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, Pacifico Yokohama, Yokohama, Japan, January 25-28, 2000.

KressArray Xplorer: A New CAD Environment to Optimize Reconfigurable Datapath Array Architectures

R. Hartenstein, M. Herz, Th. Hoffmann, U. Nageldinger

Computer Structures Group, Informatik
University of Kaiserslautern
D-67663 Kaiserslautern, Germany
Tel.: +49 631 205 2606
Fax: +49 631 205 2640
hartenst@rhrk.uni-kl.de
http://configware.de

Abstract— This paper introduces a CAD environment for design-space exploration of coarse grain reconfigurable KressArray architectures and similar platforms. To find an optimal solution to a given application domain this KressArray Xplorer supports experimenting with different architectures. An estimator analyses the input and suggests an architecture onto which then the application is mapped using an heuristic algorithm. A graphic editor supports user interaction. The system generates performance analysis data, which is used by an automatic tool to refine the result iteratively.

I. Introduction

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].

However, it turns out that FPGAs do not suit well for computational applications due to several reasons [3]. Thus, several coarse grain reconfigurable platforms have been developed, which use multiple-bit wide operators instead of single-bit configurable logic blocks. Examples for coarse grain approaches are the MATRIX project [4], the RAW Machines [5], the CHESS platform [8], and the RaPiD project [6] and the KressArray [7]. The KressArray features arithmetic and logic operators at C language level, making the mapping of computational applications much simpler than for FPGAs. The original version KressArray-I is accompanied by the programming framework DPSS (Datapath Synthesis System) [10], which allows the development of applications from high level language sources.

Experiences with the KressArray-I indicate, that a fully universal reconfigurable computing machine platform based on a single design of a “universal” array cell, which is well suited for any application, seems to be an illusion. Thus, an entire family of KressArray architectures has been defined, featuring different architectural characteristics. From this family, an optimal structure can be selected for a given application domain by experimenting with different architectures.

To support the finding of a suitable architecture for a given application domain, the KressArray design space explorer (KressArray Xplorer) has been developed. It supports experimenting with different architectures, allowing a judgement of the architecture efficiency for a given application. When a suitable solution has been found, the structural code to run the architecture is also provided. Furthermore, the explorer system is capable of generating a Verilog description of the resulting array for simulation with commercial tools. This paper gives an overview about the concepts of the KressArray Xplorer system. The Xplorer incorporates some tools, which also existed in the DPSS, but were considerably extended in their functionality to handle different architectural properties.

The rest of this paper is organized as follows: Section II briefly summarizes the KressArray having been published elsewhere [2][7][10]. In section III the KressArray Xplorer will be introduced, which incorporates the enhanced MA-DPSS (Multi-Architecture DPSS). The design space exploration is demonstrated by an example in section IV. Finally, the paper is concluded.

II. The KressArray

An architecture of the KressArray family is a regular array of coarse grain reconfigurable Data Path Units (rDPUs) [7] of multiple-bit data path width. Figure 1 illustrates the communication ressources 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 north, east, west, and south sides (fig. 1a,b). Each rDPU cell may be used to implement an operator and simultaneously to route data words through the array (fig. 1c,d). Additional communication ressources can be provided by row-/ or column buses (fig. 1e,f). Further, all rDPUs are connected by a serial global bus (not shown in the figure). In fig. 1, the communication ressources are drawn in separate pictures for readability. However, they may be combined arbitrarily.

To optimize a KressArray architecture for a particular application area, the functionality of the rDPUs can be tailored to the demands of this application domain. Often, a tradeoff has to be found between the silicon area and the performance of the operators, resulting in different implementations of a certain operator. Examples for tradeoff issues are sequential or parallel multiplication or the realisation of complex operators like MAC (Multiply-Accumulate) by one single or several rDPUs.

A. Silicon Area Estimation

A design study has been carried out on a CADENCE environment to estimate the chip area for a KressArray cell. Assuming a 0.18 µm CMOS process, the routing resources of one cell in an 18 bits wide rDPU architecture with 8 NNports (2 per side), 2 row and 2 column buses need 4 metal layers of an area of approximately 0.06 mm^2 (i.e. about 10 mm^2 for a 10 x 16

array). Without exceeding this cell size 2 more metal layers are sufficient for moderately complex functional resources.

