenstein, Th. Hoffmann, U. Nageldinger

Computer Structures Group (Rechnerstrukturen), Informatik
University of Kaiserslautern, D-67653 Kaiserslautern, Germany
hartenst@rhrk.uni-kl.de - http://xputers.informatik.uni-kl.de - Fax: +49 631 205 2640

Abstract. Coarse-grain reconfigurable architectures promise to be more adequate for computational tasks due to their better efficiency and higher speed. Since the coarse granularity implies also a reduction of flexibility, a universal architecture seems to be hardly feasible, especially under consideration of low power applications like mobile communication. Based on the KressArray architecture family, a design-space exploration system is being implemented, which supports the designer in finding an appropriate architecture featuring an optimized performance / power trade-off for a given application domain. By comparative analysis of the results of a number of different experimental application-to-array mappings, the explorer system derives architectural suggestions. This paper proposes the application of the exploration approach for low power KressArrays. Hereby, both the interconnect power dissipation and the operator activity is taken into account.

1. Introduction

Many of today’s application areas, e.g. mobile communication, require very high performance as well as a certain flexibility. It has shown, that the classic ways of realizing the associated algorithms, ASIC implementation and microprocessors, are often not adequate, as microprocessors cannot provide the performance, while ASIC implementations lack flexibility.

As a third way of implementation, reconfigurable computing has gained importance in the recent years. It is expected, that in order to obtain sufficient flexibility for high production volume most future SoC implementations need some percentage of reconfigurable circuitry [1]. Reconfigurable computing even has the potential to question the dominance of the microprocessor [2].

Early approaches in reconfigurable computing were based on Field-Programmable Gate Arrays (FPGAs), which provide programmability at bit-level. These solutions soon turned out to have some disadvantages for computing applications, as complex operators have to be composed from bit-level logic blocks. This leads to the following drawbacks, among others:

- A large amount of configuration data is needed, which makes fast configuration hard and increases power dissipation during configuration.
- As the operators are made of several logic blocks, regularity is mostly lost and an extra routing-overhead occurs. This extra routing also increases the power dissipation of the circuit during run-time.

To encounter the disadvantages of FPGA-based solutions, coarse-grain reconfigurable architectures have been developed for computational applications [3], [4], [5], [6], [7], [8], [9], [10]. These devices are capable of implementing high-level operators in their processing elements, featuring multiple-bit wide datapaths. Coarse grain reconfigurable architectures avoid several drawbacks of FPGAs. Featuring relatively few powerful operators instead of many logic blocks, coarse grain architectures need much less configuration data. Also, as the processing elements can be implemented in an optimized way, both the expected performance and the power dissipation are lower than for FPGAs, as shown in [11]. However, for the use of a coarse granularity some problems still have to be solved. to cope with the reduced flexibility compared to FPGA-based solutions:

- Processing elements of coarse-grain architectures are more "expensive" than the logic blocks of an FPGA. While it is possible for FPGA mappings, that a number of logic blocks is unused or cannot be reached by the routing resources, especially the latter situation is quite annoying for coarse-grain architectures due to the fewer processing elements of higher area consumption.
- While the multi-bit datapath applies well to operators from high-level languages, operations working on smaller word-lengths and especially bit manipulation operations are either weakly supported, or need a sophisticated architecture. If such operations occur, like e.g. in data compression algorithms, they may result in a complex implementation requiring several processing elements, or, in difficult architectural requirements.
- Although the power consumption caused by the routing resources can be expected to be lower than for FPGAs [11], the problem of a careful architectural design of the interconnect network, which provides both low power and adequate flexibility remains.

Due to these problems, the selection of architectural properties like datapath width, routing resources and operator repertory is a general problem in the design of coarse-grain architectures. Thus, a design space exploration is done in many cases, to determine suitable architectural properties. As the requirements to the architecture are mostly dependent on the set of typical applications to be mapped onto it, previous efforts use normally a set of example applications, e.g. DES encryption, DCT, or FIR filters. Examples for such approaches are published in [9] and [10]. According to these methods, coarse-grain architectures are often optimized for a specific application area, like DSP or multimedia applications.

