Synthesis and Domain-specific Optimization of KressArray-based Reconfigurable Computing Engines

R. Hartenstein, M. Herz, Th. Hoffmann, U. Nageldinger
University of Kaiserslautern
Dept. of Computer Science, Erwin-Schrödinger-Strasse
67663 Kaiserslautern, Germany
abakus@informatik.uni-kl.de
http://configware.de

Abstract

The paper introduces a concept for reconfigurable hardware platforms along with an according CAD framework. The framework is not restricted to a specific architecture, but is capable of generating mappings for a number of members of the coarse grain reconfigurable KressArray architecture family, and similar platforms. Based on the multi-architecture CAD framework, a design-exploration environment has been set up to find a well suited architecture for a given application domain. The paper describes the architectural and generic principles of the KressArray family as well as the design flow for the determination of a domain-specific optimized KressArray architecture. Furthermore, we present the principles for the embedding of the KressArray into an accelerator engine, which can be used as a co-processor. Also, an overview about the tools in the CAD framework is given.

1. Introduction

While the focus of interest at contemporary PCs is still on the microprocessor used, most of the silicon area within a PC architecture (fig. 1 a) is occupied by accelerators for different tasks, e.g. graphic subsystem, I/O controllers, and others. Considering this misrelation, the microprocessor can be seen as the tail wagging the dog. Increasing design costs and shrinking product lifetimes urge for alternatives like reconfigurable accelerators, which have the advantage of being upgradable by downloading new reconfiguration data - „configware“ (fig. 1 b). The research area of reconfigurable computing has experienced a rapid expansion in the last few years. In many application areas reconfigurable hardware systems have proven to achieve substantial speed-up over classical (micro)processor-based solutions [1]. The range of application areas is extending continuously. Makimoto's third wave, which predicts a new trend to standardized parts (reconfigurable hardware) after a time of customized hardware, appears to become true [2].

The question is, which application area is dominated by the right business model to support a commercial break-through of reconfigurable computing. Currently it is not the Wintel world as people discard hardware frequently. The business model of future smart mobile phones and other more or less portable information appliances favors reconfigurable accelerators - for number of reasons discussed elsewhere [3] [4] [5] [6]. Under the banner "Software Radio" a growing R&D community is fascinated by the idea of totally reconfigurable radio hardware [7] [8]. Several start-ups in silicon valley aim at reconfigurable accelerators in cellular wireless communication and other applications: [9] [10] [11] [12] and others.

However, it turns out that FPGAs do not suit well for computational applications due to several reasons [2] [3]. With coarse grain platforms a drastically higher area-efficiency can be achieved than with FPGAs [2]. Due to the reconfigurability overhead, only about 1% of the transistors on an FPGA chip are used for the active circuitry [13]. In contrast to using FPGAs with single bit Configurable Logic Blocks (CLBs) the area of „Reconfigurable Computing“ stresses „coarse grain" platforms based on reconfigurable Data Path Units (rDPUs) with higher path widths [14] [15] [16] [17] [18]. For computational tasks, coarse-grained architectures offer several advantages over FPGAs:

- Multiple-bit wide datapaths provide more powerful operators and a higher area-efficiency,
- Operators don't need to be assembled from single-bit CLBs (Configurable Logic Blocks).
- The powerful operators support structural compilation from high level programming languages sources like C programs [19] [20] [21] [22], whereas FPGA-based synthesis mostly relies on HDLs (Hardware Description Languages).

- The communication architecture avoids routing congestion by more powerful and gracefully degrading interconnect resources (see section 4).
- Long distance interconnect through coarse grain mesh-connected platforms is *not* expensive - in contrast to (fine grain) mesh-connected FPGAs like, for example, the Algotronix CAL architecture [23].

A number of coarse grain approaches has been published recently, for example the MATRIX project at MIT [14], and the CHESS system by Hewlett Packard [18]. A much earlier coarse-grained reconfigurable architecture example is the KressArray [17], which features arithmetic and logic operators of the C programming language, making the mapping of applications much more simple than for FPGAs.

1.1 Why Domain-specific Compute Engines ?

The authors of this paper believe, that the *universal* coarse grain high performance reconfigurable computing engine would be an illusion. The first barrier on the way to universality is the path width decision. A very wide data path will be a source of area-inefficiency for a majority of applications. A narrow data path, however, will limit the number of applications to be implemented efficiently, or, will cause difficulties in programming due to the need of partially serial or semi-serial implementations.

Especially in many important algorithms of interesting application areas, such as e. g. 3G and 4G cellular wireless communication, or multimedia applications, communication resources are the bottleneck, rather than functional resources. The communication resource requirements widely differ from one application domain to another one. For example, the communication requirements for implementing a high resolution Viterbi decoder on a KressArray or other reconfigurable computing platform are so high, that there is no chance to get it from a universal KressArray. So there is a strong need to have a reconfigurable platform optimized for wireless communication.

1.2 A Generic Family of KressArray-based Compute Engines

This paper introduces a design space exploration methodology to find an optimized reconfigurable hardware platform solution for a particular application area. The design space is provided by a generic family of KressArray architectures. To find an optimum solution to a given application domain a CAD environment called KressArray Xplorer supports experimenting with alternative architectures and applications derived from a source description in a high level language similar to C. An estimator analyses the input and suggests an architecture onto which then the application is mapped by a mapper, which finds good mappings within a few minutes or even faster. A graphic editor supports user interaction. The system generates performance analysis data, which is used by a tool to refine the result iteratively.

This way, several different KressArray architectures can be selected and typical applications, or the main application of the given application domain may be mapped onto each of these arrays. Finally the best architecture can be selected to run a module generator to create an array cell core. By replication and wiring by abutment an array of arbitrary size may be created from such a cell core. The concept described in this paper also supports heterogenous arrays, which include several different types of cells (see fig. 2 i).