However, generally speaking, the silicon area required for a rDPU cell is determined by functional resources or by 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, assuming there is at least one horizontal and vertical port. 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 $a_b = \max(1,p_{row}) \times \max(1,p_{column})$.
- 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. 1a have an area factor of $a_t = 1$, the one of fig. 1b has $a_t = 6$, a combination of fig. 1a with fig. 1e has $a_t = 1$, and fig. 1b with fig. 1f has $a_t = 9$.

B. Interfacing the KressArray

The execution of KressArray operators 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 [15] or wavefront arrays. In this sense, the KressArray can be seen as a generalization of the systolic array (SA). But its array interconnect and its rDPU functionality is reconfigurable instead of being hardwired like in SAs.

However, like an SA, the KressArray needs a stream of data to process, which has to be provided from outside. The

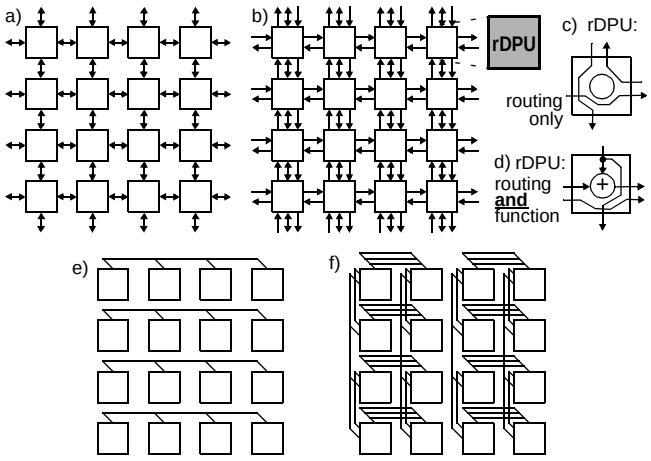

Fig. 1: KressArray communication architecture examples: a) 4 nearest neighbor ports, b) 10 rNN ports, c) reconfigurable Data Path Unit (rDPU), use for routing only; d) rDPU use for function and routing, e) single row bus, f) several segmented row/column buses

data to be processed 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.

C. Organizing Array I/O Data Streams

Like SA synthesizers, the synthesis system for KressArrays, which is incorporated in the KressArray Xplorer (see section III) defines, in addition to placement and routing, also the I/O data streams. To generate these streams, a data sequencing methodology has been published elsewhere ([13] [14]), which is based on address sequences to one or more memory banks connected to the array, preferably on the same chip. There are three ways to implement a GAG Data Sequencer:

- by programming a controller inside the rDPU, if available,
- by a rDPU especially designed for application domain and sequencer configuration,
- special sequencer rDPU in a heterogenous rDPU type architecture

All 3 methods are supported by the KressArray Xplorer described below. The third one is supported by the feature which allows to use different types of rDPU cells in the same array. A data-sequencer-based smart memory communication architecture supporting interleaved or burst mode multiple memory banks, as well as its KressArray application have been published elsewhere [11][12][14].

III. Design Space Exploration

As shown in section II, the KressArray structure defines an architecture class rather than a single architecture. The class members differ by the available communication resources, and the functionality of the operators, which both have a considerable impact on the application performance. In contrast to designing with FPGAs the mapper described here always finds a valid mapping as long as there are enough rDPUs available, due to global bus usage. Thus, instead of routing congestion, there may be only performance degradation by bus cycle overhead. 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 algorithms from mobile communication are dominated by long-distance interconnects.

The discussion above implies that there is no general KressArray architecture for any application. Instead, for a given application domain, there is a trade-off between performance and chip-area, depending on a particular rDPU functionality, and, on a specific communication architecture (also determined by rDPU cell design), that will meet the requirements of the application best. 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 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 [10] onto 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 framework [10], the MA-DPSS can handle a large variety of different architectures..

An overview of the KressArray Xplorer is given in fig. 2. The user provides a description of the application using the high level ALE-X language [10]. 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.

A. 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 function sets of the rDPUs. This allows more primitive architectures, where complex operators may be assembled from several rDPUs.
- The available repertory of nearest neighbor connections (cf. fig. 1). The numbers of horizontal and vertical connections can be specified individually for each side and in any combination of unidirectional or bidirectional links.