While most of these explorations focus on performance or area, power considerations are often thrown over the wall from architectural exploration to physical design. But in regard to interconnect networks, Zhang et al. have presented a comparison and analysis of

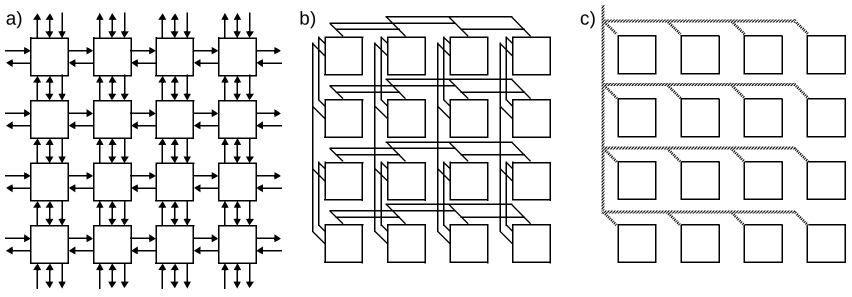

Figure 1: Three levels of interconnect for KressArray architectures: a) nearest neighbor links, b) backbuses in each row or column (for more examples see figure 2), c) serial global bus.

reconfigurable interconnect architectures using energy as a metric in [12], giving general results for DSP applications. However, we feel that optimized architectures can also be found for more specific application domains, which are defined by a number of sample applications.

To find a suitable architecture for a given domain, the KressArray Xplorer framework is currently being implemented. The framework uses the KressArray [7] [8] architecture family as basis for an interactive exploration process. When a suitable architecture has been found, the mapping of the application is provided directly. The designer is supported during the exploration by suggestions of the system how the current architecture may be enhanced. This paper proposes the application of this framework for power-aware architecture exploration, with the discussion of related issues.

The rest of this paper is structured as follows: To give an overview on the topic, the next two sections briefly sketch the KressArray architecture family and the design space for the exploration process published elsewhere [13]. In section 4, our general approach for an interactive design space exploration for an application domain is presented. After this, a short overview on the KressArray Xplorer framework is given. The next section outlines our approach for the generation of design suggestions, which can be used to incorporate the models for power estimation presented in the following section. Finally, the paper is concluded.

2. The KressArray architecture family

The KressArray family is based on the original mesh-connected (no extra routing areas, see figure 2 d, e) KressArray-1 (aka rDPA) architecture published elsewhere [8]. An architecture of the KressArray family is a regular array of coarse grain reconfigurable DataPath Units (rDPUs), each featuring a multiple-bit datapath and providing a set of coarse grain operators. The original KressArray-1 architecture provided a datapath of 32 bits and all integer operators of C, the proposed system can handle also other datapath widths and operator repertories. The different types of communication resources are illustrated in figure 1. There are three levels of interconnect: First, a rDPU can be connected via nearest neighbor links to its four neighbors to the north, east, south and west. There are unidirectional and bidirectional links. The data transfer direction of the bidirectional ones is determined at configuration time. Second, there may be backbuses in each row or column, which connect several rDPUs. These buses may be segmented, forming several independent buses. Third, all rDPUs are connected by one single global bus, which allows only serial data transfers. This type of connection makes only sense for coarse grain architectures with a relatively low number of elements. However, a global bus effectively avoids the situation, that a mapping fails due to lack of routing resources. The rDPUs themselves can serve as pure routing elements, as an operator, or as an operator with additional routing paths going through. Some more communication architecture examples are shown in figure 2.

The number and type of the routing resources as well as the operator repertory are subject of change during the exploration process. Typically, a trade-off has to be found between the estimated silicon area, the performance, and the power dissipation of the architecture, where both performance and power dissipation will typically depend on the application to be implemented.

3. The KressArray Design Space

The KressArray structure defines an architecture class rather than a single architecture. The class members differ mainly by the available communication resources and the operator repertory. Both issues have obviously a considerable impact on the performance of the architecture. In the following, we define the design space for KressArray architectures based on the introduction given in section 2.

The following aspects of a KressArray architecture are subject to the exploration process and can be modified by the tools of the exploration framework:

- The size of the array.
- The operator repertory of the rDPUs.
- The available repertory of nearest neighbor connections. The numbers of horizontal and vertical connections can be specified individually for each side and in any combination of unidirectional or bidirectional links.
- The torus structure of the array. This can be specified separately for each nearest neighbor connection. The possible options are no torus structure or torus connection to the same, next or previous row or column respectively.