The rest of this paper is structured as follows: section 2 briefly summarizes the KressArray architecture. In section 3, a short motivation for design space exploration is given. A design space exploration environment is introduced in section 4, followed by an example in section 5. Finally, the paper is concluded.

2. The KressArray

For better comprehensibility this section summarizes the KressArray principles and its original application development framework DPSS (Datapath Synthesis System) having been published elsewhere [2] [17] [20]. The KressArray (fig. 2) is a regular array of coarse grain reconfigurable processing elements called Data Path Units (rDPUs) [17] [20]. But also other regular schemes are supported like hexagonal and other regular arrays not shown by this paper.

Figure 2, illustrates the communication architecture of the KressArray family by a few rectangular array examples. KressArray layout is based on full custom rDPU cells connected through wiring by abutment using rNN ports (reconfigurable Nearest Neighbor ports) at their northern, eastern, western and southern sides. There are no extra routing areas, so that physically there are no routing resources other than wires and multiplexers inside rDPU cells. Especially, each rDPU can serve as an operator and/or as a routing element (cp. fig. 2 d and e). Row- and column buses (s. examples in fig. 2 f - h) using wires running over the rDPU cells, provide additional communication resources. Either by external wiring or by dedicated row/column buses, different torus structures are possible (s. fig. 2 j - m). A serial global bus, which connects all rDPUs (not shown) is the slowest connection resource. It can be used for long distance connections. In the mapping process, connections over this bus are discouraged. Normally, the exploration process aims at using no global bus connections at all, so that this communication resource can be omitted, given that it is not used for I/O of data to or from the Array.. However, the

software always assumes the existence of a global bus to keep the space of valid solutions as big as possible during the heuristic mapping process.

The rDPU of the original KressArray-I supported the integer arithmetic and logic operators of the programming language C. However, to optimize a KressArray architecture for a particular application area, the functionality of the rDPUs can be tailored to the application domain. Experimenting with alternative domain-specific KressArray architectures supports finding a proper balance between rDPU functionality and array-wide communication architecture to achieve a good silicon area / performance trade-off (also see sect. 2.1). Typical trade-off issues are, for example, sequential multiplication versus physical multiplier, physical floating point operator vs. sequential floating point implementation, or others. The Xplorer environment is sufficiently flexible to support all these alternatives, as well as the choice between different degrees of parallelism, such as e. g. between a fully parallel 32 bit adder and a pipe of 4 rDPUs with 8 bit adders.

2.1 Silicon Area Estimation

The area-efficiency of KressArray cells has been demonstrated by a design study carried out on a CADENCE environment. For example, considering an 0.18 µm CMOS process with 6 metal layers, routing resources in an 18 bits wide rDPU architecture with 8 NNports (2 per side) and 4 backbuses (2 vertical and 2 horizontal) needs 4 metal layers of an area of approximately 0.06 mm^2 (i.e. about 10 mm^2 for a 10 x 16 array, compare fig. 4 and fig. 5). Without exceeding this cell size 2 more metal layers are sufficient for moderately complex functional resources. On the other side, more complex functional resources like multipliers or floating point resources, also provide more cell area for a larger repertory of routing resources, like, for instance, 12, 16, or more NNports.

The silicon area required for a rDPU cell is determined by either functional resources, or routing resources. depending on which of the two needs the larger area. Given a uniform data path width, like, for instance, 18 bits (e. g. 16 bits for data and 2 spare bits for decision data), the silicon area required for routing resources can be roughly estimated by counting the number of rNN ports and the number of row/column buses.

For four rNN ports (one at each side) we define an area unit of 1. Also, a pair of one row bus and one column bus, roughly requires an area of 1 unit. Obviously, the number of rNN ports at opposite sides is always the same. Thus, we denote the number of horizontal and vertical rNN ports by $p_{NNhoriz}$ for the horizontal ports and p_{NNvert} for the vertical ones. The number of row/column buses is given by p_{row} and p_{column} respectively. Then we get:

- The port area factor a_p as $a_p = p_{NNhoriz} \times p_{NNvert}$.
- The row/column bus area factor a_b as $a_b = p_{row} \times p_{column}$.
- The total resource area factor a_t as $a_t = \max(p_{NNhoriz}, p_{row}) \times \max(p_{NNvert}, p_{column})$.

For example, the routing resources of the architecture shown in fig. 2a have an area factor of $a_t = 1$, the one of fig. 2b has $a_t = 4$, fig. 2c has $a_t = 6$, and a combination of fig. 2a with fig. 2g has $a_t = 1$, b with f has $a_t = 4$, and c with h has $a_t = 9$.

2.2 Interfacing the KressArray

KressArray execution is transport-triggered, i.e. the operation starts as soon as all operands are available at the inputs and the output is free. This supports pipelining of rDPUs within a KressArray like known from systolic arrays or wavefront arrays. The KressArray is a generalization of the systolic array (SA). But its array interconnect and its rDPU functionality is reconfigurable instead of being hardwired like in SAs. Due to the use of linear projection methods, SA synthesis only yields linear pipelines and array-wide uniform rDPU functionality, so that reconfigurability would not make sense. Traditional SA synthesis also does not support bus usage, so that data streams can access a SA only via the edges of the array, but not at internal cells.

