On Reconfigurable Co-Processing Units

Reiner W. Hartenstein, Michael Herz, Thomas Hoffmann, Ulrich Nageldinger

University of Kaiserslautern,
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de
URL: http://xputers.informatik.uni-kl.de

Abstract. In the last years reconfigurable computing grew from a niche application to an important R&D scene. But also today most architectures lack essential features for the convenient use as a co-processing unit. E.g. embedded accelerator design with traditional FPGAs is very similar to sophisticated ASIC-design due to the bit-level granularity of FPGAs. In this paper important topics for reconfigurable platforms in multitasking systems are discussed. Run-time programmability as well as rapid application implementation using high-level languages are illustrated. Besides the underlying concepts the hardware implementation of a field-programmable ALU array (FPAA), the KrAA-III, is explained.

1 Introduction

Most of today's computers are based on the von Neumann paradigm. This has been shown to be universal but lacks computation power in many applications. In modern desktop-computers, there are many application-specific accelerators for graphic, multimedia, image (de-) compression etc. In fact, there is more silicon area used for the accelerators than for the microprocessor itself. But such types of hardwired add-on accelerator circuits require immense design manpower, due to the complexity of the design space. Thus, a shift from designing customized systems on chips (or boards) to more standardized platforms is desired, to "programming" dynamically reconfigurable circuits, representing a shift from hardware to "structural software".

The approach to use reconfigurable devices to build add-on hardware is topic of current research. But with SRAM-based FPGAs available today, which were originally designed to implement glue logic, such a paradigm shift is not likely to happen. Their single-bit fine granularity parallelism is too inefficient in time and area. Also, the mapping of an application is similar complex as ASIC-design. What is needed instead are reconfigurable platforms supporting coarse granularity parallelism. Because of these requirements such a computing element goes more in the direction of processing elements (PEs) like used in systolic array implementation, in contrast to FPGAs, which are similar to an ASIC approach. Therefore we obtain the new class of Field Programmable ALU Arrays (FPAAs) which can be seen between FPGAs and PEs [HB96]. Examples of FPAAs are the Programmable Arithmetic Devices For DIgital Signal Processing (PADDI, [YR93]) or the Kress ALU Array (KrAA, [HB97]). FPAAs are much more suitable to build such platforms. It will also be shown, that FPAAs enable efficient compilation techniques.

A closer look onto F-CCMs shows that former field programmable devices were originally designed to implement control and glue logic, because they use only narrow datapaths (mostly one bit). Wider register or memory resources on chip are rarely implemented by these devices. The logic is mostly dedicated to implement boolean formulas and not arithmetic functions. In contrast to these fine-grained FPGAs we face different requirements for implementing a computing element. Although some FPGA manufacturers developed more feasible devices (e.g. XC6k, FLEX10k, AT40K) they meet such requirements only insufficiently. A moderately wide datapath is necessary. The

functional units have to implement also more comprehensive arithmetic functions.

In modern computers based on von Neumann CPUs, multitasking operating systems provide a comfortable and efficient way to execute applications: The computation power of the CPU is made available as a resource, which is distributed among the tasks. To make such multitasking systems possible on reconfigurable structures, the following conditions must be met: Very short configuration times are required. Reconfigurations should also be possible during runtime. Further context switches are necessary to change efficiently between several tasks during runtime.

This paper presents a novel architecture of FPAAs called Kress ALU Array III (KrAA-III). It is shown, how the KrAA-III can be used to build a universal platform for embedded accelerators. This architecture provides acceleration speed-up due to parallel processing as well as the capability to be programmed by high-level languages due to the coarse grained structure of the KrAA-III. The paper is structured as follows: In the next section, the use of reconfigurable devices as co-processing unit is discussed. Further the benefits of coarse mesh based structures are illustrated. Section 3 presents such an architectural approach the Kress ALU Array and in section 4 its compilation framework is briefly introduced. In section 5 a mapping example underlines the considerations.