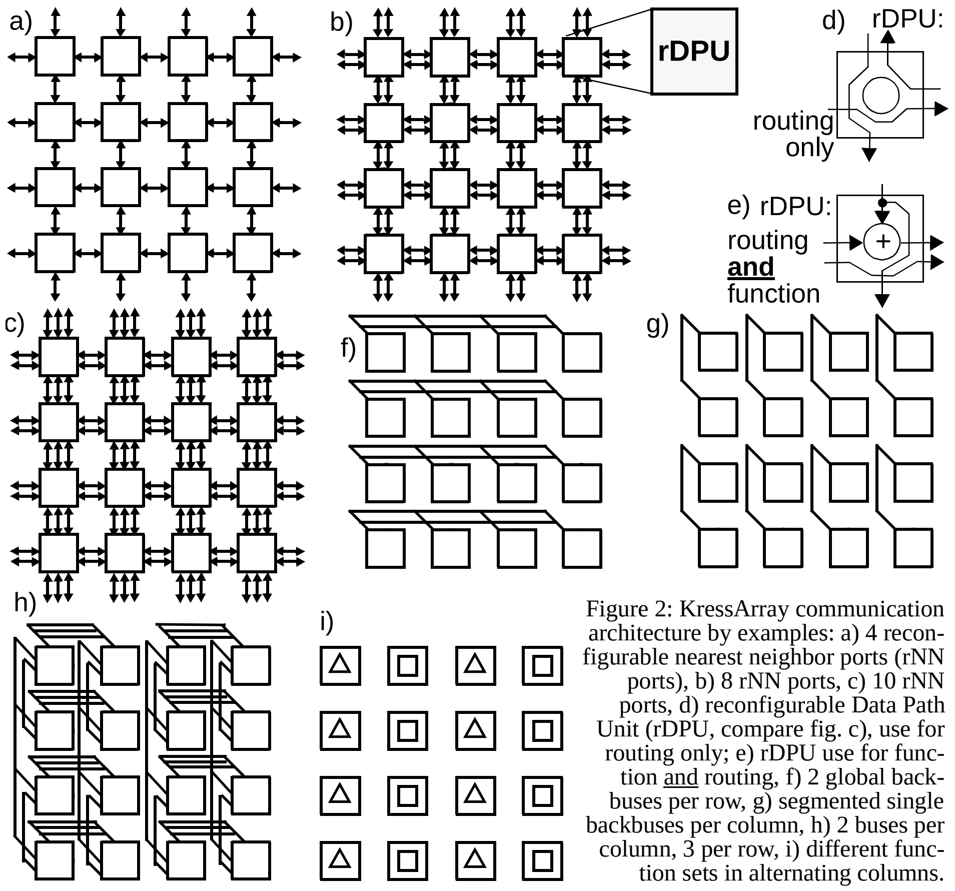

Figure 2: KressArray communication architecture by examples: a) 4 reconfigurable nearest neighbor ports (rNN ports), b) 8 rNN ports, c) 10 rNN ports, d) reconfigurable Data Path Unit (rDPU, compare fig. c), use for routing only; e) rDPU use for function <u>and</u> routing, f) 2 global backbuses per row, g) segmented single backbuses per column, h) 2 buses per column, 3 per row, i) different function sets in alternating columns.

- The available repertory of row and column buses. Here, the number of buses is specified as well as properties for each single bus: The number of segments, the maximal number of writers, and the length of the first segment, which allows buses having the same length but spanning different parts of the array.
- Areas with different rDPU functionality. For example, a complex operator may be available only in specific parts of the array. This allows also the inclusion of special devices in the architecture, like embedded memories. The operator repertory can be set for arbitrary areas of the array, using generic patterns described by few parameters.
- The maximum length of routing paths for nearest neighbor connections, which can be used to satisfy hard timing or power constraints.
- The number of routing paths through a rDPU. A routing path is a connection from an input to an output through a rDPU, which is used to pass data to another rDPU.
- The interfacing architecture for the array. Basically, data words to and from the KressArray can be transferred by either of three ways: Over the serial global bus, over the edges of the array, or over an rDPU inside the array, where the latter possibility is mostly used for library elements.

In order to find a suitable set of these properties for a given application domain, an interactive framework is currently developed [13]. The framework, called KressArray Xplorer, allows the user a guided design of a KressArray optimized for a specified problem. At the end of the design process, a description of the resulting architecture is generated.