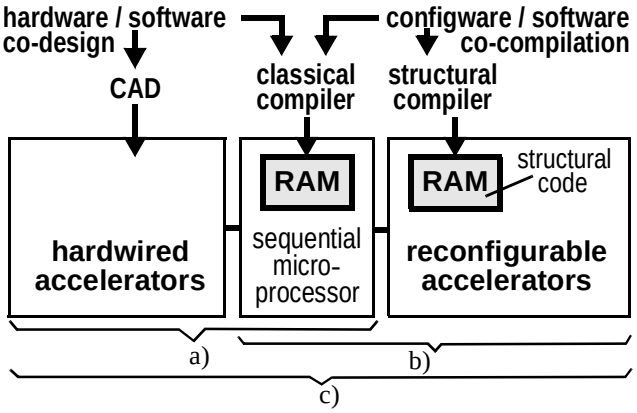

Figure 1: Changing computing system models: a) PC of the Wintel world, b) embedded reconfigurable computing, c) of future smart mobile phones and information appliances.

For mapping applications onto an array the MA-DPSS framework (see section 4) uses a heuristic algorithm being much more flexible than known SA synthesis methods. It supports locally individual rDPU functionality and a wide variety of fully free form pipelines of any regular or irregular shape, also including forks and junctions, as well as feedback data paths. These features

provide high flexibility which supports a wide variety of application areas for KressArray architectures - in contrast to SAs, which are usable only for applications with regular data dependencies.

The data to be processed by a KressArray and the results to be stored back can be transferred to and from the array in three different ways:

- **Via the edges of the array**. This way, which is also known from systolic arrays, supports data stream processing, like in DSP applications.
- **Via internal cells of the array**. Data streams may enter or leave the array also via internal rDPUs, i. e. rDPUs not located at the edge of the array. This situation applies, if previously mapped blocks are used (e.g. library elements), with data ports at specific I/O rDPUs.
- **Via the global bus.** As the global bus can handle only one transfer at a time, this is the slowest way. However, the existence of the global bus guarantees, that there is no mapping failure due to a lack of routing resources (a phenomenon often experienced at FPGA design).

Experiences with KressArrays indicate, that in many application areas the communication resources are the main throughput bottlenecks, rather than functional resources due to slow global bus connections. That's why an entire family of KressArray architectures has been defined as successors to the first KressArray prototype,, which feature substantially more connection resources. For example, with 4 (fig. 2 a), 8 (fig. 2 b), 10 (4 horizontal and 6 vertical, see fig. 2 c) bidirectional rNN ports, where "bidirectional" means that a direction is selected at configuration time.

2.3 Organizing Array I/O Data Streams

Like SA synthesizers the DPSS determines, in addition to placement and routing, also the I/O data sequence. This task is carried out by the scheduler tool (s. fig. 3). SA literature hardly describes how to arrange data in memory to implement these data sequences (what is usually up to the software of some host interfaced to the SA). For memory-intensive KressArray applications, however, a data sequencing methodology has been published elsewhere ([24] [25] [26]), which supports multiple GAG[1]-DMA[2] path interconnect with memory banks surrounding the array, preferably on the same chip. The GAG-DMA is more than a DMA for block transfers just with address incrementing: it supports a rich repertory of generic address sequences. There are three ways to implement a GAG Data Sequencer:

- by programming a controller inside the KressArray, if available [17] [20],
- by a rDPU especially designed for application domain *and* sequencer configuration,
- by a special sequencer rDPU in a heterogenous architecture ([25] [26], compare fig. 2 i)

All 3 methods are supported by the KressArray Xplorer. A GAG-based memory communication architecture supporting interleaved or burst mode multiple memory banks and its KressArray application have been published elsewhere [27] [28] [29].

3. From Compilation to Exploration

For application development from high level language sources reconfigurable computing requires novel compilation techniques. As structural code (non-procedural configware code instead of software code) has to be generated, classical compilation techniques cannot be applied (compare fig. 1). Another important difference between reconfigurable computing systems and conventional procedural systems is the communication paths setup time. In procedural systems, which include also parallel computing systems the communication paths are set up at run time. In reconfigurable computing systems, the data paths and the communication links are set up at compile time, (or loading time, respectively).

At the University of Kaiserslautern, the first version of the Data Path Synthesis System (DPSS) [20] for the KressArray-I [17], which generates configware code from a high level language source, has been introduced in 1995. The DPSS, with a mapper tool as structural code generator, is embedded in the CoDe-X co-compiler framework supporting software / configware partitioning [2] [21] [22]. Since data path reconfiguration can be seen as a kind of instruction fetch at compile time, machines with a reconfigurable ALU using instruction sequencers don't make sense. Thus, the CoDe-X framework performs a paradigm shift from instruction sequencing to the data sequencing methodology mentioned above.

An important lesson learnt from the KressArray-I and its DPSS mapper indicates, that the goal of a fully universal reconfigurable computing machine platform based on a single design of a "universal" array cell (rDPU, see section 2) seems to be an illusion. Obviously much more architectural flexibility is desirable to provide optimized domain-specific platforms. That's why an entire family of KressArray architectures has been defined, rather than

1). Generic Address Generator
2). Direct Memory Access

a single architecture. From this family, an optimized structure can be selected for a given application domain by experimenting with different architectures. For this design space exploration task a novel application development framework is needed, which is much more flexible than the DPSS.

The KressArray design space explorer (KressArray Xplorer), being implemented at Kaiserslautern, supports experimenting with alternative solutions. When a suitable architecture has been found, the structural code to run this architecture is also provided. Furthermore, the explorer system is capable of generating a Verilog description of the resulting array for simulation with commercial tools.

4. Domain-specific Optimization by Design Space Exploration

As shown in section 2, the KressArray structure defines an architecture class rather than a single architecture. The class members differ by the available communication resources, which have a considerable impact on the application performance. In contrast to designing with FPGAs the mapper introduced here always finds a valid mapping as long as there are enough rDPUs available due to global bus usage. Instead of routing congestion, as known from FPGAs, there may be only performance degradation by bus cycle overhead. But this throughput degradation can be reduced by changing the architecture, resulting in a trade-off between communication resources (which are bound to more chip area) and performance. The actual resource requirements depend heavily on the application domain. For example, systolic array applications rely on nearest neighbor connections, while Turbo Code or Viterbi channel decoding algorithms are dominated by long-distance interconnects.