2 Reconfigurable Devices as Co-Processing Units

Not every reconfigurable device is well suited to implement a co-processing unit. Several demands are made on reconfigurable devices. The paper presents a fundamentally new approach to the design of general purpose dynamically reconfigurable accelerator hardware platforms, which are much more area-efficient than the current fine granularity approach. The paper briefly illustrates the feasibility of substantial speed-ups for a variety of application areas. The vision behind this approach is creating and upgrading accelerators by downloading reconfiguration code generated by compilers accepting high level programming language sources, onto a general purpose reconfigurable accelerator.

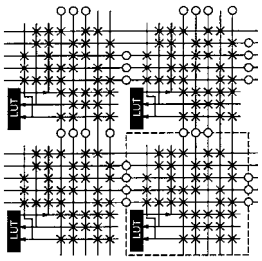

Fig. 1. Interconnect caricature [DH96]

Very harmful is placement and routing as the dominant design style for reconfigurable architectures like FPGAs, where a PE, also called CLB (configurable logic block), is only a single bit wide and includes just a few gates. For function selection it needs 4 or more flipflops with at least 4 gates per flipflop of configuration RAM. Using the FPGA as a computing element several CLBs are united to form a several bit-wide datapath. Having the configuration bits in mind this is a disaster, because the configuration bits are the same for each datapath bit of an operator. The fine granularity FPGAs commercially available today use only about 1% chip area for active logic circuits and about 90% for wiring ([DH96], figure 1), from which a major part is used for reconfigurable routing areas. With severe difficulty commercial FPGAs route in most designs less than 80-90% of the available LUTs. It has shown that on applications with large data elements, fine-grained devices pay much more area for interconnect than coarser-grained devices. For area efficiency an optimized platform should be suitable for full custom design, like known from ASAPs. But ASAPs are not structurally programmable and support only problems with completely regular data dependencies. Therefore FPAAs are as dense as ASAPs but each PE is programmed individually.

2.1 Mesh Based Mapping.

This section introduces a new form of such a high density methodology for dynamically reconfigurable circuits. It is a new form of mesh-connected PE array (also called NEWS

network, figure 4) with direct interconnect of a PE only to its north, east, west and south neighbour, having the following advantages:

- coarse granularity permits full custom style PEs instead of single bit CLBs
- a mesh-connected PE array, similar to PE arrays known from 2-dimensional systolic arrays, allows wiring by abutment instead of using routing channels. Large amounts of chip area required for routing can be saved.
- because of the absence of routing channels the structure synthesis is carried out by a placement-only algorithm.

This method is called “mesh-based mapping”. The application problem is mapped onto a mesh-based reconfigurable architecture, having been designed in full custom design style also using wiring by abutment. Mesh-based reconfigurable is not new: Already in the early 80ies it has been commercialized by Algotronix in Edinburgh, but only at single bit granularity. New, is coarse grain mesh-based reconfigurable, which provides the following main advantages over the current main stream solution by placement and routing.

- very high area-efficiency through custom layout style
- much better optimization results through a restricted design space

The bad optimization results of placement and routing are also caused by the enormous size of the design solution space, which is a severe complexity problem. Therefore the meshed array reduces the design space by the predefined mesh scheme. A new method for application mapping has been developed. The synthesis problem has been mainly reduced to a placement problem only, with only a small residual routing problem, which also is solved by placement of routing elements into a particular PE (selecting a PE to aid routing, e. g. see figure 9). The placement also benefits from the coarse structure of the PE. While bit-level placement struggles out to find implementations of arithmetic functions, placement for coarse grained architectures maps complete arithmetic operations (e.g. addition, subtraction) onto PE.

To act as a co-processing unit a processor interface is of fundamental importance for high speed computation data and configuration code transfer. The processor interface must implement a standard protocol to be connectable to a commercial microprocessor. This includes also word level datapaths (e.g. 16, 32 bit). Also commercially available fine-grained FPGAs, which aim at the co-processor market, provide such an interface (e.g. Xilinx XC6200, Atmel AT40K).