4. General Approach to Design Space Exploration

The design flow of design space exploration for a domain of several applications is illustrated by figure 3. In most cases a cycle through the loop takes only a few minutes, so that a number of alternative architectural designs may be created in a reasonable time. First, all applications are compiled into a representation in an intermediate format, which contains the expression trees of the applications. All intermediate files are analyzed to determine basic architecture requirements like the number of operators needed. The main design space exploration cycle is interactive and meant to be performed on a single application. This application is selected from the set of applications in a way, that the optimized architecture for the selected application will also satisfy the other applications. This selection process is done by the user, with a suggestion from the system. For low power exploration, two application properties can be considered: Regularity and the estimated power consumption without regarding the component of the routing architecture. An approach to measure the regularity of an application has been published in [14]. The power estimation can be derived from scheduled data flow graphs using a methodology described in [15] and [16].

The exploration itself is an interactive process, which is supported by suggestions to the designer, how the current architecture could be modified. The application is first mapped onto the current architecture. The generated mapping is then analyzed and statistic data is generated. Based on this data, suggestions for architecture improvement are created using a fuzzy-logic based approach described below. The designer may then chose to apply the suggestion, propose a different modification, return to a previous design version, or end the exploration. Some modifications allow the new architecture to be used directly for the next mapping step, while others will require a re-evaluation of the basic architecture requirements and/or a re-selection of the application for the exploration. Especially, a change of the operator repertory for the rDPUs requires the replacement of subtrees in the dataflow graph, thus effecting the number of required rDPUs in the array, complexity of all applications, and the power consumption. Thus, for a change of the operator repertory, a re-evaluation is required.

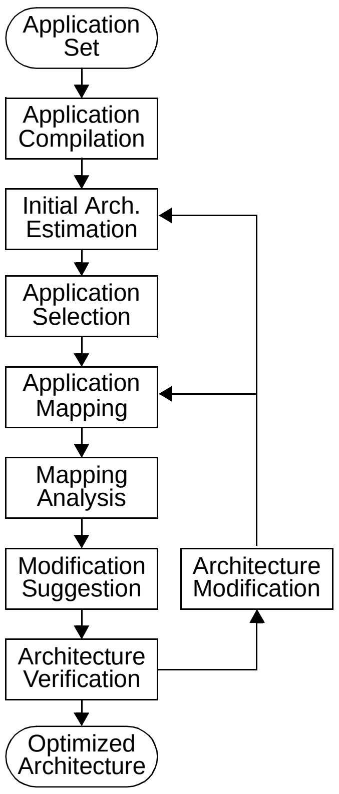

Figure 3: Global approach for domain-specific architecture optimization.

After the exploration cycle has ended, the final architecture has to be verified by mapping the remaining applications onto it. On the one hand, this step produces mappings of all applications to be used for implementation, while on the other hand, it is checked if the architecture will satisfy the requirements of the whole application domain.

5.The KressArray Xplorer

This section will give a brief description of the components of the KressArray Xplorer, which has been published elsewhere [13]. An overview on the Xplorer is shown in figure 4. The framework is based on a design system, which can handle multiple KressArray architectures within a short time.

It consists of a compiler for the high-level language ALE-X, a scheduler for performance estimation, and a simulated-annealing based mapper. This system works on an intermediate file format, which contains the net list of the application, delay parameters for performance estimation, the architecture description, and the mapping information. The latter is added by the mapper at the end of the synthesis process. An architecture estimator determines the minimum architecture requirements in terms of operator numbers and suggests the application with the expected worst requirements to power and routing resources to the user through an interactive graphical user interface. This interface is also generally used to control all other tools. Further, it contains two interactive editors, an architecture editor, which allows to change the architecture independently by the design suggestions, and a mapping editor, which allows to fine-tune the result of the mapper. An analyzer generates suggestions for architecture improvements by information gathered directly from the mapping and other sources. An instruction mapper allows the change of the operator repertory by exchanging complex operators in the expression tree with multi-operator implementations. A simulator allows both simulation of the application on the architecture as well as generation of a behavioral HDL (currently Verilog) description of the KressArray. Finally, a Module Generator (planned) should generate the final layout of a KressArray cell. Throughput estimations are generated from Scheduler results and statistical data. From Module Generator parameters also an area estimation can be generated easily.