To assist the user in finding the best suitable architecture for a given application domain, an interactive design space exploration environment called KressArray Xplorer is being implemented at Kaiserslautern University. The

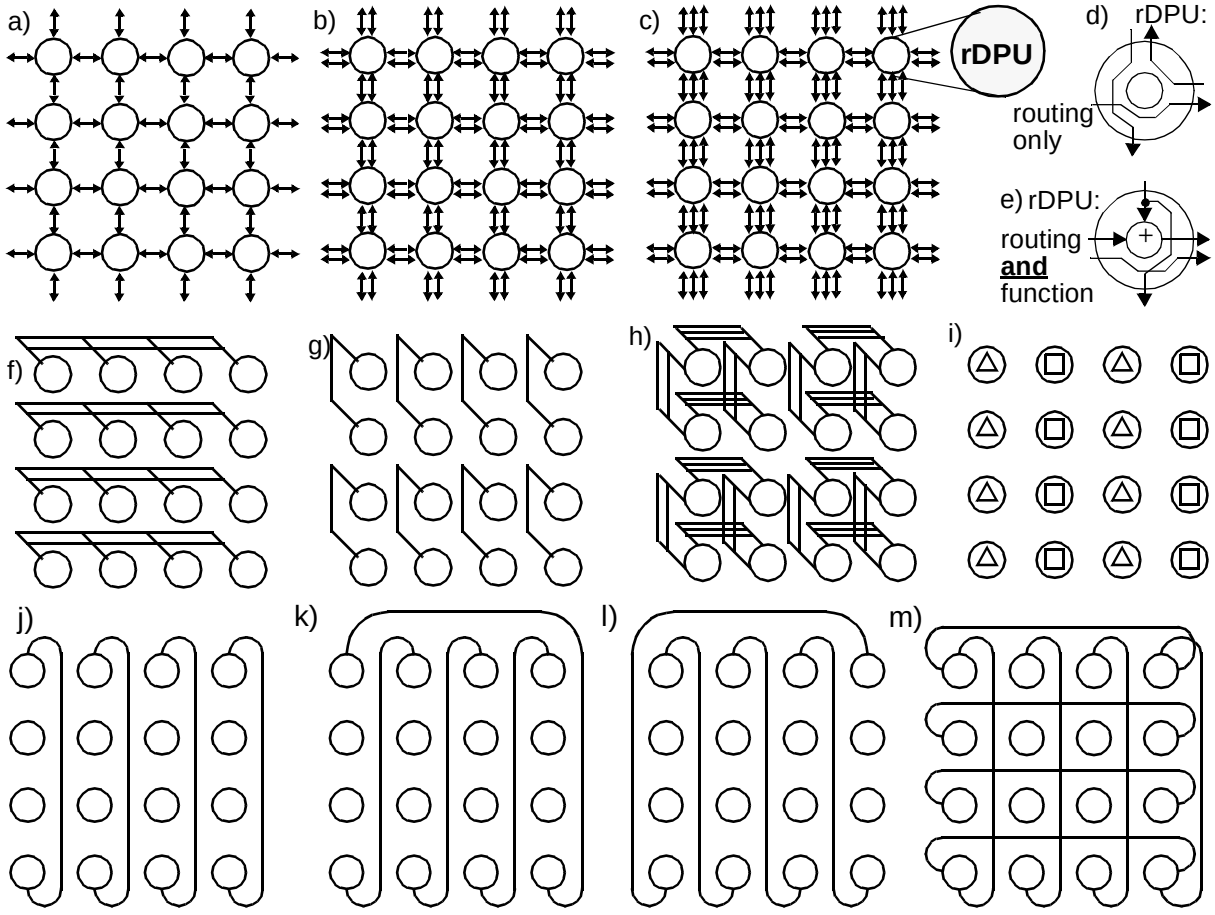

Figure 2: KressArray communication architecture by examples: a) 4 reconfigurable nearest neighbor ports (rNN ports), b) 8 rNN ports, c) 10 rNN ports, d) reconfigurable Data Path Unit (rDPU, compare fig. c), use for routing only; e) rDPU use for function and routing, f) 2 back buses per row, g) segmented single buses per column, h) 2 per column, 3 per row,i) different function sets in alternating columns, j - m) some vertical torus structure examples: j) no shift vertical; k) vertical shift right (to next column), l) shift left (to previous column), m) double torus.

Xplorer is implemented in C/C++ and is running on UNIX and LINUX. The KressArray Xplorer is based on the MA-DPSS (Multi-Architecture Datapath Synthesis System) design framework for KressArrays, which is used to map datapaths described in the high level ALE-X language [20] onto the a KressArray. ALE-X statements may contain arithmetic and logic expressions, conditions, and loops, that evaluate iterative computations on a small number of input data. In contrast to the earlier DPSS (Datapath Synthesis System) framework [17], the MA-DPSS can handle a large variety of different architectures.

An overview of the KressArray Xplorer is given in figure 3. The user provides a description of the application using the high level ALE-X language [20]. The ALE-X compiler creates an intermediate format, which is used by all other tools. At the beginning of the exploration process, the minimal requirements for the architecture are estimated and added to the intermediate format. Then, the user can perform consecutive design cycles, using the mapper and the data scheduler. Both tools generate statistical data, which is evaluated by an analyzer to make suggestions for possible architecture enhancements, which are presented to the user by an interactive editor. This editor is also used to control the design process itself. When a suitable architecture has been found, a HDL description (currently Verilog) can be generated from the mapping for simulation. In the next subsection, the design space for the KressArray architecture class will be described shortly. In the following sections, the tools of the environment will be briefly sketched.