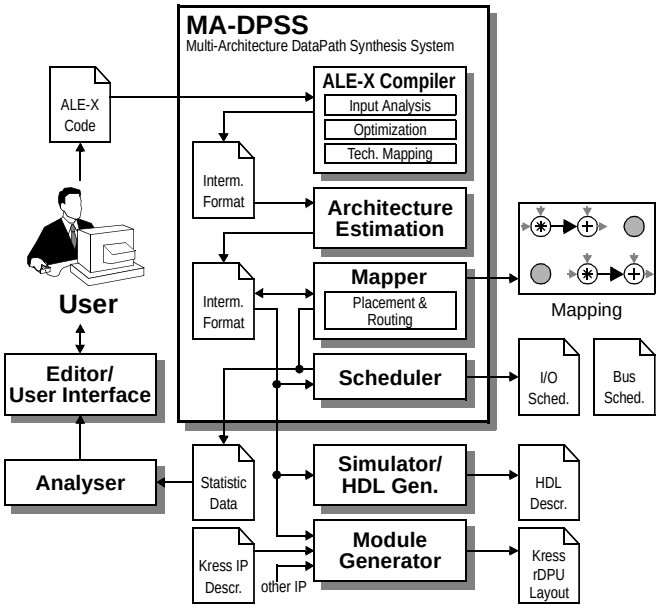

Fig. 2: KressArray Xplorer Overview

- The torus structure of the array. The torus structure can be specified separately for each nearest neighbor connection. 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.
- The number of row and column buses (cf. fig. 1e,f). These buses connect several rDPUs in a row or a column. The buses may be segmented (fig. 1f). It is possible to specify the maximal writer count for a set of buses.
- Areas with different rDPU functionality. For example, a complex operator like multiplication may be specified to be available only in certain rDPUs.
- The maximum length of routing paths for nearest neighbor connections.
- The number of routing channels through a rDPU (cf. fig. 1c,d).
- 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 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.

B. 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 inde-

pendent 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.

C. 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 architecture in terms of communication resources is estimated based on the degree of the tree vertices and the complexity of the tree.

D. 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 analysed 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, the use of 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.

E. 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.

F. The Analyser

The analyser 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 analyser is to identify bottlenecks in the current architecture.

G. 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.

H. 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.

I. The KressArray rDPU Module Generator (planned)

For the KressArray Xplorer system also a module generator (cf. fig. 2) is planned, which accepts two classes of IP core library cells:

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

The KressArray rDPU module generator extracts relevant structural data from the mapping results stored in the 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.

IV. 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 Symmetric Nearest Neighbor (SNN)-Filter [16] is used as sharpening filter in object detection. The Xplorer result output of the first mapping is shown in fig. 3. 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.