In addition the reconfigurable platform has to be programmable during runtime. Otherwise configuration time has to be counted in addition to the runtime. This would drastically decrease the number of applications to be accelerated. But only runtime programmability is not sufficient if only one configuration layer is available. In that case only unused design space could be written during runtime. At least two configuration layers are necessary to make runtime configuration useful. One (active) configuration layer holds the “program” executed at the moment, while a second configuration layer is written during runtime. After finishing the actual computation the active layer is changed by a context switch mechanism.

An architecture example of the mesh-based approach is the Kress ALU Array-III (KrAA-III) concept introduced in the following. This approach includes the DPSS (data path synthesis system) compiler, containing a simulated annealing optimizer for mapping a function to the KrAA, as well as a data (I/O) scheduler to organize and optimize data streams for communication with the host.

3 The Architecture of the Kress ALU Array III

In this section the KrAA-III is introduced. The KrAA-III consists of PEs called rDPU-III (reconfigurable DataPath Unit III) arranged in a NEWS network. The datapath width of the entire architecture is 32 bit. The KrAA-III is built of several identical KrAA-III devices, which are transparently scalable. Figure 2 shows the KrAA-III chip containing 9 rDPUs. All local interconnects are provided at the chip boundary to connect several devices for larger meshes. To reduce chip pins the local datapaths at the boundaries, which are also 32 bit, are connected in serial mode.

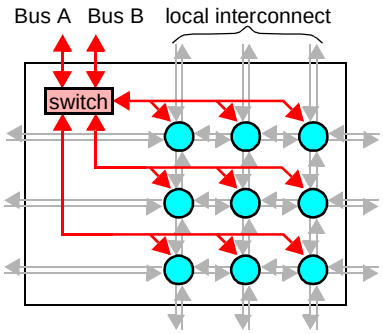

Fig. 2. KrAA-III chip structure with hierarchic global routing scheme

Basically data can be fed via all local interconnects at the border of the mesh. Normally algorithms are mapped into the array requiring the input- and/or output- data not at the border of the mesh. Therefore a hierarchical global routing network is provided, which connects a standard microprocessor with any rDPU of the array. Within an application specific solution data may be fed into the array via the local connections at the border of the array but for the use as co-processing unit data is only routed via the global databus. Local connections are used to pass intermediate results between rDPUs. To avoid that the global bus is getting a bottleneck a novel hierarchical global routing network is introduced, which allows also multiple data transfers in parallel.

Figure 2 shows the structure of the global bus network. The inner global busses connected to the rDPUs are developed once to save valuable design space. As illustrated, they are further connected to a switch, which can either isolate inner busses or connect them to any other bus. If inner busses are isolated, rDPUs connected to this bus can communicate independently from the remaining array. This is also the case if two inner busses are connected to each other. Furthermore there are two parallel busses to the outside of the KrAA-III chip, which can be connected to each inner bus. This enables parallel data transfers from outside the array, where more hierarchy levels and more parallel busses are possible. As a result of the global routing network structure pictured in figure 2 a maximum of three independent global data transfers are possible inside one KrAA-III chip. Furthermore two parallel data transfers on the global bus from outside can be performed concurrently.

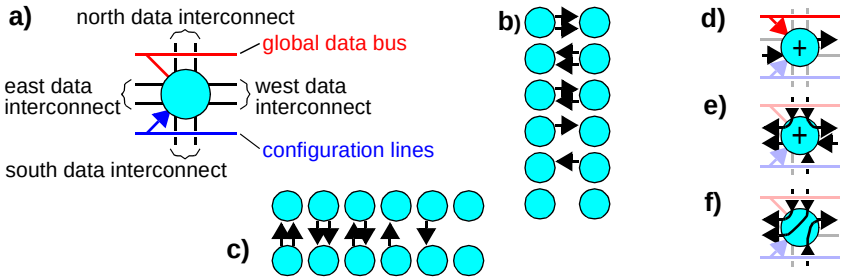

Fig. 3. KrAA-III local routability: a) rDPU-III with all connections b) all possible west-east interconnect configurations, c) all possible nortt-south interconnect configurations, d) rDPU-III serving as arithmetic operator e) rDPU-III serving as arithmetic and routing operator, and f) rDPU-III serving exclusively as routing operator.