4.1 The KressArray Design Space

The architectural properties of a specific KressArray architecture are specified in the intermediate file format together with the expression tree of the application and the mapping. The tools of the KressArray Xplorer can handle the following architectural aspects:

- The size of the array.
- The available repertory of nearest neighbor connections (for examples see figure 2 a - c). The numbers of horizontal and vertical connections can be specified individually for each side and in any combination of unidirectional, bidirectional links.
- The torus structure of the array. The torus structure can be specified separately for each nearest neighbor connection. In figure 2 j - l, the possible structures for a vertical connection are shown. The external connections can be implemented by wiring on the PCB, if the KressArray is built from several chips, or by assigning dedicated row/column buses for an array on a single chip. Torus connections support regular data dependencies, which can be found e.g. in systolic arrays [30].
- The number of row and column buses (for examples see figure 2 f - h). These buses connect several rDPUs in a row or a column. The buses may be segmented (for examples see figure 2 g and h). It is possible to specify the maximal writer count for a set of buses.
- Areas with different rDPU functionality. For example, in figure 2 i, the rDPUs in every second column of the array have a different function set available.
- The maximum length of routing paths for nearest neighbor connections.
- The number of routing channels through a rDPU (for examples see figure 2 d and e).
- Peripheral port location constraints to support particular communication architectures connecting the array to surrounding memory and other circuitry. It is e.g. possible to specify, that an input port for data has to be mapped to a certain edge of the array, in a certain range of rDPU positions.
- Port grouping: Peripheral ports at the edges can be grouped to express, that a common bus is used for these ports. This influences the I/O scheduling.
- Particular placements or groups can be frozen (modular mapping). Such array areas will stay fixed to their position during the mapping process. This way, optimal mappings for parts of the datapath (e.g. library elements) can be included in an application.

4.2 The ALE-X compiler

The ALE-X compiler of the MA-DPSS has been taken over from the original DPSS. Its task is to generate the expression tree, which describes the datapath and is the basis for the mapping process. The data created by the compiler is independent of the architecture. Though the compiler writes a default architecture to the output file, this information is overwritten in the architecture estimation step. The compilation takes place in three main phases: First, the ALE-X input is analyzed and parsed, and directed acyclic graphs are constructed from the basic blocks of the input program. Several optimizations are performed to reduce the number of required rDPUs, including the removal of common subexpressions, identical assignments, local variables, and dead code. In the technology mapping phase, the operators of the input program are selected from the operator library of the KressArray, which is described in a separate hardware configuration file.

4.3 The Architecture Estimation

This step determines a start architecture for the exploration process. By examining the expression tree from the ALE-X compiler, the minimal requirements for the KressArray architecture in terms of communication resources is estimated. The estimation is based on the degree of the tree vertices and the complexity of the tree.

4.4 The Mapper

The mapper tool maps the application datapath onto the KressArray, performing a placement and routing step. It can handle different architectures of the design space. The mapping algorithm is based on simulated annealing, including a router for the nearest neighbor connections. The output of the tool is a file in the intermediate format, with the mapping information added. Also, statistical information is produced, which can then be analyzed to enhance the architecture.

All parameters necessary for the mapping process are taken from the input file, which is also in the intermediate format. These parameters consist of the architectural properties and parameters to control the annealing process, e. g. the starting temperature, the number of iterations, and the end temperature. As the mapper produces as output the same intermediate file format as the input, several annealing steps may be performed consecutively, e.g. a low-temperature simulated annealing after a normal mapping.

For each communication resource, as well as for the global bus, the costs for a connection can be specified, which together make up the cost function for the annealing. This way, specific resources can be discouraged. Typically, one would assign the global bus a high cost factor, since those connections are slow due to the serial character of this bus. If row or column buses are available, they would normally get a medium cost level, while nearest neighbor connections have the lowest cost, as they are the preferred connections. With this strategy, the mapper will eventually find a mapping, which does not use expensive interconnect resources at all. Such connection resources can then be removed for the next iteration step in the design space exploration process, leading to a more effective implementation of the KressArray.

4.5 The Scheduler

The scheduler determines an optimized array I/O sequence of the data words from and to the array. It can handle the various architectures and produce schedules for both global bus and row/column buses, if there are any. Additionally, the performance of the mapping is estimated based on delay information specified in a separate hardware file. Like the mapper, the scheduler produces statistical data, which are evaluated by the analyser.