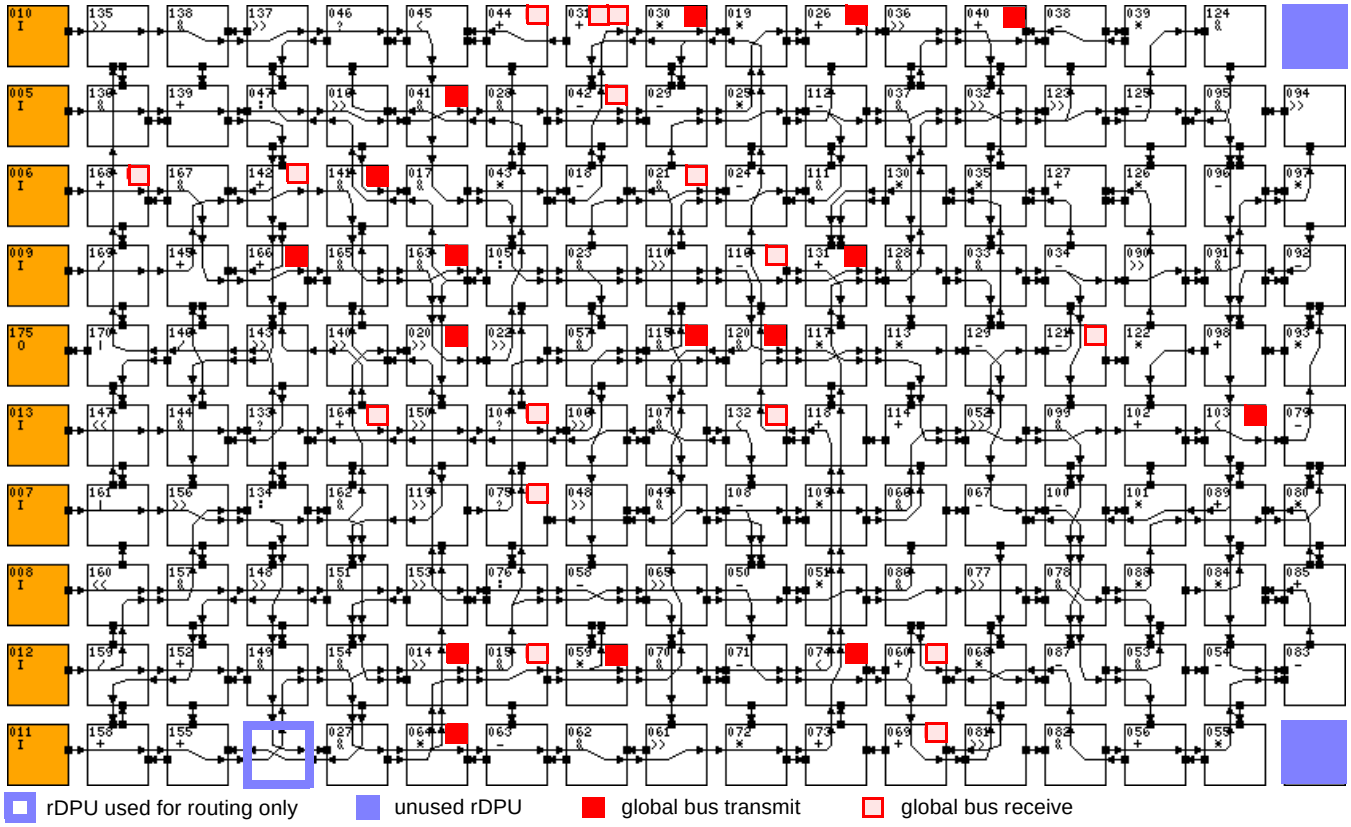

Fig. 3: Mapping the SNN filter on rDPUs with 8 NN ports using 16 global bus connects.

There are nine array input ports and one array output port (special boxes at left side in fig. 3), which were all put at the western side of the array, leaving the exact placement up to the mapper again (the output port is the fifth one from top). Note, that these boxes in the figure denote no rDPUs but just the positions, where the data words enter and leave the array. As the nearest-neighbor resources are used up in some places, additional global bus connections are necessary. These can be identified by the small boxes in the upper right corner of the rDPUs. The mapping in fig. 3 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 rDPUs. 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.

In the example above an alternative architecture could be the same KressArray (10-by-16) with three vertical nearest-neighbor connections 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 bus in the first mapping can now be routed using NN links. This explains why in the second mapping only seven bus connections are needed.

The SNN example raises an important question regarding the granularity of the configurable architecture. For the example mappings, the datapath description was fed directly in the DPSS system. However, the direct mapping is not always optimal. E.g. for the color separation, two rDPUs were used, one for a shift operation and the next for masking out the according value. It would not increase the complexity of the rDPUs to provide a function, which combines these two operations, resulting in less PE usage.

V. Conclusions

An overview about a design space exploration environment for KressArray architectures has been given. The KressArray is a coarse-grained reconfigurable architecture family, which allows much simpler and much more area-efficient application mapping than fine-grained FPGAs. The exploration environment assists the user in finding the best suitable architecture for a given application or application domain. A given application can be specified in a high level language. An architecture estimator and an analyser provide initial architectures and suggestions for enhancements. 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 to for fine-tuning the mapping and control-