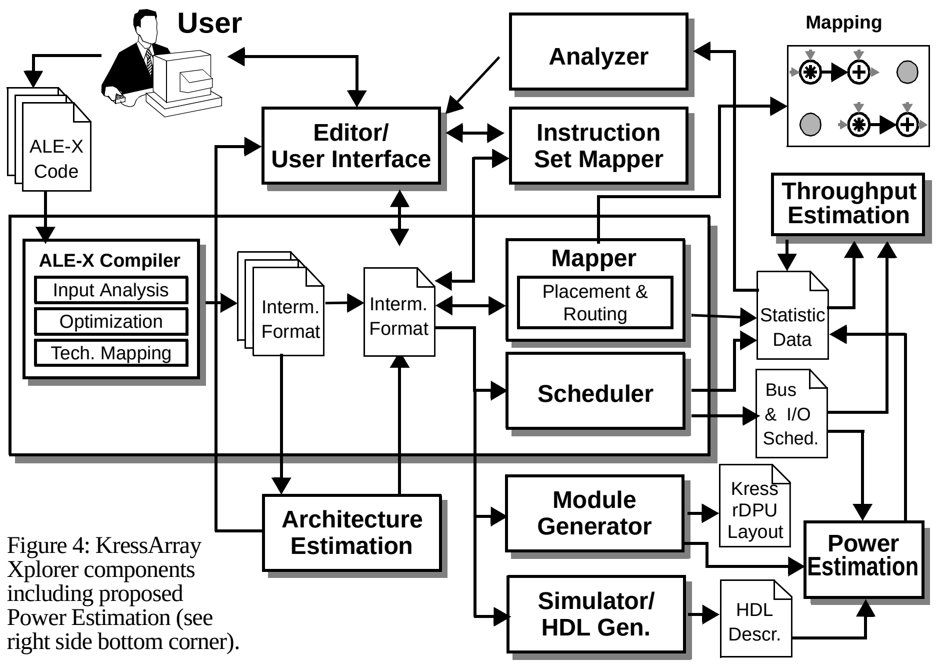

Figure 4: KressArray Xplorer components including proposed Power Estimation (see right side bottom corner).

6. Generation of Design Suggestions

In this chapter, we will give a short overview on our approach to analysis and generation of design suggestions, which is performed by the analyzer tool of the Xplorer framework. The problem of the generation of feedback on a given design can be split into several subproblems:

- Analysis of the current architecture and gathering of information. This includes the combination of data to gain derived information.
- Generation of suggestions from this information.
- Ranking of the suggestions after their importance.

In our approach, the basis for the information gathering step is the mapping produced by the design system. Primary data gathered includes the usage of buses and nearest-neighbor connections in percent, the number of serial bus connections, the estimated number of execution cycles, and others. This primary data can be used to derive secondary information, like the estimated power dissipation, based on the mapping and the actual usage of routing resources by the application.

To allow the gathering of a variety of information from the mapping, the analyzer tool is implemented using a plug-in-based approach, which provides also the flexibility to extend the system. The plug-ins are controlled by the analyzer tool, which holds also data structures for the analysis results.

The design suggestions themselves are then generated using an approximate reasoning approach based on fuzzy logic [17], [18]. The knowledge how to guide the exploration process is expressed in implication rules, like known from expert systems. The fuzzy approach has been chosen to allow the generation of suggestions based on inexact information, as during the exploration process, it is assumed the designer will apply a 'fast' mapping (by tuning the parameters of the simulated annealing), which will probably result in a mapping with a lower quality.

7. Power Estimation for the Exploration Process

In this section, we propose an approach to be used to generate a power estimation. Given an application and a KressArray architecture, onto which this application has been mapped, the total power consumption is composed from the power consumed by the operators and the power consumed by the interconnect network. We will now discuss how to estimate these in a way adequate for the exploration process. Note, that for the exploration process, a very accurate estimation of the power consumption is not necessary. Instead, we will propose simplified measures, which allow a relative quantitative comparison of power dissipation of different architectures.

The operator component is determined by the operator repertory (taken from the intermediate form seen in figure 4), by the implementation of the operators (to be derived from Module Generator parameters), by the configuration (indicating which operator has been selected) and by the switching activity (to be extracted from the intermediate form (or from the HDL description) and the Scheduler results: see figure 4).