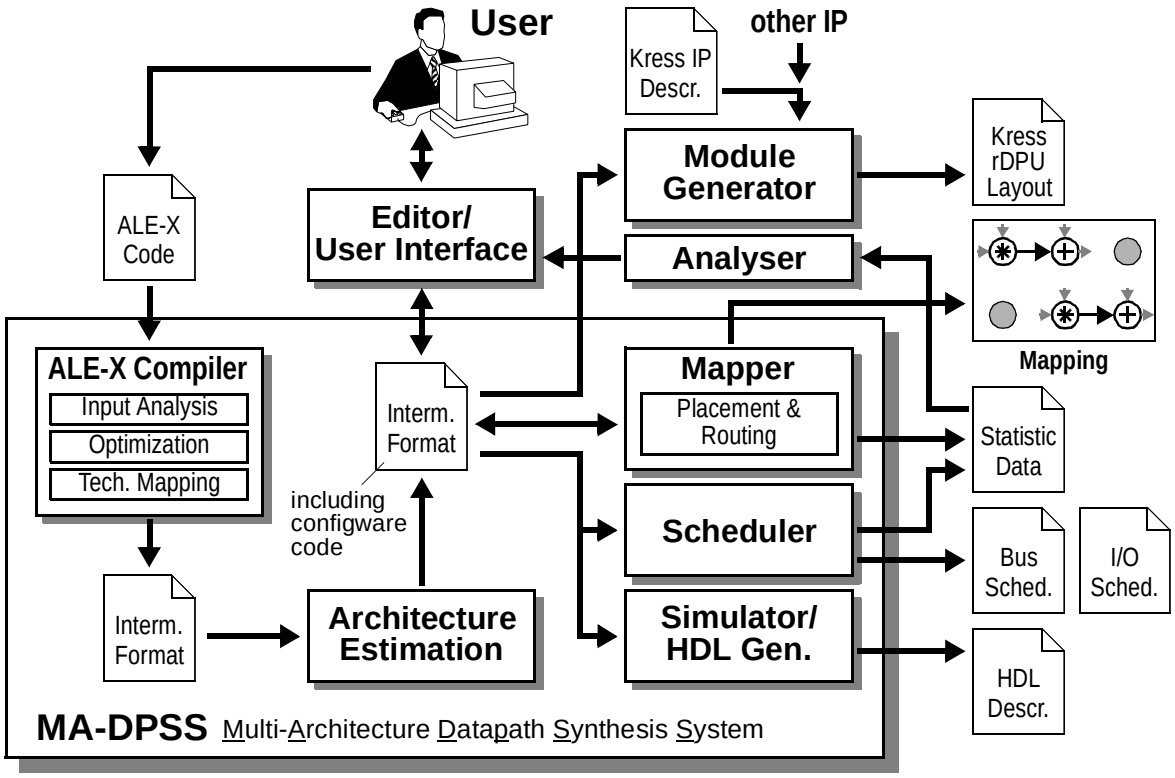

Figure 3: KressArray Xplorer overview.

4.6 The Analyser

The analyzer tries to suggest enhancements for the current KressArray architecture, based on data gathered during the mapping and scheduling. These informations include e.g. the average use of the nearest neighbor connections for each rDPU, critical path length, reasons for nearest neighbor routing failures and other. The main task of the analyzer is to identify bottlenecks in the current architecture.

4.7 The Editor

The Xplorer environment includes an interactive graphical editor which allows three types of manipulations: definition of architectural parameters, direct modification of mapping results, and tuning of the optimization parameters (i.e. the parameters for the simulated annealing and the cost function). The editor's features include defining or changing the data path width and repertory of rDPU operators, and, selecting the routing resource architecture, as well as setting port location constraints to optimize the array embedding in other parts of the application circuitry and surrounding RAM cores. To support libraries of pre-mapped modules, or to optimize architecture for communication between the array and surrounding RAM banks, the editor allows moving cells or groups of cells to a new position within the array, or freezing their locations (see Section 4.1).

4.8 The HDL generator / simulator

When a suitable KressArray architecture is found, the application and the architecture can be simulated by the Xplorer environment. The simulator is also capable of generating a HDL description (currently Verilog) for simulation with external tools, and, for export to physical and other design environments.

4.9 The KressArray rDPU Module Generator (planned)

For the KressArray Xplorer system also a module generator is planned, which accepts two classes of IP core library cells:

- cells for communication routing resources inherent to the KressArray (see fig. 3).
- other cells, mainly function cells from other module generators or external IP sources (see fig. 3).

The KressArray rDPU module generator extracts relevant structural data from the mapping results stored in ".map" internal format and assembles 6 metal layer CMOS layout of a KressArray rDPU cell. From this rDPU layout the array can be easily synthesized via iterative cell abutment.

4.10 Xplorer Use for CHESS and Other Platforms

Future work being planned is the extension of the environment by additional configware code generators, so that other platforms, like e. g. the CHESS system [18] by Hewlett Packard will be supported.

5. An Image Processing Example: SNN Filter

To demonstrate the effect of different architectures on the resulting mapping, an example using the rather complex datapath of an image processing filter is given in this section. The tested architectures are shown in shown in fig. 2 b and one similar to the one in fig. 2 c.

The Symmetric Nearest Neighbor (SNN)-Filter [31] is used as sharpening filter in object detection. The Xplorer result output of the first mapping is shown in fig. 4. The algorithm uses 157 rDPUs and has been mapped onto a 10-by-16 KressArray. In the chosen architecture each PE provides two bidirectional nearest neighbor connections to each of its four neighbor-rDPUs.

There are nine array input ports and one array output port (shaded boxes at left side in fig. 4), which were all put at the western side of the array, leaving the exact placement up to the mapper (the output port is the fifth one from top). Note that these ports are just positions at the array edges. They do not resemble physical rDPUs, although the figure might give this impression. As the nearest-neighbor resources are used up in some places, additional global bus connections are necessary. These can be identified by the blocks in the upper right corner of the rDPUs. The mapping in fig. 4 uses 16 bus connections, which is a moderate value for a datapath of such a complexity. However, for the I/O, no global bus connections were needed. Also, the neighbor connections have a quite good utilization, which is also caused by the fact, that in this algorithm many output values of rDPUs are used multiple times. E. g. eight of the nine input values are used at four different PE inputs. The mapping took about 24 minutes on a Pentium-II 366-MHz PC.

The KressArray Xplorer enables the designer to map his algorithm on different architectures and to compare the resulting structures. Depending on the chosen KressArray architecture the mappings will differ in many points like the number of used rDPUs, the used communication resources, etc. With this information the designer can decide his compromise between complexity (and costs) of the base-architecture and the execution time.