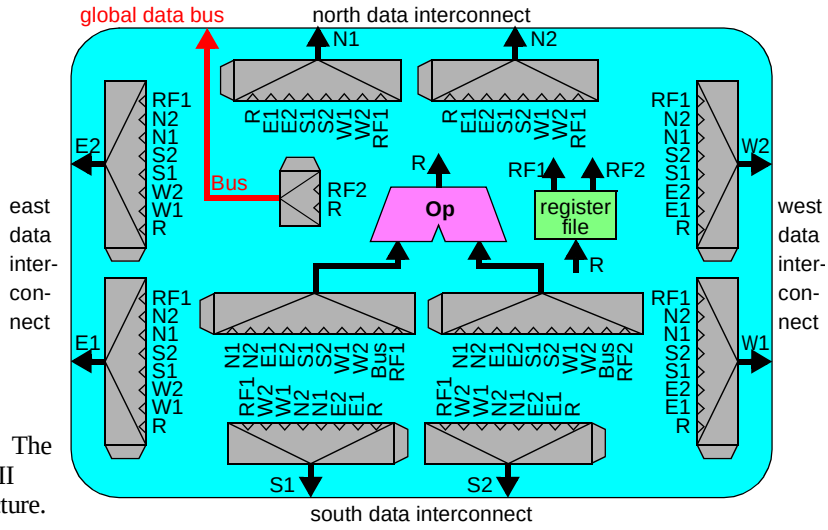

Fig. 4. The rDPU-III architecture.

In addition to the hierarchical global routing network and the configuration lines the KrAA-III has local routing capabilities between nearest neighbors. These “nearest neighbor connections” have the advantage, that they don’t require any chip area. The rDPU-III is designed for wiring the local interconnections by abutment. The KrAA-III has two 32-bit full-duplex connections for each direction of the NEWS network (figure 3a). This connections are reconfigurable, i.e. the direction of the connection has to be programmed (figure 3c,d). To extend local routing capabilities even more the rDPU can serve as routing element (figure 3f) also during performing an arithmetic operation (figure 3e). Local routing saves a lot of global bus cycles and is independent from global bus routing. figure 3b,c shows all local interconnect configurations. As stated before, the local connections over chip-boundaries are serialized in order to reduce the number of chip-pins.

Internally (figure 4) the rDPU-III provides a lot of routing capabilities. Each output can be connected to all other inputs, the internal arithmetic unit and the internal register file. Routing operations are performed independent to internal computations by the multiplexers. The internal register file can be configured with constants as well as it stores interim results. Further the register file is also capable to store input data needed several times to perform a cache like optimization. This technique is called smart interface optimization and described in [HB97]. The arithmetic unit of the rDPU-III provides a high functionality, covering all arithmetic operations of the language C. The operation configured into the rDPU-III is data triggered and performed completely independent from the rest of the array. Using the register file to store input and result of the operation-unit, applications may be mapped to the KrAA-III to form a deep computation pipeline. Furthermore computations are performed asynchronously and different operators can be of different execution time.

The performed routing of the multiplexers and the arithmetic function of a rDPU-III is stored in the configuration memory. It consists of four independent layers. Each layer holds a complete configuration data set for the rDPU. Reconfigurations can be performed very fast by a context switch mechanism. That means that all rDPUs of an array change the actual configuration memory layer. Therefore also the register file for data in the rDPU has to be implemented in four layers. Otherwise all data had to be stored before a context switch is performed. To gain a further speed up out of the independent configuration

layers the configuration control and the channels for configuration data are independent from each other (see also figure 4). Thus reconfigurations of the three idle layers can be performed in parallel to the calculations on the active layer. Therefore the configuration time is no longer a penalty for the accelerator. If configurations and calculations have to be performed sequentially the configuration time has to be added to the runtime and therefore many applications would not benefit from such an accelerator. In the case of the KrAA-III a related programming framework (see section 4) has to regard only reconfiguration times that are longer than the execution times of predecessor tasks. For background configuration already 2 layers would be sufficient. Having more configuration layers tasks executed several times (e.g. nested loops or recurrent functions) can stay in configuration layers permanently and also multitasking processing becomes possible.