In our Xplorer framework, we assume the operator repertory to be organized in several sets, which can be switched by the instruction set mapper mentioned in section 5, and that according power models for the final implementation of those sets are available from the library of the Module Generator (figure 4). With these preliminaries, the power consumption for a given application can be estimated by techniques like those presented in [15], [16], since the required data flow graph can be extracted from the intermediate format (s. figure 4). The resulting estimation does not consider the routing architecture or an actual mapping of the application at all (however, see next paragraph). Though it can be used to distinguish applications in order to select the one for the exploration process described in section 4. This selection takes place at the beginning and when the operator set has been changed.

The power consumption caused by the routing network depends also on the actual use of the routing resources during run-time. However, for our purposes, we can employ a more relaxed metric for the different routing resources of KressArrays. Instead of the power, we use the energy of the routing resources for measure, neglecting the actual usage during execution. Generally, for the energy consumption *E* of an interconnect structure, the following estimation can be used (a modified version of the model in [12]) for a net *con* starting from an operator *op* in the mapping:

$$E(con) = (C_{wire} + C_{switch} + Fanout_{op} \bullet C_{load}) \bullet V^2$$

where C_{wire} and C_{switch} are the according capacitances of the wire segments and switches for each routing resource, $Fanout_{op}$ is the fan-out of the operator resembling the source of the net, C_{load} is the load capacitance of an operator input, and *V* is the supply voltage. For our purposes, we can simplify this equation to:

$$E(con) = K_L \bullet L + K_S \bullet NR \bullet S + Fanout_{op} \bullet K_{load} \bullet NR$$

Where K_L is the energy per wire segment, L is the number of segments used, K_S is the energy per switch, NR is the number of all routing resources meeting at an rDPU which determines the width of the required switch, S is the number of switches, and K_{load} is the energy per fan-out connection. The values K_L, K_S, and K_{load} can be assumed to be constant and known for each interconnect type. Then, we get the following estimates for the energy for each routing resource:

Global Bus. For this type of connection, the capacity of the whole bus structure has to be considered, as this bus is not switched, but operates in a serial manner. The capacity is dependent of the array size and the actual layout of the bus structure. For a relative measure between different architectures, we can assume a general layout like in figure 1c. If the Sizes of the array in x and y-direction are denoted as AS_x and AS_y respectively, the number of bus segments is approximated by:

$L = AS_x \bullet (AS_y + 1)$, which we simplify to $L = AS_x \bullet AS_y$ to keep the measure independent from the aspect ratio of the array. Thus, we get:

$E_{global}(con) = K_L \bullet AS_x \bullet AS_y + Fanout_{op} \bullet K_{load} \bullet NR$

for a global bus connection *con* with source operator *op*.

Row/Column backbuses. For those connections, both source and one or more sinks lie on one bus segment, which in itself is not switched. Thus, we get:

$E_{backbus}(con) = K_L \bullet Segment(op) + Fanout_{op} \bullet K_{load} \bullet NR$

with *Segment(op)* denoting the length of the bus segment holding the source operation *op*.

Nearest Neighbor Connects. Using the nearest neighbor connects, a connection from one source operator to several sinks can be implemented, resulting in a path composed of several subpaths (cf. figure 5). There is a subpath for each sink composed of a sequence of length-1 connections, whereby different subpaths do not share such segments and each rDPU lying on this sequence inflicts additional switching energy. We get the following estimation for a path made up of $Fanout_{op}$ subpaths, each of which with the according subpath length SPL:

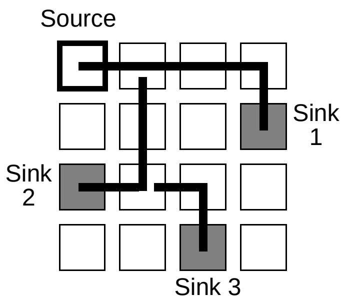

Figure 5: Path composed of three subpaths

$$E_{NN}(con) = \sum_{i=1}^{Fanout_{op}} (SPL_i \cdot K_L + (SPL_i - 1) \cdot K_L \cdot NR + K_{load} \cdot NR)$$

According to the software architecture of the framework outlined in section 5, the implementation of these estimation functions can easily done as plug-ins, which use available data from the intermediate file, given the hardware-parameters are provided.