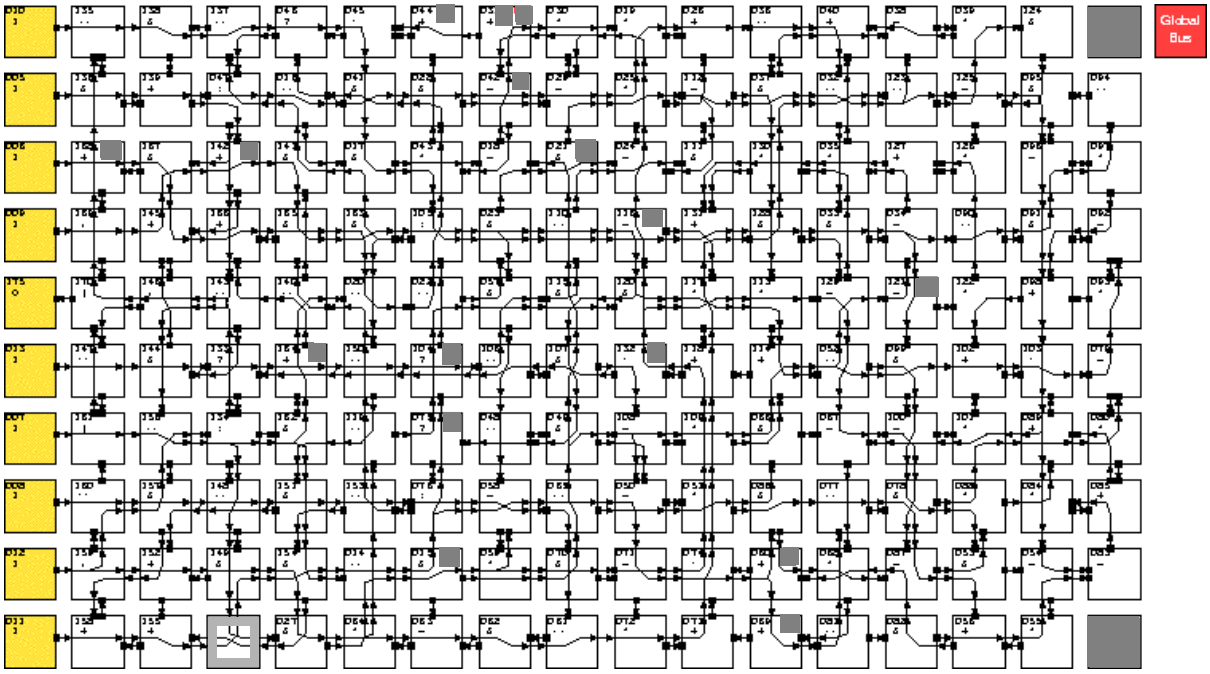

Figure 4: Mapping the SNN filter on rDPUs with 8 NN ports, 16 backbus connects for (legend: see figure 5).

In the example above an alternative architecture could be the same KressArray (10-by-16) with three vertical nearest-neighbor connections *(NN links)* and two horizontal connections. Then the number of required rDPUs is still 157 because the number of operations remains the same and the operators provided by the rDPUs are also unchanged. But, due to the enhanced routing capabilities of the rDPUs, some connections which were realized by the global background bus *(back bus)* in the first mapping can now be routed using NN links. Thus, in the second mapping only seven bus connections are needed, which leads to a better performance of the resulting structure.

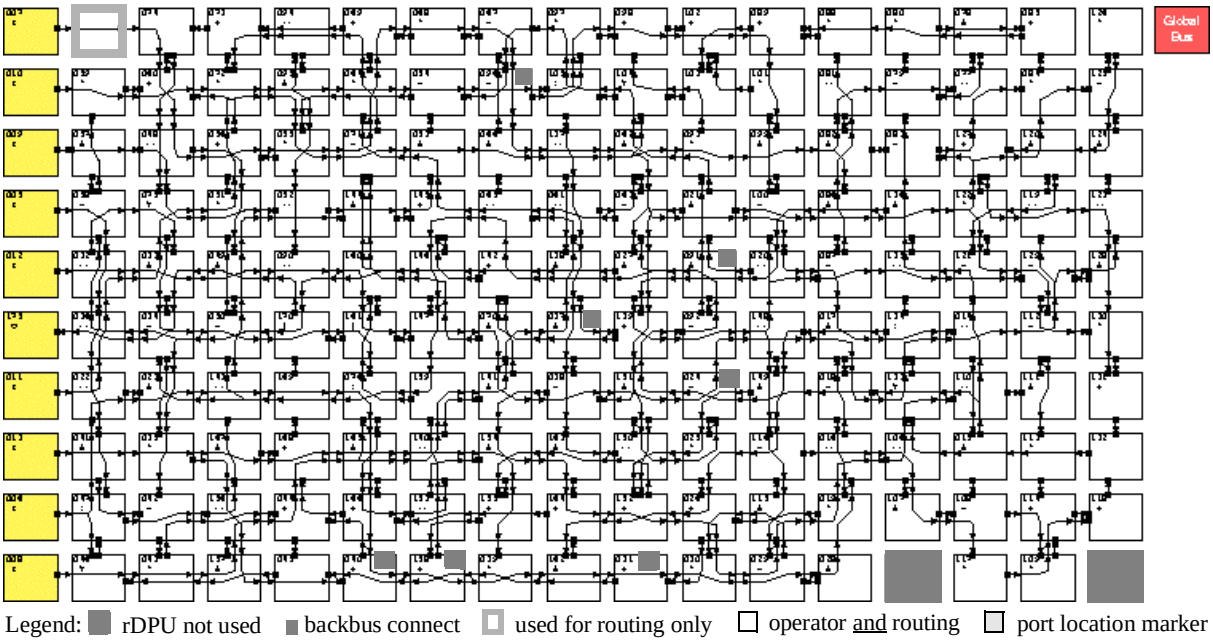

Figure 5: KressArray Mapping of an SNN filter on rDPUs with 10 NN ports: only 7 backbus connects are needed.