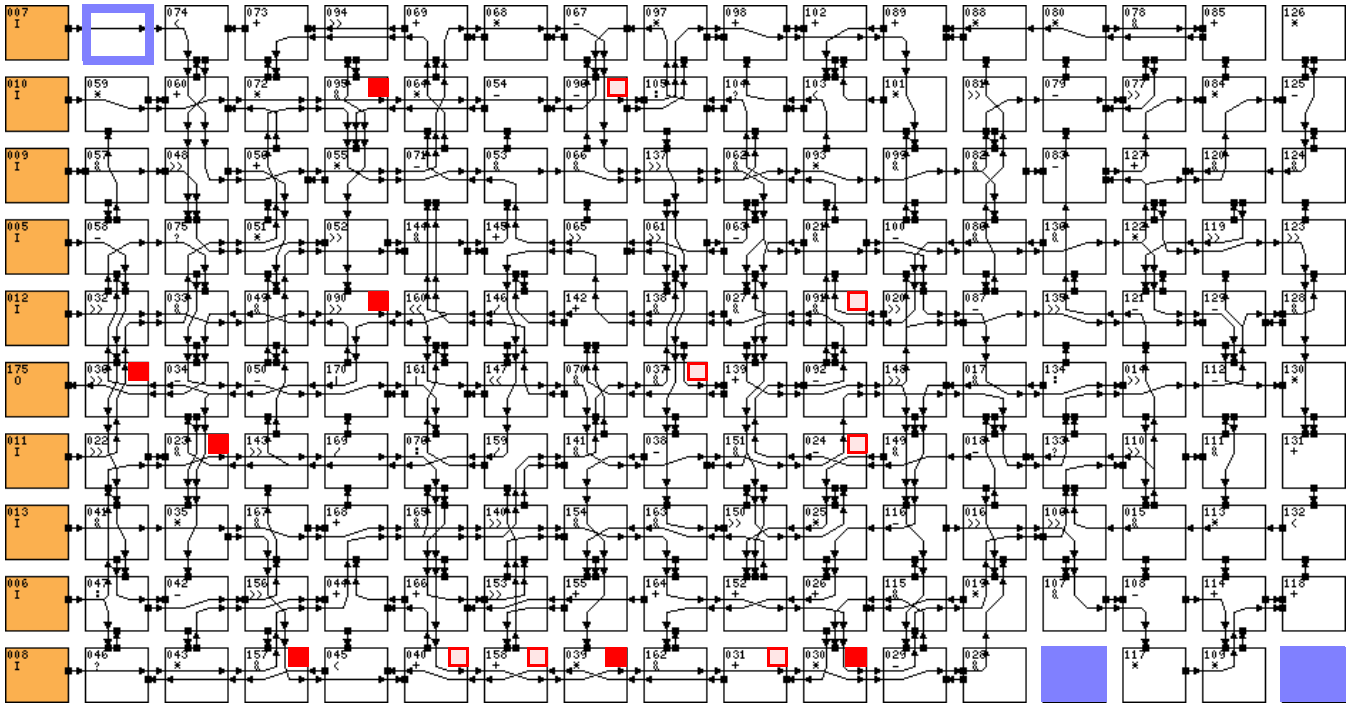

Fig. 4: KressArray Mapping of an SNN filter on rDPUs with 10 NN ports, using 7 global bus connects. See fig. 3 for a legend.

ling the exploration process, whenever desired. KressArray architectures can be simulated directly, or externally by using a HDL generator being included.

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. 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 by Hewlett Packard will be supported.

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] 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.

[5] E. Waingold et al.: Baring it all to Software: Raw Machines, IEEE Computer 30, pp. 86-93.

[6] C. Ebeling, D. Cronquist, P. Franklin: RaPiD: Reconfigurable Pipelined Datapath, Workshop on Field Programmable Logic and Applications, FPL'96, Darmstadt, Germany, 1996.

[7] R. Kress: A Fast Reconfigurable ALUs for Xputers; Ph. D. thesis, Univ. Kaiserslautern, 1996.

[8] 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

[9] A. Agarwal, S. Amarasinghe, et al.: The Raw Compiler Project; Proceedings of the Second SUIF Compiler Workshop, Stanford, CA, August 21-23, 1997.

[10]R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC'95, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995.

[11]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

[12]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

[13]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

[14]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

[15]H.T. Kung: Why Systolic Architectures?; IEEE Computer, Jan. 1982.

[16]D. Harwood et al.: A New Class of Edge-preserving Smoothing Filters; Pattern Recog. Lett. 6, pp. 155-162, Aug.1987