8. Conclusions

An interactive approach for the design-space exploration of mesh-based reconfigurable architectures from the KressArray family has been presented. An according framework called KressArray Xplorer is based on a design system which allows the specification of the input language in a high-level language. During the exploration process, which is based on iterative refinement of the current architecture, the designer is supported by suggestions on how the current solution may be improved. To apply the framework for exploration of low power architectures, according models have been proposed, which can easily be integrated into the framework.

Literature

1. J. Rabaey. “Low-Power Silicon Architectures for Wireless Communications”; Embedded Tutorial, ASP-DAC 2000, Yokohama, Japan, Jan. 2000.
2. R. Hartenstein (invited paper): The Microprocessor is no more General Purpose: why Future Reconfigurable Platforms will win; Int’l Conf. on Innovative Systems in Silicon, ISIS'97, Austin, Texas, USA, Oct 1997
3. E. Mirsky, A. DeHon: „MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources“, Proc. FPGAs for Custom Computing Machines, pp. 157-166, IEEE CS Press, Los Alamitos, CA, U.S.A., 1996.
4. A. Marshall et al.: A Reconfigurable Arithmetic Array for Multimedia Applications; FPGA’99, Int’l Symposium on Field Programmable Gate Arrays, Monterey, CA, U.S.A., Febr. 21 - 23, 1999
5. C. Ebeling, D. Cronquist, P. Franklin: „RaPiD: Reconfigurable Pipelined Datapath“, Int’l Workshop on Field Programmable Logic and Applications, FPL'96, Darmstadt, Germany, Sept 1996.
6. R. A. Bittner, P. M. Athanas and M. D. Musgrove: „Colt: An Experiment in Wormhole Runtime Reconfiguration“; SPIE Photonics East `96, Boston, MA, USA, November 1996.
7. R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; Asia and South Pacific Design Automation Conf. (ASP-DAC'95), Chiba, Japan, Aug. 29 - Sept. 1, 1995
8. R. Kress: „A Fast Reconfigurable ALUs for Xputers“, Ph.D. thesis, Univ. Kaiserslautern, 1996.
9. E. Waingold et al.: „Baring it all to Software: Raw Machines“, IEEE Computer 30, pp. 86-93.
10. S. C. Goldstein, H. Schmit, et al.: „PipeRench: A Coprocessor for Streaming Multimedia Acceleration“; Int‘l Symposium on Computer Architecture 1999, Atlanta, GA, USA, May 1999.
11. A. Abnous, K. Seno, Y. Ichikawa, M. Wan and J. Rabaey: Evaluation of a Low-Power Reconfigurable DSP-Architecture; Proceedings of the Reconfigurable Architectures Workshop, Orlando, Florida, USA, March 1998.
12. H. Zhang, M. Wan, V. George and J. Rabaey: Interconnect Architecture Exploration for Low-Energy Reconfigurable Single-Chip DSPs; Proceedings of the WVLSI, Orlando, Florida, USA, April 1999.
13. R. Hartenstein, M. Herz, Th. Hoffmann, U. Nageldinger: KressArray Xplorer: A New CAD Environment to Optimize Reconfigurable Datapath Array Architectures; 5th Asia and South Pacific Design Automation Conference 2000, ASP-DAC 2000, Yokohama, Japan, January 25-28, 2000
14. L. Guerra, M. Potkonjak and J. Rabaey: „System-Level Design Guidance Using Algorithm Properties“; IEEE VLSI Signal Processing Workshop, 1994.
15. L. Kruse, E. Schmidt, G. Jochens, A. Stammermann and W. Nebel: Lower Bounds on the Power Consumption in Scheduled Data Flow Graphs with Resource Constraints; Proceedings of the European Design and Test Conference DATE 2000.
16. L. Kruse, E. Schmidt, G. Jochens and W. Nebel: Lower and Upper Bounds on the Switching Activity in Scheduled Data Flow Graphs; Proceedings of ISPLED 1999.
17. W. Pedrycz: „Fuzzy Modelling - Paradigms and Practice“; Kluwer Academic Publishers, 1996.
18. B.R. Gaines: „Foundations of Fuzzy Reasoning“; Int’l Journal of Man-Machine Studies, Vol. 8, 1976.