The configuration memories are written by the microprocessor via the configuration bus. This bus does not allow read-back operations. For the rDPU the configuration memories are read-only as known from other field-programmable devices. Configuration data is written in two steps. First a 32-bit address word specifies exactly the rDPU in the mesh (because all rDPUs are connected to the same configuration bus), the configuration layer and the memory address in the specified RAM. After the address a 32-bit configuration word is written. This configuration scheme enables high-speed direct access to the configuration memory. No bit-stream has to be passed through the complete array as known from many FPGAs.

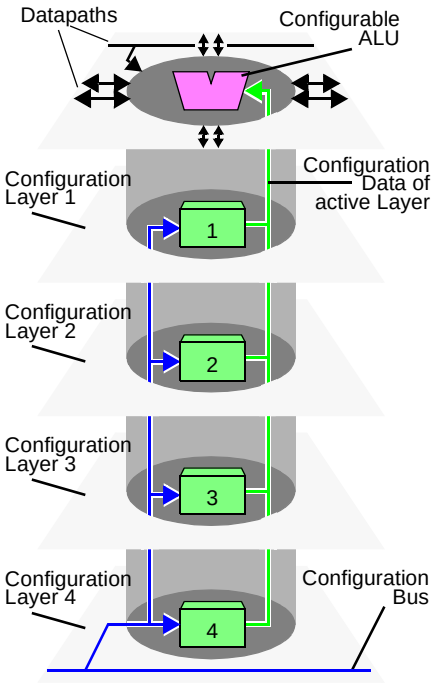

Fig. 5. rDPU with 4 configuration layers.

4 The Datapath Synthesis System

In this section, an application compiler for the KrAA-III, the datapath synthesis system (DPSS) is described briefly. The DPSS allows to map statements from a high level language description onto the KrAA-III. The input language used is called ALEX (Arithmetic and Logic Expressions for Xputers [Ha95]) and has a syntax similar to C. ALEX statements may contain arithmetic and logic expressions, as well as conditional statements. The compilation process of the DPSS executes in three phases:

- Logic optimization and technology mapping
- Placement and routing
- I/O scheduling

The three steps of the compilation process will be discussed in the following:

Logic optimization and technology mapping. In this step, the operators needed by the input application source are identified and selected from an operator library, which holds all operators the rDPUs can execute. As the rDPUs are micro-programmable, also new functions can be added to this library. The current library contains all the operators of the

programming language C, so the operators of the input source code can be mapped without further construction from lower-level functions.

The technology mapping is carried out as follows: The condition statements are converted into single assignment code. Sequences of assignments are considered as basic blocks, from which directed acyclic graphs are constructed. Using this graphs, the code is optimized by removing common subexpressions, identical assignments, local variables, and dead code. Furthermore, constant folding and reduction in strength is used. One-input operators are moved into the next operator if the operator library provides also the merged operator. This step reduces the number of required rDPUs in the KrAA array.