6. Conclusions

A generic family of coarse grain reconfigurable architectures, the KressArray family, has been introduced. The KressArray allows simpler and more area-efficient application mapping than fine-grained FPGAs. An overview about a design space exploration environment for KressArray architectures has been given. The exploration environment assists the user in finding an optimized architecture for a given application or application domain. A given application can be specified in a high level language. An architecture estimator and an analyzer provide initial architectures, and, suggestions for enhancements, respectively. The mapping of the application is done using a simulated annealing based tool without the need of user interaction. However, a graphical editor is provided for fine-tuning the mapping and controlling the exploration process, whenever desired. KressArray architectures can be simulated directly, or externally by using an included Verilog generator.

It has been demonstrated, that also massively communication-intensive applications can be successfully mapped onto mesh-connected coarse grain reconfigurable hardware, so that a highly area-efficient solution is obtained. It has been shown, that due to Xplorer use the KressArray is a generalization of the systolic array. Thus, as a by-product, a new systolic array synthesis method has been obtained and implemented, which is substantially more flexible than methods having been published earlier.

Literature

1. W. Mangione-Smith, et. al.: Seeking Solutions in Configurable Computing; Computer, Dec. 1997
2. J. Becker et al.: Parallelization in Co-Compilation for Configurable Accelerators; Asian South Pacific Design Automation Conference 1998 (ASP-DAC'98), Yokohama, Japan
3. R. Hartenstein (invited paper): The Microprocessor is no longer General Purpose: why Future Reconfigurable Platforms will win; ISIS'97, Austin, Texas U.S.A., Oct. 1997.
4. R. Hartenstein (invited talk): Reconfigurable Computing: Taking off to Overcome the Microprocessor; parc forum, Xerox PARC, Palo Alto, CA, May 13, 1999; http://www.parc.xerox.com/ops/projects/forum/1999/forum-05-13.html
5. R. Hartenstein (opening keynote): Next Generation Configware merging Prototype and Product; RSP'98, Int'l Workshop on Rapid System Prototyping, Leuven, Belgium, June 3-5, 1998 (unpublished)
6. R. Hartenstein (invited talk): On the Application of the KressArray to Rapid Prototyping; DATE'98, Paris, France, February 23 - 26, 1998
7. J. Mitola: The Software Radio Architecture, IEEE Communications Magazine, May 1995
8. Y. Shinagawa: Software Radio Technologies; 1st Int'l Software Radio Workshop, Rhodes, June 1998
9. http://www.morphics.com
10. http://www.silicon-spice.com
11. http://www.chameleonsystems.com
12. http://www.malleable.com
13. A. DeHon: Reconfigurable Architectures for General Purpose Computing; report no. AITR 1586, MIT AI Lab, 1996
14. 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.
15. E. Waingold et al.: Baring it all to Software: Raw Machines, IEEE Computer 30, pp. 86-93.
16. C. Ebeling, D. Cronquist, P. Franklin: RaPiD: Reconfigurable Pipelined Datapath, Workshop on Field Programmable Logic and Applications, FPL'96, Darmstadt, Germany, 1996.
17. R. Kress: A Fast Reconfigurable ALUs for Xputers; Ph. D. thesis, Univ. Kaiserslautern, 1996.
18. A. Marshall et al.: A Reconfigurable Arithmetic Array for Multimedia Applications; FPGA'99, Int'l Symp. Field Programmable Gate Arrays, Monterey, CA, Febr. 21 - 23, 1999
19. A. Agarwal, S. Amarasinghe, et al.: The Raw Compiler Project; Proceedings of the Second SUIF Compiler Workshop, Stanford, CA, August 21-23, 1997.
20. R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC'95, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995.
21. J. Becker: A Partitioning Compiler for Computers with Xputer-based Accelerators; Ph.D. thesis, Univ. Kaiserslautern, 1997
22. K.Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. thesis, Univ. Kaiserslautern, 1994
23. N. N.: The Configurable Logic Data Book; Algotronix, Inc., Edinburgh, UK, 1990
24. R. Hartenstein, A. Hirschbiel, K. Schmidt, M. Weber: A Novel Paradigm of Parallel Computation and its Use

to Implement Simple High-Performance Hardware; InfoJapan'90- Int'l Conf. memorising the 30th Anniversary of the Computer Society of Japan, Tokyo, Japan, 1990
25. R. Hartenstein, A. Hirschbiel, K. Schmidt, M. Weber (invited reprint of [24]): A Novel Paradigm of Parallel Computation and its Use to Implement Simple High-Performance Hardware; Future Generation Computer Systems 7 91/92, p. 181-198
26. R. Hartenstein, J. Becker, M. Herz, U. Nageldinger: A Novel Universal Sequencer Hardware; Proc. ARCS'97 (Architekturen v. Rechensystemen), Rostock, Germany, Sept. 8-11, 1997.
27. R. Hartenstein, M Herz, T Hoffmann, U Nageldinger: Using the KressArray for Configurable Computing; Conf. on Configurable Computing: Technology and Applications, Boston, Nov. 2-3, 1998, Proc. SPIE Vol. 3526
28. R. Hartenstein, J. Becker, M. Herz, U. Nageldinger: An Embedded Accelerator for Real World Computing; in Proc. IFIP Int'l Confer. on Very Large Scale Integration, VLSI''97, Gramado, Brazil, Aug. 26-29, 1997
29. R. Hartenstein, J. Becker, M. Herz, U. Nageldinger: A Novel Sequencer Hardware for Application Specific Computing; Proc. 11th Int'l Conf. on Application-specific Systems, Architectures and Processors, ASAP'97, Zurich, Switzerland, July 14-16, 1997
30. H.T. Kung: Why Systolic Architectures?; IEEE Computer, Jan. 1982.
31. D. Harwood et al.: A New Class of Edge-preserving Smoothing Filters; Pattern Recog. Lett. 6, pp. 155-162, Aug.1987