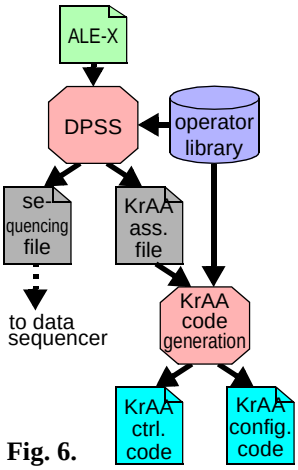

Fig. 6. Overview on the DPSS

Placement and routing. After the operations required by the application are selected, the rDPUs, which will perform these functions, and their interconnects are determined. This step is commonly called placement and routing. Due to the multiplexed KrAA-III buses a valid placement is always routable. A poor placement degrades the performance since some internal variables have to be routed via the I/O bus. During that time the bus is blocked for other I/O operations. Because of the large number of possible placements, a well developed placement algorithm has to be used. Constructive placement algorithms like the cluster growth algorithm are very fast since no iterations have to be performed. But they usually produce a poorer placement than algorithms that use an iterative improvement method. A well developed iterative improvement algorithm is simulated annealing. The simulated annealing algorithm is easy to handle even if a lot of constraints like in the KrAA-III have to be considered. It leads to good results in a relatively small amount of time when using it for small problems as the targeted KrAA-III placement. Further it allows to consider additional routing operations under utilization of the unused local connections in each iteration step. The cost function considers the chip boundaries, the routing operators and with a high cost the connections via the internal I/O bus.

I/O scheduling. The operands and results of the complex function mapped onto the KrAA-III have to be transferred over one global bus. As this bus is restricted to a single operation at one time, the sequence of I/O operations to write or read data to and from the KrAA-III determines the start times of the rDPUs. Therefore, this sequence has a severe impact on the performance. Furthermore, it must be taken care, that the I/O sequence does not violate data dependencies of the complex function. To find such an optimal sequence, an I/O scheduling is performed using a kind of list based scheduling algorithm known

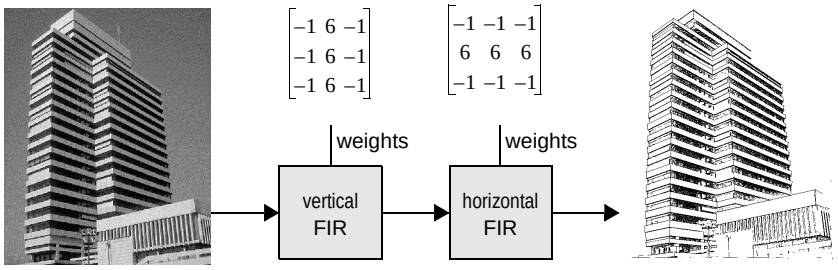

Fig. 7. Edge detection example with appropiate weights

from high level synthesis [GD92].

Optimizations. The DPSS can apply optimizations in terms of area and speed. If the statements of the algorithm belong to an inner loop and are executed several times in direct sequence, pipelining can be used to improve speed on the cost of area. On the other hand, if the statements belong to an outer loop and are executed once or several times but not in direct sequence, the vectorized expressions can be used for different variables and thus improving area use.

5 The Programming of the Kress ALU Array

In this section, the programming of the KrAA-III is demonstrated. In contrast to the synthesis of dedicated hardware, this approach offers a software-only accelerator implementation. That means a hardware designer is no longer needed for implementation. This is performed in using the DPSS (section 4) as a synthesis tool for the KrAA-III. The DPSS benefits from the coarse grained structure of the KrAA-III. While a synthesis system for a FPGA struggles to implement high-level arithmetic operations on a bit-level FPGA structure, the DPSS can map complete arithmetic expressions described in ALE-X directly onto the rDPUs.

$$\bar{w}_{11new} = w_{22}k_{22} + w_{12}k_{12} + w_{02}k_{02} + w_{21}k_{21} + w_{11}k_{11} + w_{01}k_{01} + w_{20}k_{20} + w_{10}k_{10} + w_{00}k_{00}$$

Fig. 8. Equation of 3 by 3 FIR filter

As a computation example, we assume an image processing application, performing an edge detection algorithm, which is implemented by two dimensional FIR (Finite-Impulse-Response, [GW77]) filters with appropriate parameters (figure 7). The basic structure of a 3 by 3 FIR Filter is shown in figure 8a. The edge detection is performed by applying first a 3 by 3 FIR filter which detects the vertical edges and then a second 3 by 3 FIR filter to detect the horizontal edges. The two FIR filters distinguish by the selection of the weights. For each pixel of the image the FIR filter calculates a new value as the weighted sum of all pixel-values in a 3 by 3 area with the current pixel as center. In figure 8a, the w_{xy} are the pixels while the k_{xy} denote the according weights. To perform the desired effect of edge detection, the weights k_{xy} have to be chosen in an adequate way (for example weights see figure 7).

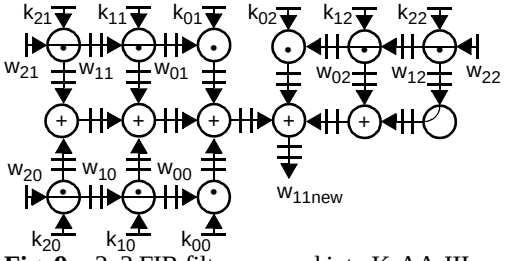

Fig. 9. 3x3 FIR filter mapped into KrAA-III

To implement the filter using the KrAA-III, the C subset ALE-X ([Ha95]) description can be used (figure 8b), allowing a rapid design of the accelerator‘s data manipulation part. The description is processed by the DPSS, which maps the application onto the KrAA-III. The resulting KrAA-III structure of the filter example is shown in figure 9. The calculation of the new value of the center pixel is executed in a mixture of a pipeline and a tree structure.

Figure 9 shows only one possible mapping of the FIR filter example. The sum of the products is performed by several two- and three-input adders. The weights are configured into the register files of the rDPUs nearby the multiplication operators. The multiplications are performed by multiplication- and multiplication-with-data-routing-operators (see figure 3d,e). This operator performs a multiplication with the input data and simultaneously routes the data input to a specified data output. Furthermore one routing operator (figure 3f) is used in the example. Alternatively this routing operator could be

replaced by a global bus routing cycle, but an only routing rDPU is already free for an arithmetic operation.

6 Conclusions

The paper motivates the use of coarse grained field programmable ALU arrays as computing element in a discussion of required features for co-processing units. To illustrate the benefits of coarse grained structures the novel KrAA-III ALU array architecture has been presented. Several new mechanisms (context switch mechanism, local routing capabilities and hierarchic routing structure) for speed up of configurable computing are introduced. It has been shown, that applications for the KrAA-III can be synthesized out of high-level specifications due to its coarse grained structure. The synthesis tool DPSS is capable of mapping algorithms into effective structures on the KrAA-III. The use of the KrAA-III has been demonstrated by an FIR filter implementation.

Like described in section 3, the KrAA-III provides multiple levels of configuration memory, allowing up to four tasks to be configured at one time. This enables configurations of unused layers in parallel to computations in the active layer. Further this feature will be used to implement a multitasking system, which manages the KrAA-III as a computation resource to be assigned to several tasks. Such multitasking operating systems are state of the art for computers based on von Neumann CPUs, providing a comfortable and efficient execution of applications.

7 References

[HB97] R. Hartenstein, J. Becker, M. Herz, U. Nageldinger: An Embedded Accelerator for Real World Computing; VLSI‘97, Gramado, Brazil, August 26-29, 1997.

[HB96] R. Hartenstein, J. Becker, M. Herz, U. Nageldinger: A General Approach in System Design Integrating Reconfigurable Accelerators; Proc. IEEE Int’l Conf. on Innovative Systems in Silicon; Austin, TX, Oct. 1996.

[DH96] Andre DeHon: Reconfigurable Architectures for General-Purpose Computing; Technical Report 1586, MIT Artificial Intelligence Laboratory, September, 1996.

[Ha95] R. W. Hartenstein, et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC’95, Chiba, Japan, Aug. 29 - Sept. 1, 1995.

[YR93] Alfred K.W. Yeung, Jan M. Rabaey: A Reconfigurable Data-Driven Multiprocessor Architecture for Rapid Prototyping of High Throughput DSP Algorithms; HICSS-26, Vol. 1, IEEE Computer Society Press, 1993.

[GD92] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, S. Y.-L. Lin: High-Level Synthesis, Introduction to Chip and System Design; Kluwer Academic Publishers, 1992

[GW77] R.C. Gonzalez, P. Wintz: Digital Image Processing; Addison-Wesley, USA, 1977.